DATA EFFICIENCY IN NAMED ENTITY RECOGNITION

by

Ryan Rene Moreno

Submitted to the University of Southern California in partial fulfillment

of graduation requirements for engineering honors

Viterbi School of Engineering, Computer Science

May 2020

Research Advisor: Dr. Xiang Ren

Signature: _____

Honors Advisor: Dr. Andrea Armani

Signature: _____

UNIVERSITY OF SOUTHERN CALIFORNIA

DEPARTMENTAL APPROVAL

of a honors undergraduate thesis submitted by

Ryan Rene Moreno

This thesis has been reviewed by the research advisor, the engineering honors advisor, and the dean of the Viterbi school of engineering and has been found to be satisfactory.

Dr. Xiang Ren, Research Advisor

Dr. Andrea Armani, Honors Advisor

Dr. Yannis Yortsos, Dean of Viterbi

ACKNOWLEDGMENTS

Thank you to Yuchen (Bill) Lin, the Ph.D. student who mentored me on both the entity trigger and data augmentation projects. Thank you for answering my questions and keeping me actively involved in the projects. I would also like to thank Dongho Lee, the masters student I worked with closely on the entity trigger project. Finally, thank you to Dr. Xiang Ren for the opportunity to work in his research lab.

DATA EFFICIENCY IN NAMED ENTITY RECOGNITION

Abstract

by Ryan Rene Moreno, Undergraduate Honors Thesis
University of Southern California
May 2020

Research Advisor: Dr. Xiang Ren

Named entity recognition (NER) is a fundamental information extraction task attempting to extract named entities from a given text and classify them using pre-defined categories (e.g. persons, locations, or organizations) (Nadeau and Sekine, 2007). The standard protocol for obtaining an annotated NER dataset involves an annotator viewing a sentence, selecting token spans as entities, and labeling the spans with their entity types. Neural NER models are powerful in settings in which an abundance of ground-truth human-annotated training data is available (Lample et al., 2016). However, human annotation is expensive and time-consuming, especially in technical domains in which the human annotator must be able to understand and contextualize technical jargon. Thus, a crucial research question is how we can most efficiently collect and use human-annotated ground-truth data.

In this thesis, I investigate two potential answers. First, I explore a novel technique for collecting human-annotated data by gathering both ground-truth entity labels and *entity triggers* – the phrases which cued the annotator that there was an entity. The entity triggers are used as additional supervision to train NER models. Our experiments demonstrate that annotating both the entities and triggers is more efficient than annotating only

entities, allowing for state-of-the-art results using a significantly smaller amount of human annotations.

Second, I explore automated methods to develop adversarial data, data which is created by perturbing a correctly classified example, causing the model to misclassify the adversarial example. Adversarial examples are useful because they demonstrate the limits of state-of-the-art NER models which can guide future research directions for making models more robust. My experiments demonstrate that the algorithms I develop to generate adversarial examples are effective at fooling state-of-the-art NER models. Further, I use these augmentation algorithms for adversarial training, in which the same adversarial techniques used to augment the test data and attack the model are used to augment the training data and train the model. My experiments show that adversarial training using my adversarial augmentation techniques improves the NER model's robustness to adversarial attacks while retaining its effectiveness on the original test data sets. By developing these adversarial examples from existing human-annotated data in an automated manner, I efficiently use the annotations, garnering the benefits of adversarial attacks and adversarial training without additional human input.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Introduction

1.1 Named Entity Recognition (NER)

1.1.1 Task Definition

Named entity recognition (NER) is a fundamental information extraction task that attempts to extract entities from a given text and classify them using pre-defined categories (e.g. persons, locations, or organizations) (Nadeau and Sekine, 2007).

Modern approaches to NER use machine learning, using a set of human-annotated, labeled sentences to train a model to classify new sentences. The main dataset I use in this thesis is CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003). This dataset uses a Beginning-Inside-Outside (BIO) format, meaning that each word is classified as either `B-TYPE` – the beginning of a named entity, `I-TYPE` – any word in a named entity that is not the first, or `O` – a non-entity. Separating `B-TYPE` and `I-TYPE` helps to distinguish entity boundaries. An example of a labeled sentence appears in Fig 1.1.

More formally, let $\mathbf{x} = [x^{(1)}, x^{(2)}, \cdots, x^{(n)}]$ denote a sentence in the labeled training corpus $\mathcal{D}_L$. Each labeled sentence has a NER-tag sequence $\mathbf{y} = [y^{(1)}, y^{(2)}, \cdots, y^{(n)}]$, where $y^{(i)} \in \mathcal{Y}$ and $\mathcal{Y}$ can be {`O`, `B-PER`, `I-PER`, `B-LOC`, `I-LOC`, $\cdots$}. Thus, we have $\mathcal{D}_L = \{(\mathbf{x_i}, \mathbf{y_i})\}$, and an unlabeled corpus $\mathcal{D}_U = \{\mathbf{x_i}\}$ that we want to label.

In the standard setup, human-annotators are asked to annotate a collection of sentences,

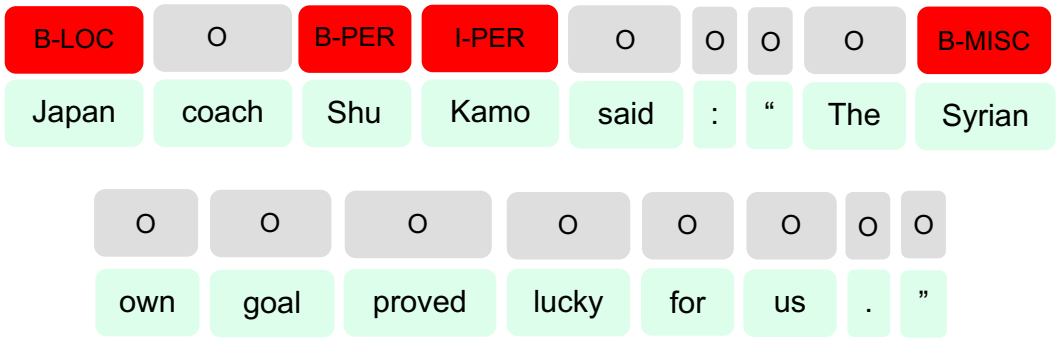

Figure 1.1 Example of a human-annotated sentence demonstrating the BIO format.

which are split into a training, dev, and test set. The CoNLL-2003 dataset is made up of 14,987 training sentences, 3,466 dev sentences, and 3,684 test sentences (Tjong Kim Sang and De Meulder, 2003). The modern approach to NER is using neural models. Sec 1.1.2 briefly explains the training process of a neural network. During training, the neural NER model has access to each sentence in the training set and its corresponding word labels. The dev set is also used during training. However, the model doesn't directly train on the dev set, but rather uses the dev set to test the model's performance between each training stage. The model predicts labels for the sentences in the dev set, comparing its predictions to the human-annotated labels and adjusting its parameters accordingly. Finally, after the model is completely trained it makes predictions on the sentences in the test set. These predictions are compared to the human-annotated labels, producing an official evaluation of the trained model. The model's performance on the test set is expected to reflect the performance it would have on completely unlabeled sentences.

NER models are evaluated using three different metrics: precision, recall, and an F1 score. Precision measures the rate of false positives by only considering the named entities that the model predicts. Precision is the percentage of the model's predictions that were correct (Tjong Kim Sang and De Meulder, 2003). In the case of NER, "positive" means that the word is labeled as a named entity.

$$\text{P} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Recall only considers the named entities in the dataset. Recall is the percentage of the named entities that the model predicted correctly (Tjong Kim Sang and De Meulder, 2003). Precision and recall are depicted in Fig 1.2.

$$\mathrm{R} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

The F1 score is a combination of precision and recall that is more heavily influenced true negatives, false negatives, and false positives than true negatives (Riggio, 2019). As one can imagine, there should be many true negatives in a NER dataset since the majority of words are non-entities. Because the dataset is more weighted towards negative examples, the F1 score is more effective as an overall measure of the model's performance than a simple accuracy score (the percentage of the model's labels that are correct). The formal definition of the F1 score is as follows:

$$\mathrm{F1} = \left(\frac{\text{precision}^{-1} + \text{recall}^{-1}}{2}\right)^{-1}$$

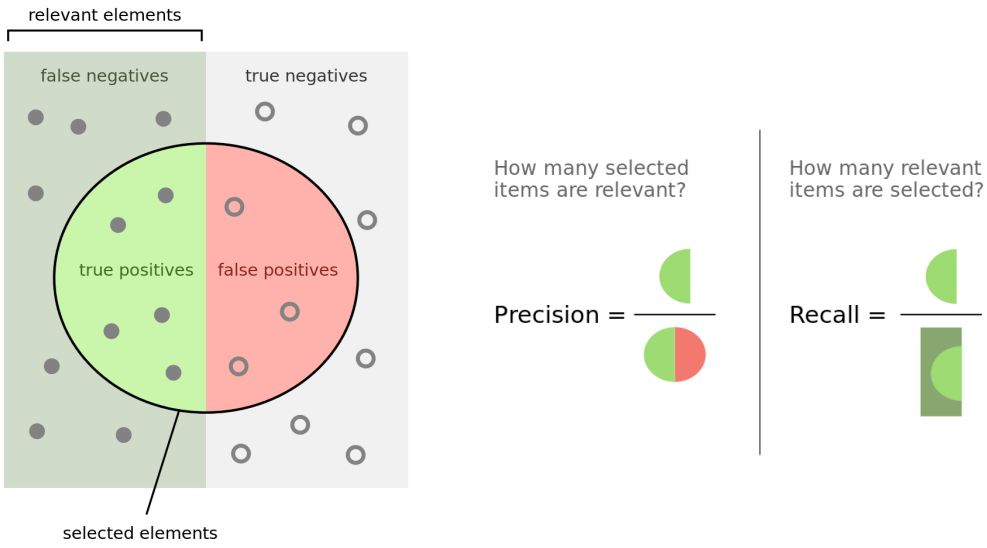

Figure 1.2 A visual depiction of precision and recall. Figure from Riggio, 2019.

1.1.2 Neural NER Frameworks

Early NER models used hard-coded rules that specified the steps for extracting named entities (Yadav and Bethard, 2019). Because of their dependence on pre-defined rules, these models were not robust. The next generation of models began to employ machine learning but notably, not deep learning (Yadav and Bethard, 2019). These models relied on hand-crafted feature engineering. Although an improvement from rule-based approaches, feature engineering still requires researchers to predefine which features are important to their NER model. Examples of such features include which words are capitalized in the sentence, the part of speech of each word, and the frequency of each word within the entire corpus (Nadeau and Sekine, 2007).

Deep learning using neural networks offers an improvement on these previous approaches by abstracting the definition of rules and features. Neural Networks attempt to model the power of the brain by representing neurons as linear threshold functions that act on a small piece of the input data. By connecting these neurons together, with their weights as trainable parameters, neural networks can represent non-linear functions that act on input data to produce an output prediction. Neural networks learn their non-linear function by repeatedly making predictions on the training data, verifying their predictions, and adjusting the weights of the neurons accordingly. In this way, neural networks learn to make predictions without hand-crafted rules or features; they simply need to be given a large set of training data with the correctly labeled outputs. Since 2011, state-of-the-art NER models have employed neural networks, which outperform previous rule-based and feature-based approaches even without access to external resources and domain-specific knowledge (Yadav and Bethard, 2019).

Our Framework

The basic framework used in this thesis is the most common neural NER architecture, a BLSTM-CRF built on both word and character-level embeddings (Ma and Hovy, 2016).

The overall structure of this framework is shown in in Fig 1.3. This framework is end-to-end differentiable, meaning that all of the parameters (in the word and character embeddings, the BLSTM, and the CRF) are trained towards the final NER performance.

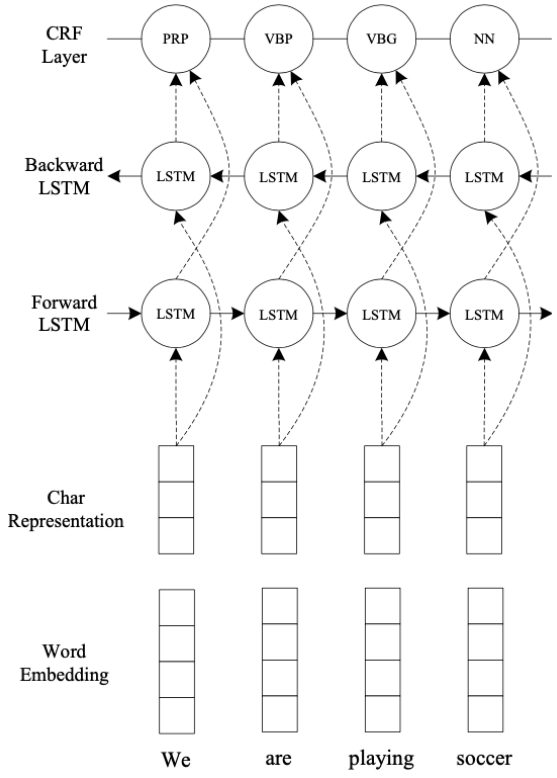

Figure 1.3 Overall architecture of neural NER network. Not pictured are the lookup table for word embeddings and the CNN resulting in character embeddings. Figure from (Ma and Hovy, 2016).

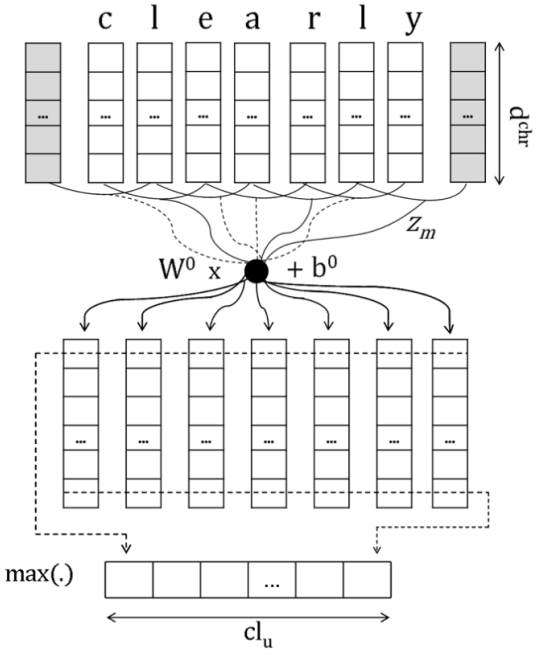

Figure 1.4 Creating character embeddings using a CNN. Figure from (Dos Santos and Zadrozny, 2014).

The first step in our framework is creating word and character embeddings. The purpose of word embeddings is to represent the words semantically; an embedding is a vector representations of a word in a semantic vector space. This means that words that have a small distance from eachother in the vector space should have a similar meaning. We use a lookup table to map each word in our corpus to its embedded vector. In order to speed up the training process, we pre-train the word embeddings using Stanford's publicly available

GloVe embeddings (Pennington, Socher, and Christopher D. Manning, 2014). The GloVe embeddings were trained using 6 billion words from Wikipedia and other text on the internet. The meaningful substructure of the resulting vector space is evidenced by increased performance using GloVe embeddings on a variety of natural language processing (NLP) tasks, including NER and analogy tasks asking "a is to b as c is to _?". By pre-training our word embedding network with GloVe embeddings, the lookup table begins with vectors in the meaningful vector space of the GloVe embeddings rather than starting with a random initialization.

Concurrent with the creation of word embeddings is the creation of character embeddings. Although word embeddings are important for their semantic representation of a word, they leave out useful information regarding word morphology, the actual characters within a word. For example, if a word has the suffix "ion," we can infer that it is probably a noun, while a word ending in "ing" is likely a verb. We follow the lead of Dos Santos and Zadrozny, 2014, who use a convolutional neural network (CNN) to create character embeddings, something that had previously been done via handcrafted feature engineering. As depicted in Fig 1.4, the characters themselves are passed into the CNN as vectors (representing if the character is a "c" or an "H" for example). These input vectors are then convolved, meaning that a matrix operation is performed over every window of k neighboring vectors. Convolution is beneficial because characters are morphologically useful in the context of their neighbors. We then extract a single character embedding vector for the word using max pooling over the convolved vectors.

After creating the character and word embeddings, the two vectors are concatenated and fed into a bi-directional long-short term memory (BLSTM) neural network (Ma and Hovy, 2016). When making labeling predictions, a vanilla neural network would only have access to the word in question. Because it would not have access to any of the surrounding words, a vanilla neural network is unable to take advantage of contextual clues in the sentence. LSTMs provide access to previous words in the sentence by using the previous words' LSTM

outputs as inputs, proportionally "forgetting" the most distant words. In NER, it is useful to have access to both the past words and the future words in a sentence. Because of this, we use a bi-directional LSTM. As depicted in Fig 1.3, a BLSTM consists of a forward LSTM (which evaluates the sentence from left to right) and a backward LSTM (which evaluates the sentence from right to left). When evaluating a given word, the forward LSTM has access to the past context while the backwards LSTM has access to the future context. The outputs of the forward and backward LSTMs for a given word are concatenated to form the final output of the BLSTM for that word.

Finally, we are ready to perform label prediction on our word. If we directly used the BLSTM output to predict our labels, we would have to make each word's prediction individually. We prefer to jointly label the words in a sentence because their labels are highly correlated. For example, it is highly unlikely that a word tagged as an organization entity would be directly followed by a person entity without any non-entity words in between. In order to jointly label the words of a sentence, we use a conditional random field (CRF) as our final network layer (Ma and Hovy, 2016). Rather than choosing the highest probability label for an individual word, the CRF maximizes the log-likelihood of the sentence's entire sequence of labels. This can be done efficiently using the Viterbi algorithm (Viterbi, 1967).

Another concept that is important to our framework is attention. Attention allows an external context vector to scale a word's hidden state vector, emphasizing some dimensions while demphasizing others. Given an input word's hidden state $\vec{h_t}$ and word-level context vector v^T as well as trainable parameters $\mathbf{W}$ and $\vec{b}$, the attention vector $\vec{\alpha_t}$ is computed as follows (Rome, 2018).

$$\vec{u_t} = \tanh(\mathbf{W}\vec{h_t} + \vec{b})$$

$$\vec{\alpha_t} = \text{SoftMax}(v^T\vec{u_t})$$

Geometrically, $\mathbf{W}$ and $\vec{b}$ rotate and scale the word's hidden state vector. Then, the tanh function stretches components of the vector. The external context vector v^T represents the relative importance of each dimension of the new vector $\vec{u_t}$. Taking the dot product of v^T and $\vec{u_t}$ scales each component of $\vec{u_t}$ by its dimension's weight in v^T. Because W and $\vec{b}$ are trainable parameters, the model learns how to manipulate the word's hidden state vector so that its dimensions are most effectively scaled by v^T. Finally, the SoftMax function turns the resulting attention vector into a probability distribution, normalizing each element.

Attention is particularly useful when we want to incorporate external information because this information can be included as the context vector (as we do in Sec 2.3.2). However, attention can also be used in the absence of external information through self-attention. In self-attention, a word's hidden state vector is itself used as the context vector $v^T = h_t^T$ (Z. Lin et al., 2017). The resulting attention vector focuses on a specific semantic component of a sentence (such as a set of related words). Because a sentence often has multiple important components, self-attention can be computed k times with a different set of parameters for each, resulting in an attention matrix.

1.2 Problem Statement

Neural NER models are powerful in settings in which an abundance of ground-truth training data is available (Lample et al., 2016). However, human annotation is expensive and time-consuming, especially in technical domains, such as biomedical publications, in which the human annotator must be able to understand and contextualize technical jargon. Unfortunately, neural NER model performance decreases significantly when training data is restricted (Lee et al., 2020). Thus, a crucial research question is how we can most efficiently collect and use human-annotated ground-truth data. In this thesis, I investigate two potential answers. First, I explore a novel way of collecting human-annotated data by gathering both ground-truth entity labels and *entity triggers* – the phrases which cue entities. This ap-

proach allows for state-of-the-art results using a smaller set of human annotations. Second, I explore automated methods to create adversarial data to evaluate models and additional training data without any additional human input.

1.2.1 Entity Triggers

Our approach

As a method to more efficiently collect human annotations, I worked with USC's Intelligence and Knowledge Discovery Research Lab to introduce *entity triggers* (Lee et al., 2020). We hypothesize that collecting human explanations of the annotators' choices in entity labels could provide a more label-efficient learning of NER models. An entity trigger is a group of words in a sentence that helps to explain why a human would recognize an entity in the sentence. For example, in the sentence "He ate lunch at IHOP," *ate lunch at* is an trigger phrase for the entity **IHOP**. After collecting human annotations for both entities and entity triggers we train a neural network to predict entity triggers in unlabeled sentences. We use these predicted entity triggers as additional supervision for the final goal of predicting entities. Our experiments demonstrate the cost effectiveness of using entity triggers. Training our framework on only 20% of the trigger-annotated training sentences results in a comparable performance to conventional approaches using 70% of the training data.

Related Works

Towards low-resource learning for NER, recent works have mainly focused on dictionary-based distant supervision, using a dictionary of entities and an unannotated target domain corpus to generate weakly labeled data, data which we have less confidence in because it was not annotated by hand. This approach uses exact character matchings of words in the unannotated corpus with entities in the entity dictionary, regarding the hard-matched sentences as additional, weakly labeled data for learning a NER model. Shang et al., 2018 and Yang

et al., 2018 follow this approach, employing Partial CRFs to tentatively assign unlabeled words all possible labels, choosing the pattern that maximizes the total probability. One drawback to their approach is that they rely on a domain-specific seed dictionary. Cao et al., 2019 use Wikipedia anchors, assumed to be named entities, as their entity dictionary. Using the Wikipedia anchors, they automatically construct weakly labeled data, splitting them into high-quality and noisy data. Then, they train a classification module, which regards name tagging as a multi-label classification problem. This module uses noisy data first and fine-tunes with high-quality data. After pre-training this classification module, they share the overall neural network with the sequence labeling module. Then, they use a sequence labeling module to infer the named entity. This paper does not rely on a domain-specific seed dictionary, but it relies on Wikipedia. One drawback is that their weak labeling method doesn't consider the context of a sentence. Zhu et al., 2019 similarly construct weakly labeled data from Wikipedia and DBpedia. Using a small amount of human annotated data, they implement a semi-supervised correction model with curriculum learning to correct the false-negative entity labels in the weakly labeled data. Finally, they use the corrected data to train a neural NER model. They show the effectiveness of curriculum learning using weakly labeled data, but it can only be applied to general NER tasks such as CoNLL. It cannot be applied to domain-specific tasks such as medical text.

Although the dictionary-based approaches largely reduce human efforts in annotating, the quality of matched sentences is highly dependent on the coverage of the dictionary and the quality of the corpus. The learned models tend to have a bias towards entities with similar surface forms as the ones in the dictionary. Without further tuning under better supervision, these models have low recall. Unlike these works aiming to get rid of training data or human annotations, our work focuses on how to more cost-effectively utilize human efforts.

Another line of research which also aims to use human efforts more cost-effectively is active learning (Shen et al., 2018; B. Y. Lin et al., 2019). This approach focuses on instance

sampling and the human annotation UI, asking workers to annotate the most useful instances first. However, a recent study (Lipton and B. C. Wallace, 2018) argues that actively annotated data barely helps when training new models.

Inspired by recent advances in learning sentence classification tasks using explanations (i.e. human-written rules) (S. Li, Xu, and Lu, 2018; Hancock et al., 2018; Wang et al., 2020; Zhou et al., 2020) such as relation extraction, we propose the concept of an "entity trigger" for NER. These prior works primarily focus on sentence classification, in which the rules are continuous token sequences and there is a single label for each input sentence. The unique challenge in NER is that we have to deal with rules which are discontinuous token sequences and there may be multiple rules applied at the same time for an input instance. Our work using entity triggers sheds light on future research directions for more cost-effectively learn NER models.

1.2.2 Generating Adversarial Data

Adversarial Examples

An adversarial example for a neural network is an example that is created by perturbing a correctly classified example, causing the model to misclassify the adversarial example. Adversarial examples are easily understood within the image domain, where a slight rearrangement in the pixels of an image results in a new image that is almost indistinguishable to a human (and thus would be classified the same as the original image). A classic example in pop-science is the slight rotation of an image of a tabby cat. Google's image classifying network is over 80% confident the original image is a cat. However, with the slight rotation, the network becomes 90% confident that the same image is a bowl of guacamole (Athalye et al., 2017). Adversarial examples in the natural language domain are slightly more complicated because rearranging words or characters in a sentence doesn't always result in a sentence that is syntactically valid, let alone semantically similar to the original sentence. I

explore several ways to perturb sentences such that the original entity labels and the sentence structure both remain valid.

Adversarial examples are useful because they demonstrate the limits of state-of-the-art NER models. By creating adversarial data to effectively trick a NER model, we develop a better idea of the model's brittleness, which can guide future research directions for making the model more robust. Adversarial examples are particularly important because test data is usually collected in a similar manner as the training and dev data, so models are biased towards that particular writing style (Ribeiro, Singh, and Guestrin, 2018). When used in real-world applications, using brittle models can be dangerous.

My experiments demonstrate that the algorithms I develop to generate adversarial examples are effective at fooling state-of-the-art NER models.

Adversarial Training

Another use of adversarial examples is adversarial training. By developing scripts to create adversarial examples, I've effectively developed scripts to augment a set of labeled data, producing additional, slightly different, labeled data. I hypothesize that since the adversarial examples of our test data are able to trick our NER models, the adversarial examples must have implicit knowledge that the NER model currently lacks, knowledge which could improve the NER model if it was embedded into the training data. With this in mind, I use the same scripts for creating adversarial examples to augment the training data, creating more, slightly different, training data without additional human annotation cost. This approach – augmenting the training data using techniques for generating adversarial test examples – is called *adversarial training*. My experiments show that adversarial training improves the NER models' robustness to adversarial attacks while retaining its effectiveness on the original test data sets.

Related Works

Most of the research on adversarial examples for neural networks has been performed within the image domain. Within NLP, there has been little research on adversarial examples within NER specifically. However, recent research on adversarial examples within other subcategories of NLP inspires this thesis.

Alzantot et al., 2018 use genetic algorithms to create adversarial examples via word substitutions for sentiment analysis and textual entailment tasks. Their attacks cause standard models to make errors with a rate of 97% and 70% respectively. Further, over 90% of their adversarial examples were correctly classified by humans, evidencing the validity of their attacks. They found that adversarial training did not result in models robust to their adversarial attacks.

Jia and Liang, 2017 create adversarial examples for the Stanford Question Answering Dataset by inserting random sentences into the paragraphs that the model is asked a question about. They find that the model's accuracy drops from an F1 score of 75% to 36%. They find that adversarial training increases the model's accuracy on the adversarial test set to 70%. However, the adversarial training does not offer robustness beyond their specific attack; when they train the model with extraneous sentences added to the end of the paragraph, it performs poorly on data with a sentence added to the beginning, reaching only a 39% F1 score.

By using back translation to create paraphrased training data, Iyyer et al., 2018 develop an encoder-decoder model to produce a paraphrase of an input sentence conforming to a specific inputted syntactic pattern. This paraphrasing model is then used to produce adversarial examples for sentiment analysis. Because their paraphrasing model makes dramatic structural changes to the input sentence rather than minor lexical substitutions, Iyyer et al., 2018 find that their model creates more convincing adversarial examples than uncontrolled paraphrasing models. Their paraphrasing model creates adversarial examples which cause

the sentiment analysis model to fail on 33%/41% (depending on the specific task) of paraphrases of originally successful test examples. In comparison, the adversarial data generated by uncontrolled paraphrasing models only breaks 20%/20% of the sentiment analysis test examples. Adversarial training reduces the percentage of broken test examples from 33%/41% to 20%/31%.

Another example is the research done by Jia, Raghunathan, et al., 2019 on adversarial word substitutions for sentiment analysis tasks. They focus on training a model to be robust to word substitution attacks on test data. They note that their main challenge is knowing what the best (i.e. hardest) adversarial attacks are despite an exponentially sized set of possible perturbations. They use an algorithm called Interval Bound Propagation (IBP) to tractably compute an upper bound for the model's error considering the worst-case adversarial attack, and they set the training goal to be minimizing the upper bound on the worst-case error. They find that this mathematical approach is even more effective than adversarial training via direct data augmentation. Direct data augmentation alone raised the models' accuracy on adversarial test data from 6.9-9.6% using models trained on standard data to 33.0-35.2% using models trained on augmented data. Although this is a major achievement, supporting the effectiveness of data augmentation, their mathematical approach using IBP raises the models' accuracy to 64.7-75.0% (Jia, Raghunathan, et al., 2019).

The research of Ribeiro, Singh, and Guestrin, 2018 comes closest to my work on adversarial examples and adversarial training, although they work in the fields of machine comprehension, visual question-answering, and sentiment analysis rather than NER. Because of the fields they work in, their adversarial perturbations must retain both sentence validity and semantic similarity. Thus, their adversarial examples are *SEAs* (semantically equivalent adversaries) which they generalize into the rules that produce them, *SEARS* (semantically equivalent adversarial rules). Examples of SEARs include changing *"What is..."* to *"What's..."* or chaging *"What* NOUN*..."* to *"Which* NOUN*..."*. The SEARs are shown

to create far more numerous and more effective SEAs than humans writing SEAs directly. They then use the SEARs for adversarial training. This resulted in an insignificant change in performance on the original test data and a significantly lower error rate on the SEAs, from 12.6% to 1.4% for visual question answering and 12.6% to 3.4% on sentiment analysis (Ribeiro, Singh, and Guestrin, 2018). This project in particular motivates my research in adversarial training.

Adversarial examples for NER is a relatively untouched field. However, there are a few recent pieces in that direction. Agarwal et al., 2020a released a manuscript in April 2020 regarding entity-switched datasets for NER. It has not yet been peer-reviewed. The idea is that NER models should be able to recognize named entities from a variety of national origins equally well. This follows recent research on fairness in machine learning[1]. To investigate NER model robustness across national origins, they switch out entities in the CoNLL03 dataset, replacing them with entities of a different national origin. They find that state-of-the-art models have significantly unequal performance across national origins, ranging from a recall of 99.6-99.7% on US-based named entities to only 78.2-84.2% on Vietnamese-based named entities.

Agarwal et al., 2020b is another example of recent research in adversarial data for NER. This manuscript was released in April 2020 and has not yet been peer-reviewed. They attempt to address the question of whether state-of-the-art NER models are learning the context of named entities or are merely memorizing the named entities in the training data. In order to investigate this question, they test the ability of state-of-the-art models to label words given only the context of the word. The models are asked to label sentences of the form "*Tom traveled to _ _ _ last year.*" A sentence is created for each word in the CoNLL03 dataset, with that word blanked out. This context-only approach is similar to the data

[1]For example, language identification models are poor at recognizing African-American Vernacular English as English despite their high performance recognizing text as English when associated with white speakers (Blodgett, Green, and O'Connor, 2016)

augmentation technique of entity masking that I explore, where I replace each entity with a random string of letters. Agarwal et al., 2020b find that models have an F1 score of above 90% given words and context, 50-60% given context only, and 65-80% given the single word only. This suggests that modern models rely more heavily on memorizing named entities than learning the context for named entities. Interestingly, human tests demonstrate that many of the mistakes that the models make during context-only labeling are also made by humans given the context-only sentence. For example, in the sentence *"Their other marksmen were Brazilian defender Vampeta _ _ _ Belgian striker Luc Nilis"* the actual blanked word is *"and"*, so it should be labeled with the tag O. However, the models and human annotators alike labeled the word PER. This is relevant to my research because it suggests the possibility that some of my augmented sentences could be too difficult for human annotators. If this were the case, it would be unreasonable to expect models to correctly label these sentences.

Finally, Araujo et al., 2020 is yet another manuscript on adversarial data for NER released in April 2020. It also has not yet been peer-reviewed. They focus on the biomedical corpora. They have two main attack strategies. First, they mimic typos by either swapping neighboring characters in a word or replacing a character with a character that is nearby on a keyboard. This is different from my approach in that it yields sentences that are grammatically incorrect. However, because they are small perturbations, humans could still annotate the adversarial sentences correctly. Their second approach is to replace chemical and disease names with synonyms. This is different from my approach because they are replacing the actual named entities themselves (chemical and disease names) with synonyms whereas I replace non-entities with synonyms. They find a significant drop in state-of-the-art NER models' performance on these adversarial examples, with the F1 score dropping 20%-50%. Adversarial training helps the models' performance on the adversarial examples. However, it also negatively impacts the models' performance on the original test data.

Chapter Two

Entity Triggers

This section is adapted from the research paper I co-authored titled "Entity Triggers: Learning with Explanations for Named Entity Recognition" which will appear in the 2020 Association for Computational Linguistics conference (Lee et al., 2020).

2.1 Motivation

Recent advances in NER have primarily focused on training neural network models with an abundance of human annotations, yielding state-of-the-art results (Lample et al., 2016). However, NER often requires huge amounts of labeled data (e.g. tens of thousands of labeled sentences). Collecting these human annotations is expensive and time-consuming, especially in technical domains such as biomedical publications, financial documents, legal reports, etc. As we seek to advance NER into more domains with less human effort, how to learn neural models for NER in a annotation-efficient way becomes a crucial research problem.

The standard protocol for obtaining an annotated NER dataset involves an annotator viewing a sentence, selecting token spans as entities, and labeling the spans with their entity types. Because this annotation process provides limited supervision per example, in order to train a high-performance model, it is necessary to collect a large amount of annotations. Given the effort the annotator has already spent analysing the sentence and recognizing the

entity, we aim to investigate how we can obtain extra supervision from this entity annotation.

A human's recognition of an unlabeled entity usually depends on cue words or phrases in the sentence. For instance, we cold infer that the randomized word Kasdfrcxzv is likely to be a location entity mention in the sentence "Tom *traveled* a lot last year *in* Kasdfrcxzv." We recognize this location entity because of the cue phrase "*travel ... in*," which indicates that there should be a location entity after the word "*in*." We hypothesize that these cue phrases not only explain the recognition process, but can also help our NER models to learn and generalize faster. We call such entity-associated rational phrases *entity triggers*. Specifically, we define an entity trigger (or trigger for simplicity) as a group of words that can help explain the recognition process of a particular entity in the same sentence. For example, in Figure 2.1, *"had ... lunch at"*[1] and *"where the food"* are two distinct triggers associated with the RESTAURANT entity "Rumble Fish." An entity trigger should be a necessary and sufficient cue for humans to recognize its associated entity even if we mask the entity with a random word. Thus, unnecessary words such as "fantastic" should not be considered part of the entity trigger.

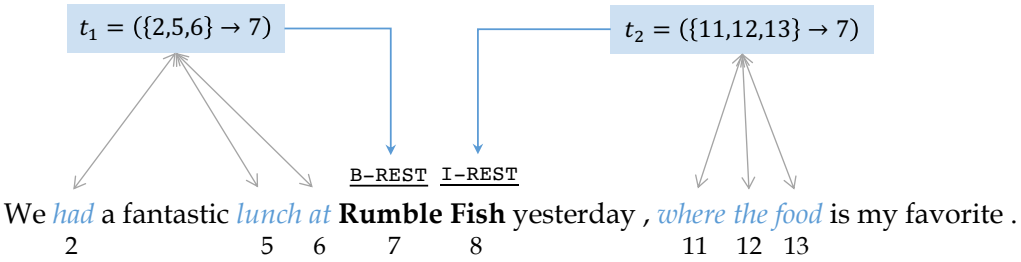

Figure 2.1 Example of entity triggers: *"had(2) ... lunch(5) at(6)"* and *"where(11) the(12) food(13)"* are two individual entity triggers associated to the entity mention "Rumble Fish" typed as RESTaurant, which starts at the 7th token.

We argue the benefits of supervising models using a combination of entity triggers and standard entity annotations. This approach is more powerful because unlabeled sentences, such as "Bill *enjoyed* a great *dinner* with Alice *at* Zcxlbz." can be matched with the existing

[1]Note that a trigger can be a discontinuous phrase.

trigger "*had ... lunch at*" via their semantic relatedness. This makes it easier for NER models to recognize Zcxlbz as a RESTAURANT entity than if we had only trained the NER model using entity labels. In contrast, if we only had the entity annotation itself (i.e. "Rumble Fish") as supervision, the model would require many similar examples in order to learn this simple pattern. Annotating triggers in addition to entities is not significantly more laborious since annotators have already read the sentences and analysed the entities. Thus, we hypothesize that using triggers as additional supervision is a more cost-effective way to train models.

We crowd-sourced 14,708 triggers on two well-studied NER datasets to study the usefulness of entity triggers. We proposed a novel framework named *Trigger Matching Network* (TMN). Because it exploits triggers, TMN is more powerful and cost-effective than standard approaches. The TMN framework consists of three components: 1) a trigger encoder, 2) a semantic trigger matching module, and 3) an entity tagger. Our first learning stage jointly trains the trigger encoder and semantic trigger matching module. We then learn a sequence tagger using trigger-enhanced attentions in order to incorporate existing trigger information into entity predictions on unseen sentences.

Our contributions are as follows:

- We introduced the concept of "entity triggers," a novel form of explanatory annotation for named entity recognition problems. We crowd-sourced and publicly released 14k annotated entity triggers on two popular datasets: *CoNLL03* (generic domain), *BC5DR* (biomedical domain).

- We proposed a novel learning framework, named *Trigger Matching Network*, which encodes entity triggers and softly grounds them on unlabeled sentences to increase the effectiveness of the base entity tagger (Section 2.3).

- Experimental results (Section 2.4) show that the proposed trigger-based framework is significantly more cost-effective. The TMN only uses 20% of the trigger-annotated

sentences from the original CoNLL03 dataset and achieves comparable performance to the conventional model using 70% of the annotated sentences.

2.2 Problem Formulation

We consider the problem of how to cost-effectively learn a model for NER using entity triggers. This section introduces basic notations and provides a formal task definition for learning using entity triggers.

As described in Sec 1.1.1, in the conventional setup for supervised learning for NER, we have a labeled corpus $\mathcal{D}_L = \{(\mathbf{x_i}, \mathbf{y_i})\}$, and an unlabeled corpus $\mathcal{D}_U = \{\mathbf{x_i}\}$ that we want to label.

We use $T(\mathbf{x}, \mathbf{y})$ to represent the set of annotated entity triggers, where each trigger $t_i \in T(\mathbf{x}, \mathbf{y})$ is associated with an entity index e and a set of word indices $\{w_i\}$. Note that we use the index of the first word of an entity as its entity index. That is, $t = (\{w_1, w_2, \cdots\} \rightarrow e)$, where e and w_i are integers in the range of $[1, |\mathbf{x}|]$. For instance, in the example shown in Figure 2.1, the trigger *"had ... lunch at"* can be represented as a trigger $t_1 = (\{2, 5, 6\} \rightarrow 7)$, because this trigger specifies the entity starting at index 7, *"Rumble"*, and it contains a set of words with indices: *"had"* (2), *"lunch"* (5), and *"at"* (6). Similarly, we can represent the second trigger *"where the food"* as $t_2 = (\{11, 12, 13\} \rightarrow 7)$. Thus, we have $T(\mathbf{x}, \mathbf{y}) = \{t_1, t_2\}$ for this sentence.

Adding triggers creates a new form of data $\mathcal{D}_T = \{(\mathbf{x_i}, \mathbf{y_i}, T(\mathbf{x_i}, \mathbf{y_i})\}$. Our goal is to learn a model for NER from a trigger-labeled dataset $\mathcal{D}_T$, such that we can achieve comparable learning performance to a model using a much larger $\mathcal{D}_L$.

2.3 Proposed Framework

This section presents our framework for using triggers. Our intuition is to treat triggers as soft templates such that new sentences can be matched with these triggers via their deep semantic relatedness in inference time. Recall the trigger matching example discussed in Section 2.1. By using soft matching via semantic relatedness rather than hard matching via character similarity, we are able to match sentences such as "Bill *enjoyed* a great *dinner* with Alice *at* Zcxlbz." to the semantically similar trigger "*had ... lunch at*." In order to soft match entity triggers, we need a trainable way to learn trigger representations (i.e. *trigger vectors*). Given trigger vectors, we can train a model to label each token in the unlabeled sentences with a NER tag using the trigger vectors as extra supervision. We propose a straightforward yet effective framework, named *Trigger Matching Network* (TMN), consisting of a trigger encoder (`TrigEncoder`), a semantic-based trigger matching module (`TrigMatcher`), and a base sequence tagger (`SeqTagger`). We have two learning stages for the proposed framework: the first stage (Section 2.3.1) jointly learns the `TrigEncoder` and `TrigMatcher`, and the second stage (Section 2.3.2) uses the trigger vectors to learn NER tag labels. Figure 2.2 shows this pipeline. We introduce the inference in Section 2.3.3.

2.3.1 Trigger Encoding and Semantic Trigger Matching

Learning trigger representations and semantically matching them with sentences are inseparable tasks. Desired trigger vectors capture the semantics in a shared embedding space with token hidden states such that sentences and triggers can be semantically matched. Learning an attention-based matching module between entity triggers and sentences is necessary so that triggers and sentences can be semantically matched. Therefore, as the first learning stage, we propose to jointly train the trigger encoder (`TrigEncoder`) and the attention-based trigger matching module (`TrigMatcher`) using a shared embedding space.

Specifically, for a sentence $\mathbf{x}$ with multiple entities $\{e_1, e_2, \cdots\}$, for each entity e_i we

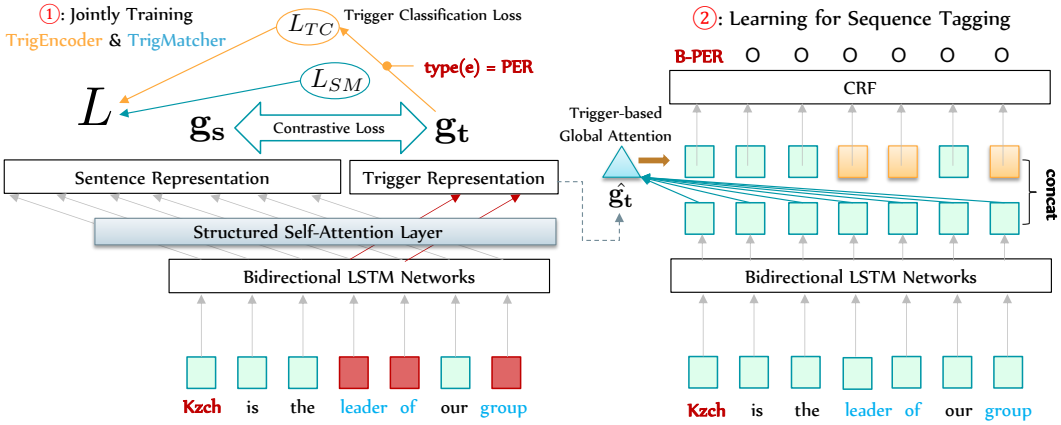

Figure 2.2 Two-stage training of the *Trigger Matching Network*. We first jointly train the `TrigEncoder` (via trigger classification) and the `TrigMatcher` (via contrastive loss). Then, we reuse the training data trigger vectors as attention queries in the `SeqTagger`.

assume that there is a set of triggers $T_i = \{t_1^{(i)}, t_2^{(i)}, \cdots\}$. To enable more efficient batch-based training, we reform the trigger-based annotated dataset $\mathcal{D}_T$ such that each new sequence contains only one entity and one trigger. We then create a training instance by pairing each entity with one of its triggers, denoted $(\mathbf{x}, e_i, t_j^{(i)})$. The learning objective in this stage is to output the tag sequence $\mathbf{y}$.

For each reformed training instance $(\mathbf{x}, e, t)$, we first apply a bidirectional LSTM (BLSTM) on the sequence of word vectors of $\mathbf{x}$, obtaining a sequence of hidden states that are the contextualized word representations $\mathbf{h}_i$ for each token x_i in the sentence. We use $\mathbf{H}$ to denote the matrix containing the hidden vectors of all of the tokens, and we use $\mathbf{Z}$ to denote the matrix containing the hidden vectors of all of the trigger tokens inside the trigger t.

In order to learn an attention-based representation of both triggers and sentences, we follow the self-attention method introduced by Z. Lin et al., 2017 and described in Sec 1.1.2

as follows:

$$\vec{a}_{sent} = \text{SoftMax}\left(W_2 \tanh\left(W_1 \mathbf{H}^T\right)\right)$$
$$\mathbf{g_s} = \vec{a}_{sent}\mathbf{H}$$
$$\vec{a}_{trig} = \text{SoftMax}\left(W_2 \tanh\left(W_1 \mathbf{Z}^T\right)\right)$$
$$\mathbf{g_t} = \vec{a}_{trig}\mathbf{Z}$$

Here, W_1 and W_2 are two trainable parameters for computing self-attention score vectors $\vec{a}_{sent}$ and $\vec{a}_{trig}$. W_1 and W_2 are essentially the level of importance the self-attention layer gives to each token. We obtain a vector representing the weighted sum of the token vectors in the entire sentence as the final sentence vector $\mathbf{g_s}$. Similarly, $\mathbf{g_t}$ is the final trigger vector, representing the weighted sum of the token vectors in the trigger.

We want to use the type of the associated entity as supervision to guide the trigger representation. Thus, the trigger vector $\mathbf{g_t}$ is further fed into a multi-class classifier to predict the *type* of the associated entity e (such as `PER`, `LOC`, etc) which we use $\texttt{type}(e)$ to denote. Because our model is a multi-class classifier (selecting which label should be assigned to a given word), we use a cross-entropy loss function. The loss is low if the model predicts a high probability for the correct label. The trigger classification loss is as follows, where θ_{TC} is the learning parameter:

$$L_{TC} = -\sum \log \text{P}\left(\texttt{type}(e) \mid \mathbf{g_t}; \theta_{TC}\right)$$

In order to learn to match triggers and sentences based on their self-attention based representations, we use contrastive loss (Hadsell, Chopra, and LeCun, 2006). Contrastive loss is a loss method for unlabeled data depending on both positive examples (a pair of vectors of the same class) and negative examples (a pair of vectors of different classes). The contrastive loss is low if the positive examples have a small distance between their two vectors and the negative examples have a large distance between their two vectors. This ensures that vectors of the same class are encoded near each other in the vector space.

The intuition behind our use of contrastive loss is that a trigger and the sentence that the trigger belongs to should have similar representations (i.e. have a small distance betweeen them, d). `TrigMatcher` needs to be trained with both positive and negative examples of the form {sentence, trigger, label}, so we randomly mix the triggers and sentences so that we have negative examples (i.e. mismatches). For the negative examples, we expect a margin m between their representations. The contrastive loss of the soft matching is defined as follows, where $\mathbb{1}_{\text{matched}}$ is 1 if the trigger belongs to this sentence and 0 if they are mismatched:

$$d = \|\mathbf{g_s} - \mathbf{g_t}\|_2$$
$$L_{SM} = (1 - \mathbb{1}_{\text{matched}})\frac{1}{2}(d)^2 + \mathbb{1}_{\text{matched}}\frac{1}{2}\{\max(0, m - d)\}^2$$

Thus, the joint loss we aim to optimize is $L = L_{TC} + \lambda L_{SM}$.

2.3.2 Trigger-Enhanced Sequence Tagging for NER

Following the most common design of neural NER architecture, BLSTM-CRF (Ma and Hovy, 2016), we incorporate the entity triggers as attention queries (explained in Sec 1.1.2) to train a trigger-enhanced sequence tagger for NER. Note that the BLSTM used in the the `TrigEncoder` and `TrigMatcher` modules is the same BLSTM we use in the `SeqTagger` to obtain $\mathbf{H}$, the matrix containing the hidden vectors of all of the tokens. Given a sentence $\mathbf{x}$, we use the previously trained `TrigMatcher` to compute the mean of all the trigger vectors $\hat{\mathbf{g_t}}$ associated with this sentence. Following the conventional attention method (Luong, Pham, and Christopher D Manning, 2015), we incorporate the mean trigger vector as the query, creating a sequence of attention-based token representations, $\mathbf{H}'$.

$$\vec{\alpha} = \text{SoftMax}\left(\boldsymbol{v}^\top \tanh\left(U_1\mathbf{H}^T + U_2\hat{\mathbf{g_t}}^T\right)^\top\right)$$
$$\mathbf{H}' = \vec{\alpha}\ \mathbf{H}$$

where U_1, U_2, and v are trainable parameters for computing the trigger-enhanced attention scores for each token. Finally, we concatenate the original token representation $\mathbf{H}$ with the trigger-enhanced one $\mathbf{H}'$ as the input ($[\mathbf{H}; \mathbf{H}']$) to the final CRF tagger. Note that in this stage, our learning objective is the same as conventional NER, which is to correctly predict the tag for each token.

2.3.3 Inference on Unlabeled Sentences

When inferencing tags on unlabeled sentences, we do not know the sentence's triggers. Instead, we use the `TrigMatcher` to compute the similarities between the self-attended sentence representations and the trigger representations, using the most suitable triggers as additional inputs to the `SeqTagger`. Specifically, we have a trigger dictionary from our training data, $\mathcal{T} = \{t|(\cdot, \cdot, t) \in \mathcal{D}_T\}$. Recall that we have learned a trigger vector for each of them, and we can load these trigger vectors as a look-up table in memory. For each unlabeled sentence $\mathbf{x}$, we first compute its self-attended vector $\mathbf{g_s}$ as we do when training the `TrigMatcher`. Using L2-norm distances to compute the contrastive loss, we efficiently retrieve the most similar triggers in the shared embedding space of the sentence and trigger vectors. Then, we calculate $\hat{\mathbf{g_t}}$, the mean of the top k nearest semantically matched triggers, and use it as the attention query for `SeqTagger`, as we did in Section 2.3.2. Now, we can produce trigger-enhanced sequence predictions on unlabeled sentences, as shown in Figure 2.3.

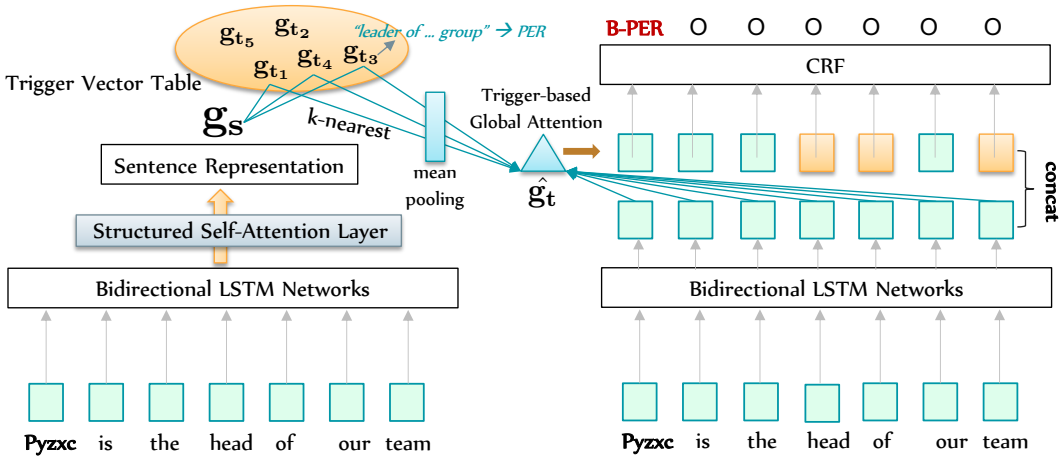

Figure 2.3 The *inference* process of the TMN framework. It uses the `TrigMatcher` to retrieve the k nearest triggers and average their trigger vectors as the attention query for the previously trained `SeqTagger`. This is how a previously unseen cue phrase (e.g., *"head of ... team"*) can be matched with a previously seen trigger (e.g., *"leader of ... group"*).

2.4 Experiments

This section discusses how we collected entity triggers and empirically studies the data-efficiency of our framework.

TRIGGER ! TRIGGER 2
Germany's representative **<PERSON>** said on Wednesday consumers should buy sheepmeat.

TRIGGER ! TRIGGER 2
We upgraded the **<ASPECT_TERM>** to four gigabytes.

TRIGGER !
Intake of **<CHEMICAL>** may accelerate the progression of the disease.

Figure 2.4 Example of trigger annotations.

2.4.1 Annotating Triggers as Explanatory Supervision

We used a general domain dataset CoNLL2003 (Tjong Kim Sang and De Meulder, 2003) and a bio-medical domain dataset BC5CDR (J. Li et al., 2016). Both datasets are well-studied and popular in evaluating the performance of neural named entity recognition models such as BLSTM-CRF (Ma and Hovy, 2016). In order to collect the entity triggers from human

annotators, we used *Amazon SageMaker*[2] to crowd-source entity triggers. Specifically, we sampled 20% of each training set and reformed the data to reflect the entity + entity trigger format that we discussed in Section 2.2. Annotators were asked to annotate a group of words that would be helpful in typing and/or detecting the occurrence of a particular entity in the sentence. We masked the entity tokens with their types so that human annotators were more focused on the non-entity words in the sentence when considering the triggers. For annotators' convenience, we separated a sentence with multiple entities into single-entity sentences and masked the entity word into its corresponding entity type. Figure 2.4 demonstrates the annotator's interface. In the end, we consolidate multiple triggers for each entity by taking the intersection of the three annotators' results. Statistics of the final curated triggers are summarized in Table 2.1. We released the 14k entity triggers to the community for future research in trigger-enhanced NER learning.

Dataset	Entity Type	# Entities	# Triggers	Avg. # Triggers per Entity	Avg. Trigger length
CONLL 2003	PER	1,608	3,445	2.14	1.41
	ORG	958	1,970	2.05	1.46
	MISC	787	2,057	2.61	1.4
	LOC	1,781	3,456	1.94	1.44
	Total	5,134	10,938	2.13	1.43
BC5CDR	Disease	906	2,130	2.35	2.00
	Chemical	1,085	1,640	1.51	1.99
	Total	1,991	3,770	1.89	2.00

Table 2.1 Statistics of the crowd-sourced entity triggers.

2.4.2 Base model

We require a base model to compare with our proposed TMN model in order to validate whether the TMN model effectively uses triggers to improve model performance in a limited label setting. We chose the CNN-BLSTM-CRF as our base model, as described in Sec 1.1.2. Our TMNs are implemented within the same codebase and use the same external

[2]An advanced version of *Amazon Mechanical Turk*. https://aws.amazon.com/sagemaker/

word vectors from GloVE (Pennington, Socher, and Christopher D. Manning, 2014). The hyper-parameters of the CNNs, BLSTMs, and CRFs are also the same. This ensures a fair comparison between a typical non-trigger NER model and our trigger-enhanced framework.

2.4.3 Results and analysis

CONLL 2003										
	BLSTM-CRF				TMN			TMN + SELF-TRAINING		
ent.	Precision	Recall	F1	**trig.**	Precision	Recall	F1	Precision	Recall	F1
5%	70.85	67.32	69.04	3%	76.36	74.33	75.33	80.36	75.18	77.68
10%	76.57	77.09	76.83	5%	81.28	79.16	80.2	81.96	81.18	81.57
20%	82.17	80.35	81.3	7%	82.93	81.13	82.02	82.92	81.94	82.43
30%	83.71	82.76	83.23	10%	84.47	82.61	83.53	84.47	82.61	83.53
40%	85.31	83.1	84.18	13%	84.76	83.69	84.22	84.64	84.01	84.33
50%	85.07	83.49	84.27	15%	85.61	84.45	85.03	86.53	84.26	85.38
60%	85.58	84.54	85.24	17%	85.25	85.46	85.36	86.42	84.63	85.52
70%	86.87	85.3	**86.08**	**20%**	86.04	85.98	**86.01**	87.09	85.91	**86.5**
BC5CDR										
	BLSTM-CRF				TMN			TMN + SELF-TRAINING		
ent.	Precision	Recall	F1	**trig.**	Precision	Recall	F1	Precision	Recall	F1
5%	63.37	43.23	51.39	3%	66.47	57.11	61.44	65.23	59.18	62.06
10%	68.83	60.37	64.32	5%	69.17	73.31	66.11	68.02	66.76	67.38
20%	79.09	62.66	69.92	7%	64.81	69.82	67.22	69.87	66.03	67.9
30%	80.13	65.3	71.87	10%	71.89	69.57	70.71	69.75	72.75	71.22
40%	82.05	65.5	72.71	13%	73.36	70.44	71.87	75.11	69.31	72.1
50%	82.56	66.58	**73.71**	15%	70.91	72.89	71.89	71.23	73.31	72.26
60%	81.73	70.74	75.84	17%	75.67	70.6	73.05	77.47	70.47	73.97
70%	81.16	75.29	76.12	**20%**	77.47	70.47	**73.97**	75.23	73.83	**74.52**

Table 2.2 Labor-efficiency study on BLSTM-CRF and TMN. "ent." means the percentage of the sentences (labeled only with entity tags) we used for BLSTM-CRF, while "trig." denotes the percentage of the sentences (labeled with both entity tags and trigger tags) we used for TMN.

We experimented with using different proportions of the training data for both the base model and the TMN. From Table 2.2, we see that the TMN models using 20% of the training sentences with annotated triggers achieve comparable results to the BLST-CRF models using 50% (BC5CDR)/70% (CONLL03) of the training data. Even though the annotators spent slightly more effort per sentence when tagging triggers, the drastic improvement in model performance evidences the merit of our approach. With the self-training method (Rosenberg, Hebert, and Schneiderman, 2005), which iteratively expands the labeled set with the most

confident predictions from the unlabeled set during training, we further improve the TMN model's F1 scores by about 0.5∼1%.

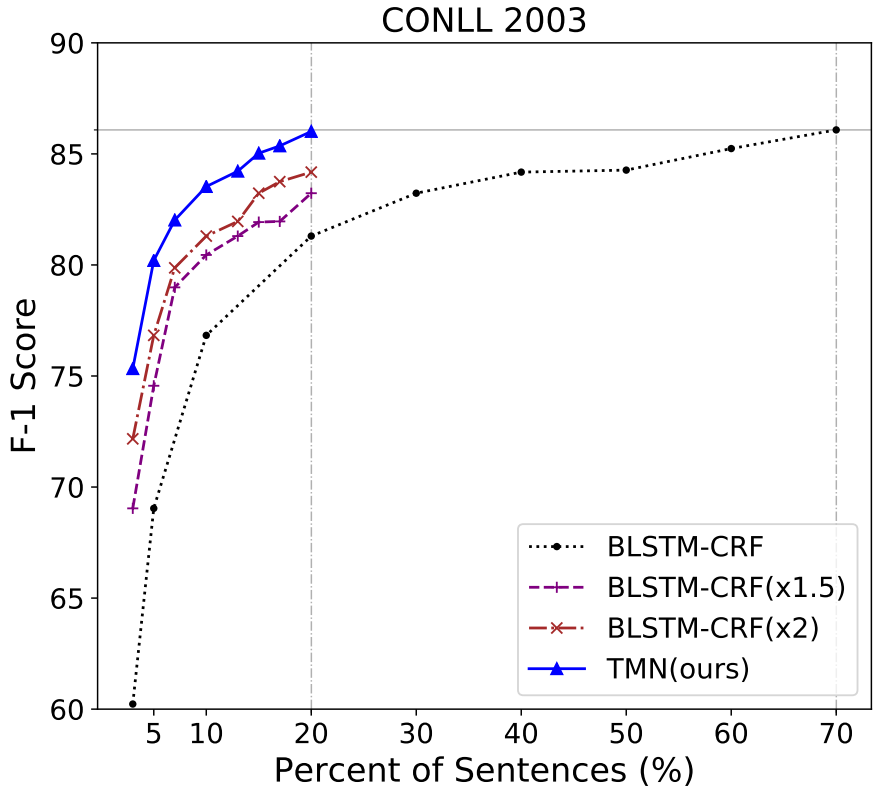

Figure 2.5 The cost-effectiveness study. We stretch the curve of the BLSTM-CRF parallel to the x-axis by 1.5 and 2 respectively. Even if we assume annotating entity triggers cost 150/200% the amount of human effort as annotating entities only, TMN is still much more effective.

Although it is hard to accurately study the time cost on the crowd-sourcing platform we used[3], based on our offline simulation we argue that annotating both triggers and entities are about 1.5 times ("BLSTM-CRF (x1.5)") longer than only annotating entities. In Figure 2.5, the x-axis for BLSTM-CRF reflects the number of sentences annotated with only entities, while for TMN it means the number of sentences tagged with both entities and triggers. In order to reflect human annotators spending 1.5 to 2 times as long annotating triggers

[3]Annotators may suspend jobs and resume them without interaction with the crowd-sourcing platform.

and entities as they spend annotating only entities, we stretch the x-axis for BLSTM-CRF. For example, the line labeled ("BLSTM-CRF (x2)") associates the actual F1 score for the baseline model trained on 40% of the sentences with the x-axis value of 20%. We can clearly see that the proposed TMN outperforms the BLSTM-CRF model by a large margin. Even if we consider the extreme case that tagging triggers requires twice the human effort ("BLSTM-CRF (x2)"), the TMN is still significantly more labor-efficient in terms of F1 scores. Based on these results, we claim that 1) when annotating data from scratch, it is more effective to ask annotators to label both entities and *triggers*, and 2) if entity annotations already exist for some datasets, annotating *triggers* can still boost model performance.

Figure 2.6 shows two examples illustrating that the trigger attention scores help the TMN model recognize entities. The training data has 'per day' as a trigger phrase for chemical-type entities, and this trigger matches the phrase 'once daily' in an unseen sentence during the inference phase of `TrigMatcher`. Similarly, in CoNLL03 the training data trigger phrase "said it" matches with the phrase "was quoted as saying" in an unlabeled sentence. These results not only support our argument that trigger-enhanced models such as TMN can effectively learn, but they also demonstrate that trigger-enhanced models can provide reasonable interpretation, meaning that they can be examined to understand "why" the model chose its entity predictions. Interpretability lacks in other neural NER models.

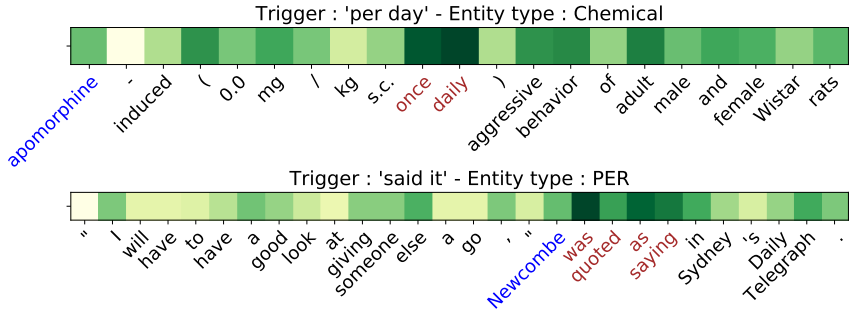

Figure 2.6 Two case studies of the trigger attention scores during inference. The darker cells have higher attention weights.

2.5 Future Directions

Future research should investigate the use of our framework for trigger-enhanced NER learning in conjunction with related approaches to low-resource learning for NER. For example, one could combine dictionary-based distant supervision to our approach by first creating a dictionary using a high-quality corpus. One could also apply active learning by asking human annotators to annotate the triggers chosen by an active sampling algorithm designed for TMN.

Another future direction is to develop highly interpretable NER models using a trigger-enhanced. As was shown in 2.6, the trigger attention scores during the inference stage can provide reasonable interpretation which could guide future research directions for fine-tuning models.

It would also be interesting to investigate the possibility of automating entity trigger annotation by developing models to extract novel triggers or by transferring existing entity triggers from one language to low-resource languages.

Finally, I would like to combine the topics of entity triggers and adversarial examples. Since entity triggers can come in a variety of similar forms (e.g. *had lunch at* versus *ate dinner at*), I am interested in perturbing entity triggers from the training data to make the model more robust to variations on entity triggers in the test data.

Chapter Three

Generating Adversarial Data

3.1 Background

3.1.1 Definitions

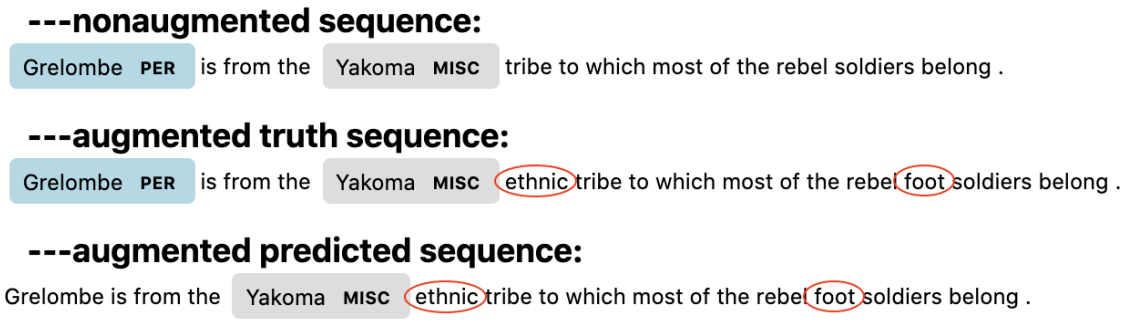

Figure 3.1 Motivating example of a context-based adversarial attack. The first sentence is from the original CoNLL03 dataset. The second sentence is the correct labeling sequence. The third sentence is what the model predicted.

As described in Sec 1.2.2, an adversarial example for a neural network is an example that is created by perturbing a correctly classified example, causing the model to misclassify the adversarial example. I explore several ways to perturb sentences such that the original entity labels and the sentence structure remain valid. These adversarial attacks include entity-based attacks – masking entities and switching entities – and context-based attacks –

inserting and removing adjectives as well as replacing non-entity words with synonyms.

A motivating example of a context-based attack is seen in Fig 3.1. In this adversarial attack, two adjectives are inserted into the sentence. Although the adversarial sentence has the exact same meaning as the original sentence, the model misses a PER entity label in the adversarial sentence despite correctly labeling the original sentence.

Fig 3.2 is a motivating example for an entity-based attack. In this attack, two LOC entities were switched between sentences. Despite predicting correctly on the two original sentences, the model predicted incorrectly on the adversarial sentence, classifying "VANCOUVER" as an ORG instead of a LOC.

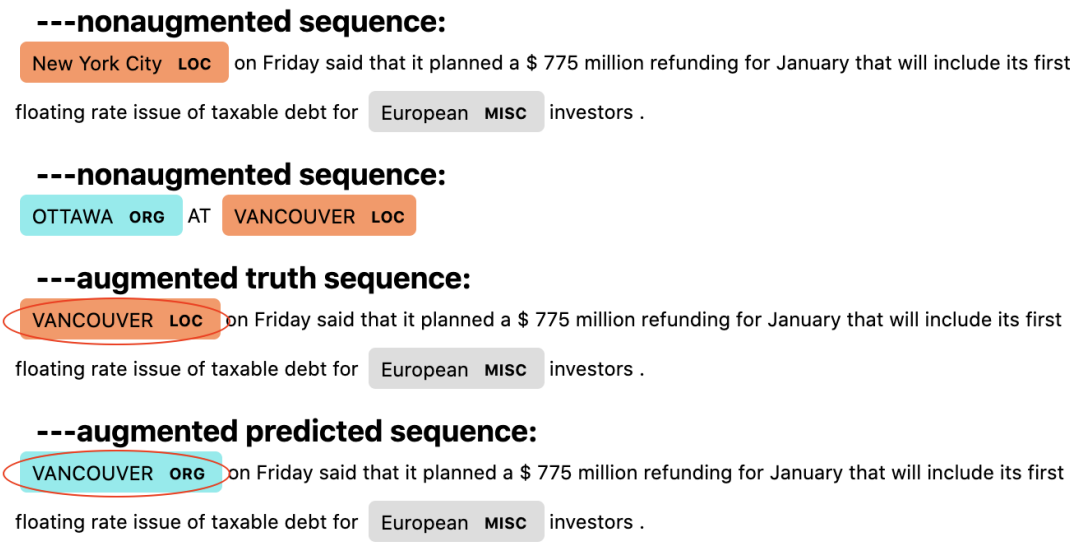

Figure 3.2 Motivating example of an entity-based adversarial attack. The first two sentences are from the original CoNLL03 dataset. The third sentence is the correct labeling sequence. The fourth sentence is what the model predicted.

As described in Sec 1.1.2, I use a BLSTM-CRF neural NER model throughout these experiments. I use CoNLL03 as the dataset for my experiments due to its pervasiveness throughout NER literature and its simplistic structure, having only four entity categories.

3.1.2 Motivation

An important question when evaluating a NER model is how well it will work in a real-world scenario. Because training, test, and dev data are usually collected in the same manner, the test data may be over-familiar to the NER model, causing the model to perform better on the test data than it might have on a completely different corpus during a real-world application.

One cause of this discrepancy is that models rely heavily on memorizing named entities, meaning that they have difficulty generalizing to unseen entities. This memorization is evidenced in Table 3.1 and Fig 3.3, which show that our baseline model performs significantly better given sentences with previously seen entities rather than sentences with only new entities. To test this, I split the CoNLL03 test data into two subsets, one having all of the sentences containing one or more entity from the training data, and the other having all of the sentences with only unseen entities.

	Performance on dev data			**Performance on test data**		
Subset of unlabeled data	*Precision*	*Recall*	*F1*	*Precision*	*Recall*	*F1*
All sentences	94.86	94.5	94.68	91.03	90.91	90.97
One or more seen ents	96.93	96.84	96.88	94.24	94.13	94.18
Only unseen ents	90.17	88.82	89.49	87.66	85.86	86.75

Table 3.1 Comparison of the our baseline model's performance on the standard CoNLL03 test data, the subset of the test data with at least one entity from the training data, and the subset of the test data with only unseen entities.

In order to further test a model's dependence on known entities, I develop adversarial sentences in which the entities are masked with random letters, and I develop adversarial sentences in which the entities are swapped between sentences.

Another reason that a NER model might perform worse on a new dataset is that the writing style is different from the training set. Because NER models are generally trained and tested on data from the same corpus, their results can be overconfident in their ability to label new sentences. In order to test whether the model is overfitted to the writing style of the dataset, I develop adversarial sentences in which non-entities are manipulated. In

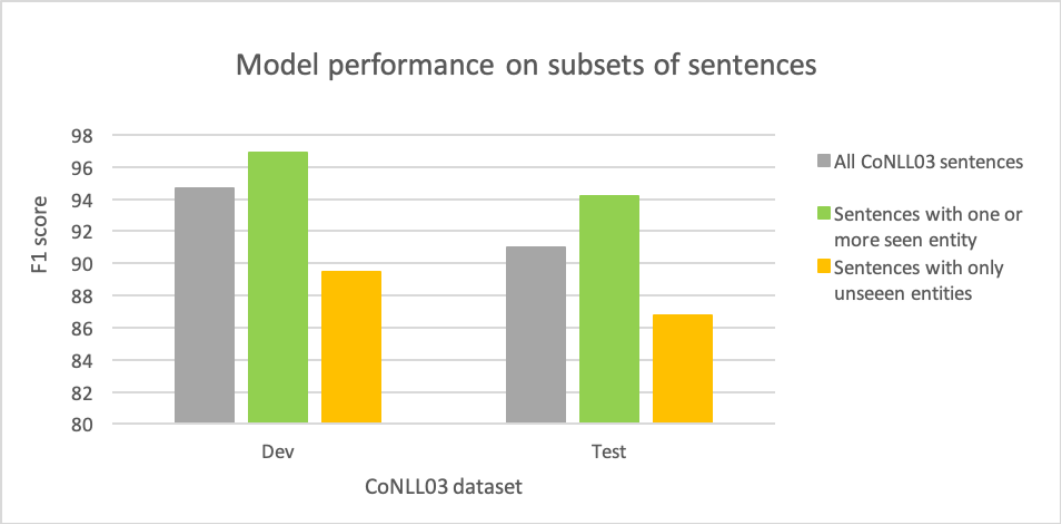

Figure 3.3 Comparison of the our baseline model's performance on the standard CoNLL03 test data, the subset of the test data with at least one entity from the training data, and the subset of the test data with only unseen entities.

particular, I remove adjectives, insert adjectives, and replace words with synonyms.

The main motivation for this work is developing schemes to create adversarial test data so that state-of-the-art models can be more accurately evaluated. An accurate evaluation will allow researchers to better understand in what circumstances it is safe to deploy state-of-the-art NER models. Additionally, these adversarial examples can shed light on the models' shortcomings, pointing future research in a productive direction. A secondary motivation for this work is using adversarial data to benefit the models. I investigate the effectiveness of adversarial training – training the models on adversarial training examples – to develop more robust models.

3.2 Adversarial Attack Experiments

In this section, I describe the different types of adversarial attacks I experiment with and examine their effectiveness. For each attack type, I develop a script to augment an input dataset in an adversarial manner. For example, the entity masking script takes a standard dataset as input and returns the same dataset with each entity replaced by a random string of

characters. I run these augmentation scripts on the CoNLL03 datasets to produce augmented dev and test datasets. I then train a NER model on the original CoNLL03 training and dev set and test its ability to correctly label the augmented dev and test datasets. The results on the augmented test datasets are particularly meaningful because the test set is not used anywhere in the training of the model, so its sentences are completely new to the NER model. I find that our baseline NER model performs worse on these adversarial datasets than on the original CoNLL03 dev and test set, evidencing the success of my adversarial attacks in demonstrating a lack of robustness in state-of-the-art NER models.

3.2.1 Entity-based attacks

Full Entity Masking

The most straightforward approach I took was fully masking every entity in a given sentence. To form a mask, I replaced each letter in the entity with a random letter with the same case (to retain the same capitalization). Any non-alphabetic characters (e.g. hyphens or digits) were left the same.

Masking entities drastically affected the model's performance, causing the model to achieve an F1 score of 61.66% compared to 90.97% on the original test data. In fact, this attack may perform too well, changing the sentence to an extent that would make labeling it too difficult even for humans. For example, one of the sentences in the entity-swapped data reads *"Kelyta feedlot cattle roundup – RZIJ."*. This corresponds to the original sentence *"Kansas feedlot cattle roundup – USDA."* In this example, even as a human we need the entity *USDA* itself to understand the final word as an organization.

Single-Entity Masking

In order to address the problem of over-masking, I propose several strategies. First, I only mask one entity in a sentence. Often, when trying to label a given entity in a sentence,

other nearby entities help to provide context. The single-entity masking approach allows for the context of other entities. As shown in Table 3.2 and Fig 3.4, masking a single entity per sentence still successfully reduces the model's performance, while reducing it to a more realistic extent, from a 90.97% F1 score to 84.63%.

Within the two categories of single-entity masking and full entity masking, I test three different masking approaches. The first method is fully masking every letter as described above.

Half Masking

My second approach is half masking, in which I only mask every other letter in an entity. As with full masking, I retain casing and special characters. For example, the original sentence *"SOCCER - JAPAN GETS LUCKY WIN, CHINA IN SUPRISE DEFEAT."* is masked to *"SOCCER - JPPZN GETS LUCKY WIN, CKIKA IN SUPRISE DEFEAT."*. As a human, I find this half masked sentence is easier to label than its fully masked counterpart *"SOCCER - LNQUY GETS LUCKY WIN, INNXM IN SUPRISE DEFEAT."* As shown in Table 3.2 and Fig 3.4, partial masking allows the model to perform better than full masking, although the difference is small.

Masking with Retained Vowels

My final approach to masking is to mask the entire word, but only replace vowels with vowels and consonants with consonants. I came up with this approach in the hopes that the vowel/consonant pattern would retain enough morphological structure to help the model decipher the proper labels. However, I was surprised to find that there was almost no difference between the model's performance on fully masked sentences and sentences masked with the vowels retained (shown in Table 3.2 and Fig 3.4).

Entity Switching

Along with masking entities, I created adversarial sentences by switching entities between sentences. Specifically, for every pair of entities of the same type in the corpus, I created two new sentences with the entities switched. That is, given a sentence $m = (m_{ne}, i)$ with non-entities m_{ne} and entity i and a sentence $n = (n_{ne}, j)$ with non-entities n_{ne} and entity j, I produce two new sentences $m' = (m_{ne}, j)$ and $n' = (n_{ne}, i)$. The goal of this attack is to test if the model is memorizing the context for specific entities rather than the context for entity types. For example, the `PER` entity "Michael Phelps" might often be associated with words about swimming. If we switch "Michael Phelps" and "Vin Diesel," putting Michael Phelps in a sentence about acting, can the model still recognize Michael Phelps as a `PER` entity? As can be seen in Table 3.2 and Fig 3.4, entity switching had no significant effect on model performance.

Results of entity-based adversarial attacks

The results in Table 3.2 and Fig 3.4 demonstrate the effectiveness of entity masking in attacking our NER model, suggesting that the model relies heavily on entity memorization. The fact that entity switching was not an effective strategy suggests that the NER model does not focus on the context of specific entities. In order to validate these attacks, we will need to perform a human study to ensure that the generated sentences can still be labeled by humans. In this regard, it is likely that the entity masking attacks using only one entity are more realistic.

Adversarial Technique	Performance on dev data			Performance on test data		
	Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Original data	94.86	94.50	94.68	91.03	90.91	90.97
Entity Switching	93.99	93.62	93.81	91.37	90.47	90.92
All ents fully masked	62.89	58.60	60.67	63.94	59.54	61.66
All ents half masked	65.26	61.76	63.46	66.16	62.20	64.12
All ents masked, retain vowels	62.82	59.32	61.02	64.68	60.82	62.69
One ent fully masked	87.46	85.65	86.55	85.68	83.60	84.63
One ent half masked	88.01	86.58	87.29	86.12	84.23	85.16
One ent masked, retain vowels	87.54	86.04	86.79	85.62	83.77	84.68

Table 3.2 Comparison of our baseline model's performance on the standard CoNLL03 datasets with its performance on entity-based adversarial attacks.

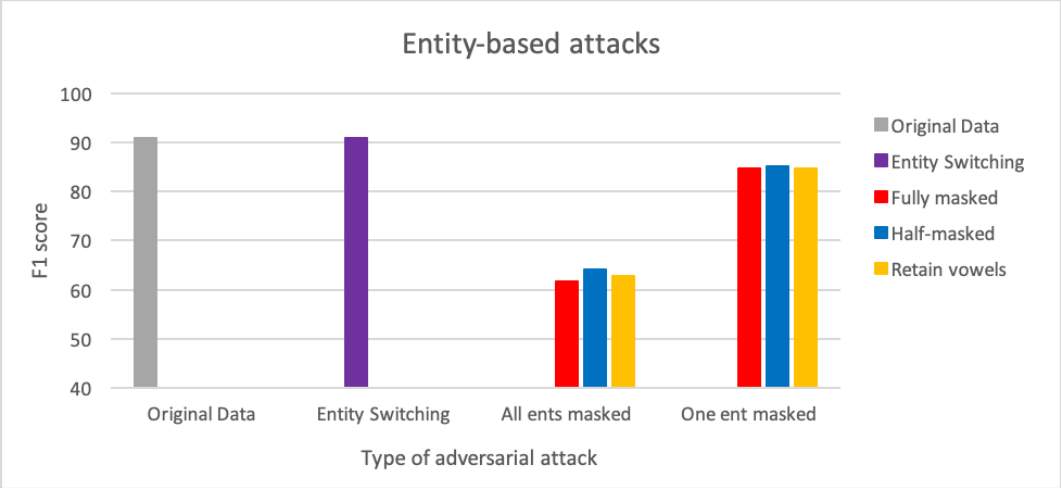

Figure 3.4 Comparison of our baseline model's performance on the standard CoNLL03 test data with its performance on entity-based adversarial attacks.

3.2.2 Context-based attacks

In this section I focus on context-based attacks: adding non-entity adjectives before non-entity nouns, removing non-entity adjectives, and replacing non-entity words with synonyms. When choosing related words to insert or replace original words with, I mimic the existing casing pattern (all lowercase, all uppercase, or capitalized) in order to retain continuity with the rest of the sentence. Further, in order to decrease the chance of inserting a named entity, I don't allow suggested related words that were capitalized before this formatting.

Within these context-based attacks there are several subcategories based on two variables: how to choose which non-entity words to attack and how to select related words (necessary for inserting adjectives or replacing words with synonyms). Intuitively, I would like to attack the most influential words since the goal is to create a sentence that the model can no longer label properly. I investigate three different ways of choosing the most influential words: random selection, model differentiation, and cooccurrence matrices. I also use two different publicly-available platforms for selecting related words: ConceptNet and Bert.

Selecting related words: ConceptNet

The first technique for selecting related words that I explored is ConceptNet, a publicly-available semantic network, created to provide computers an understanding of the meaning of words (Speer, Chin, and Havasi, 2017). ConceptNet contains a massive dictionary of related words with a variety of relation types in many languages. For example, once ConceptNet is parsed to only show English words with relations relevant to my work, "abandon synonym quit 1.0" indicates that "quit" is a synonym for "abandon" with a confidence score of 1.0. I also use the relation *hasproperty* to find adjectives. For example, "apple hasproperty red 9.592" indicates that "red" is an adjective for apple. ConceptNet's *hasproperty* relation also includes non-adjective phrases (such as "antlers hasproperty shaped_like_trees," so I only use single-word properties.

A major downside to using ConceptNet for word manipulation is that it does not take the sentence context into account when selecting related words. As an example, the original CoNLL03 sentence *"The presidential contest put the stock market in the twilight zone"* was changed to *"The presidential contest put the broth market in the twilight zone"*. In this case, "stock" was changed to "broth" because they are synonyms in the context of soups. However, in the context of this sentence, the substitution does not make sense.

Selecting related words: BERT

In order to use the full sentence as context when deciding on a replacement word, I used BERT's masked token prediction capability (Devlin et al., 2018). BERT (Bidirectional Encoder Representations from Transformers) is an NLP model that can be fine-tuned for specific tasks, such as NER or question answering. In order to train a deep representation from unlabeled text, BERT is pre-trained by masking tokens in the unlabeled text and learning to predict those masked tokens (Devlin et al., 2018).

In order to get a related word, I mask the word I want a synonym of or insert a mask before the noun I want to add an adjective to. I then use BERT's masked token prediction functionality to generate a list of possible words for the mask. For example, if I wanted a synonym for "stock" in *"The presidential contest put the stock market in the twilight zone."* I would input the sentence *"The presidential contest put the [MASK] market in the twilight zone."*. In this example, BERT suggests the words "entire", "local", "fish", "stock", and "black", giving a corresponding confidence score for each. Of course, I cannot use "stock" because that was the original world. In contrast to ConceptNet, which suggested "broth," BERT shows an understanding of the sentence's context.

Currently, I am using the suggested related word with the highest confidence rating, with the intuition that this is the most likely to form a valid sentence. However, my next direction will be to use lower-ranking words. My hypothesis is that these lower-ranking words will be more effective as adversarial attacks because they are less common words.

Results of context-based attacks: related word selection

As can be seen in Table 3.3 and Fig 3.5, BERT's adversarial attacks tend to be more effective across a variety of experiments. The specifics of these experiments are explained in Sec 3.2.2 and Sec 3.2.2. Additionally, BERT is more likely to create semantically meaningful sentences because it takes into account the context of a sentence, as discussed in Sec 3.2.2. For this

reason, for the remainder of the experiments I only present results using BERT to select related words unless otherwise specified.

Adversarial Technique	Performance on dev data			Performance on test data		
	Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Original data	94.86	94.50	94.68	91.03	90.91	90.97
Insert adjectives randomly (BERT)	88.46	82.63	85.45	86.03	85.4	85.71
Insert adjectives randomly (ConceptNet)	93.10	85.71	89.26	87.10	90.00	88.52
Replace words randomly (BERT)	93.35	92.79	93.07	90.30	89.33	89.81
Replace words randomly (ConceptNet)	95.44	94.96	95.20	92.67	91.79	92.23
Insert adjectives using AllenNLP Interpret (BERT)	94.49	94.61	94.55	90.88	90.79	90.83
Insert adjectives using AllenNLP Interpret (ConceptNet)	93.71	94.29	94.00	90.99	90.76	90.87
Replace words using AllenNLP Interpret (BERT)	95.35	95.18	95.26	92.89	92.12	92.50
Replace words using AllenNLP Interpret (ConceptNet)	95.91	95.80	95.86	93.28	92.99	93.13

Table 3.3 Comparison of of BERT and ConceptNet for selecting related words. This experiment was performed using our baseline model and the CoNLL03 dataset.

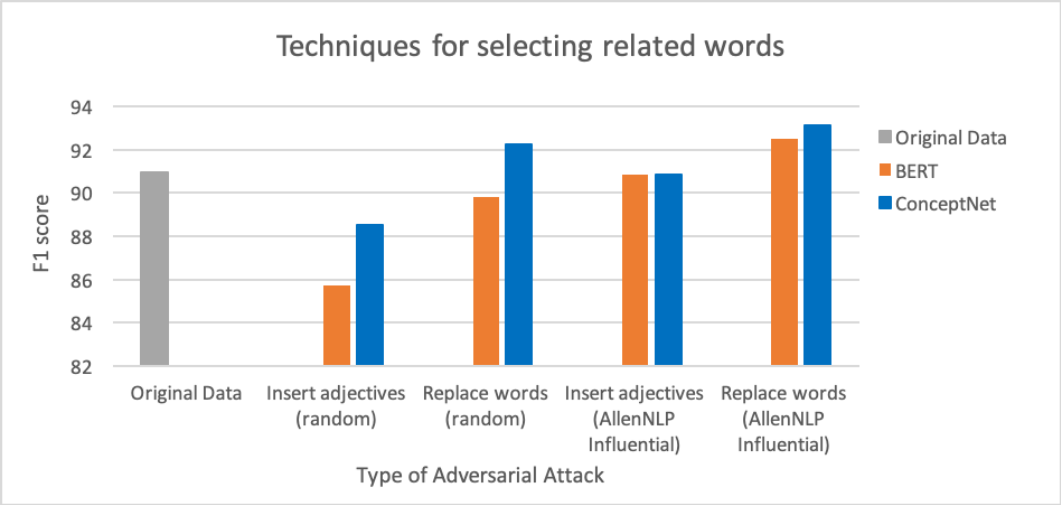

Figure 3.5 Comparison of of BERT and ConceptNet for selecting related words. This experiment was performed using our baseline model and the CoNLL03 test set.

Influential word selection: random

The baseline for selecting influential words is to randomly select them. In order to investigate whether words closer to entities are more influential, I use two subcategories of random selection: manipulating words immediately preceding entities and manipulating words immediately preceding non-entities. As can be seen in Tab 3.4 and Fig 3.6, the context-based

adversarial attacks are more effective when words adjacent to entities are manipulated. For this reason, throughout this thesis random selection means randomly selecting words before entities unless otherwise specified.

	Performance on dev data			Performance on test data		
Adversarial Technique	*Precision*	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Original data	94.86	94.50	94.68	91.03	90.91	90.97
Remove adjectives before ents	89.47	82.93	86.08	80.65	78.12	79.37
Remove adjectives before non-ents	93.54	93.16	93.35	89.48	88.89	89.18
Insert adjectives before ents	88.46	82.63	85.45	86.03	85.4	85.71
Insert adjectives before non-ents	93.29	93.26	93.27	88.48	88.48	88.48
Replace words before ents	93.35	92.79	93.07	90.30	89.33	89.81
Replace words before non-ents	94.41	94.16	94.28	91.22	91.31	91.26

Table 3.4 Comparison of manipulating randomly-chosen words before entities versus before non-entities in context-based attacks. This experiment was performed using our baseline model and the CoNLL03 dataset.

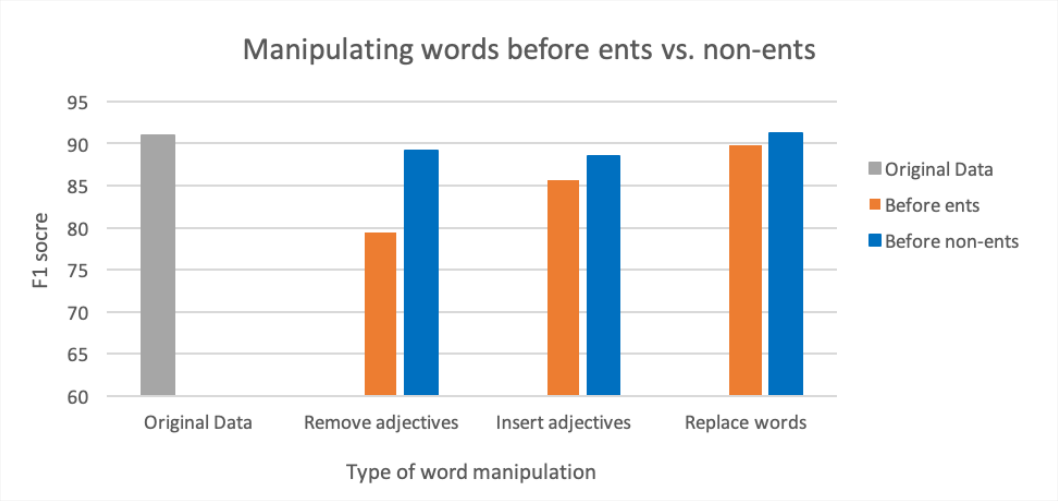

Figure 3.6 Comparison of manipulating randomly-chosen words before entities versus before non-entities in context-based attacks. This experiment was performed using our baseline model and the CoNLL03 test set.

Influential word selection: AllenNLP Interpret

The first non-random way for selecting influential words that I investigate is using AllenNLP Interpret (E. Wallace et al., 2019). AllenNLP Interpret can be applied to any differentiable NLP model. By differentiating the model and finding the gradient of the loss (with respect to

the model's actual output) for each token, AllenNLP Interpret determines which input word is the most influential for the model's prediction (E. Wallace et al., 2019). I use AllenNLP Interpret to select the top k most influential words, which I manipulate through removal, insertion, and replacement.

The benefit of AllenNLP Interpret is that it takes a model-specific approach to quantitatively and definitively selecting the most influential words. However, the downside is that it requires a white-box approach to adversarial attacks, meaning that in order to develop the adversarial examples, we need to know the structure of the model at hand. Additionally, the model has to be end-to-end differentiable so that we can calculate the gradient of the loss.

Influential word selection: Cooccurrence Matrix

A cooccurrence matrix is an approach to finding influential words that is model agnostic. In this approach, I create a dictionary with every non-entity word in the corpus. For each word in the corpus, I record the number of times that it appears in the same sentence as an entity of each type. For example, the sentence "*Costco* has taken off in *Canada*" would increment the counters for each entry in the set {(has, `ORG`), (taken, `ORG`), (off, `ORG`), (in, `ORG`), (has, `LOC`), (taken, `LOC`), (off, `LOC`), (in, `LOC`)}. I considered ignoring stop words (words that are commonly filtered out in NLP such as "the", and "and"). However, some stop words might have an association with a particular entity type. For example, "in" and "at" are highly associated with `LOC`.

In order to determine a word's influence on the model's prediction for a sentence, I sum the word's influence over all of the sentence's entities. To determine the influence of each entity-word combination, two values must be taken into account. First we must consider how many times the word and entity type have been seen together in the same sentence throughout the corpus. This is exactly the information stored in the cooccurrence matrix. This value is important because it is likely that the model has learned which words are most often associated with a given entity type, making these words influential. For example,

the word "said" often appears in sentences with a `PER` entity. In fact, according to the cooccurrence matrix I developed on the CoNLL03 training data, "said" is the word most frequently associated with `PER` besides "the."

However, neither "said" nor "the" are assigned the highest influence rating. This is because there is a second value to consider: the relative frequency of a word. If we strictly consider the frequency with which a word and entity type appear together, "the" would rank among the most influential words for every entity type. The relative frequency is the ratio of the frequency with which a word is associated with a specific entity type compared to the frequency of the word throughout the entire corpus. This value gives an idea of how influential a word is to a specific entity type rather than merely indicating how prevalent that word is throughout the corpus. This ratio alone is not enough to accurately represent a word's influence because extremely infrequent words would end up with a high ratio despite their lack of influence. For example, according to the cooccurrence matrix I developed on the CoNLL03 training data, the pair ("pavilion", `PER`) has a perfect relative frequency of 1.0. However, this is because the word pavilion only appears once in the corpus. In order to accurately represent a word's influence on an entity's type, I combine these two indicators such that the influence of a specific word/entity type pair is calculated as follows:

$$\text{influence} = \text{total associations} \times \text{relative frequency}$$

Results of context-based attacks: influential word selection

As seen in Table 3.5 and Fig 3.7, random selection of words is the most effective way to create adversarial attacks in this context-based setting and AllenNLP Interpret is the least effective. This result was surprising considering that AllenNLP Interpret leverages knowledge of the structure of the NER model. My next step is to further investigate these results, looking

through the specific successful attacks to see why random selection of influential words seems most effective.

Adversarial Technique	Performance on dev data			Performance on test data		
	Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Original data	94.86	94.50	94.68	91.03	90.91	90.97
Remove adjectives randomly before ents	89.47	82.93	86.08	80.65	78.12	79.37
Remove adjectives using AllenNLP Interpret	94.39	93.97	94.18	91.34	90.93	91.13
Remove adjectives using cooccurrence matrix	93.54	93.09	93.31	89.48	88.9	89.19
Insert adjectives randomly before ents	88.46	82.63	85.45	86.03	85.4	85.71
Insert adjectives using AllenNLP Interpret	94.49	94.61	94.55	90.88	90.79	90.83
Insert adjectives using cooccurrence matrix	92.96	93.01	92.99	88.52	88.55	88.54
Replace words randomly before ents	93.35	92.79	93.07	90.30	89.33	89.81
Replace words using AllenNLP Interpret	95.35	95.18	95.26	92.89	92.12	92.50
Replace words using cooccurrence matrix	94.97	94.55	94.76	92.14	91.76	91.95

Table 3.5 Comparison of techniques for choosing influential words for context-based attacks. This experiment was performed using our baseline model and the CoNLL03 dataset.

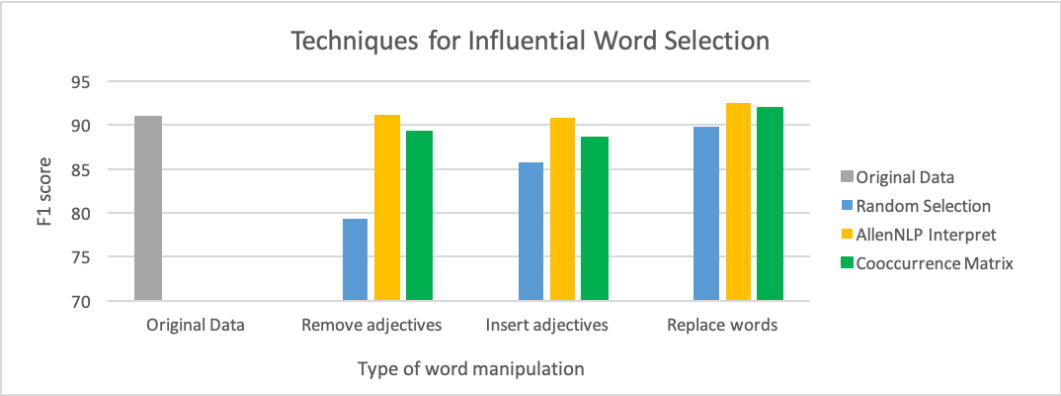

Figure 3.7 Comparison of techniques for choosing influential words for context-based attacks. This experiment was performed using our baseline model and the CoNLL03 test set.

3.3 Adversarial Training Experiments

Although the main purpose of the adversarial augmentation techniques was to *test* the robustness of a model, they can also be used to *improve* the robustness of the model to these same types of adversarial attacks. In this section, I investigate the efficacy of adversarial training using the adversarial augmentation techniques described in Sec 3.2.

3.3.1 Adversarial Training Technique

Adversarial training uses the same augmentation scripts as were used for adversarial attacks. The first step is to use these scripts on the training and dev data. For each adversarial augmentation technique, I make an augmented training file that includes the original training data as well as the augmented training data. I make an augmented dev file in the same manner. I then train a new NER model (of the same BLSTM-CRF structure as the baseline model) on this augmented training data. Finally, I test the augmented model's performance on the adversarial test dataset in order to evaluate whether the adversarial training improved the model's robustness to the adversarial attack. Additionally, I test the augmented model's performance on the original CoNLL03 test dataset to evaluate whether the adversarial training was detrimental to the model's success on the original data.

3.3.2 Adversarial Training Results

As seen in Table 3.6 and Fig 3.8, the augmented NER models' performance on the adversarial test dataset tends to be higher than the original baseline model, demonstrating the efficacy of adversarial training. Further, as seen in Table 3.6 and Fig 3.9, this improvement comes without a significant decrease in performance on the original test dataset, evidencing that adversarial training increases the model's robustness without harming its performance on standard data. For brevity, I only present the adversarial training results on the most successful entity-based and context-based attacks. However, the other augmentation techniques produce a similar pattern.

Augmentation Type	Original model on adversarial data			Augmented model on adversarial data			Augmented model on original data		
	Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Original model on original data	-	-	-	-	-	-	91.03	90.91	90.97
All ents fully masked	63.94	59.54	61.66	76.02	76.49	76.25	89.63	90.16	89.89
All ents half masked	66.16	62.20	64.12	77.56	77.71	77.63	90.38	90.78	90.58
All ents masked, retain vowels	64.68	60.82	62.69	75.55	76.03	75.79	90.37	91.02	90.69
One ent fully masked	85.68	83.60	84.63	88.13	88.19	88.16	90.47	91.25	90.86
One ent half masked	86.12	84.23	85.16	88.60	88.73	88.66	90.66	91.64	91.15
One ent masked, retain vowels	85.62	83.77	84.68	88.22	88.06	88.14	90.52	91.27	90.89
Remove adjectives	80.65	78.12	79.37	77.42	75.00	76.19	91.36	90.93	91.14
Insert adjectives	86.03	85.40	85.71	88.41	89.05	88.73	91.24	90.88	91.06
Replace words	90.30	89.33	89.81	90.79	90.61	90.70	90.75	91.18	90.97

Table 3.6 Comparison of the original model's performance and augmented model's performance on adversarial datasets based on the CoNLL03 dataset. Related words were chosen using BERT. Influential words are selected as random words before entities.

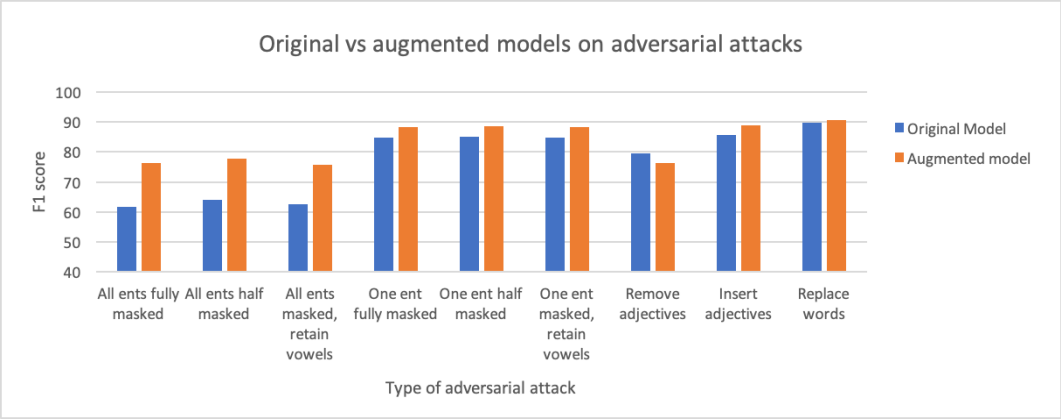

Figure 3.8 Comparison of the original model's performance and augmented model's performance on adversarial datasets based on the CoNLL03 dataset. Related words were chosen using BERT. Influential words are selected as random words before entities.

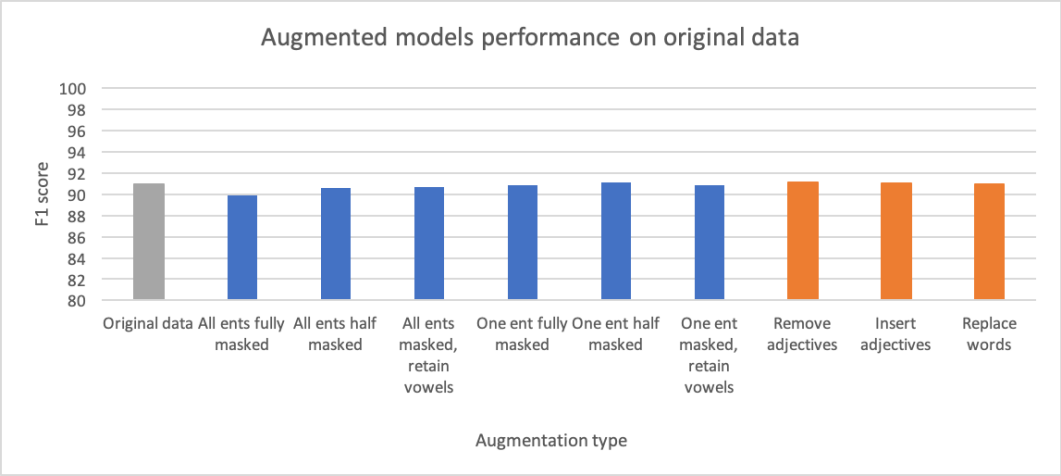

Figure 3.9 Comparison of the augmented models' performance and the original model's performance on the original CoNLL03 test dataset. Related words were chosen using BERT. Influential words are selected as random words before entities.

3.4 Future Directions

This is an ongoing project. One of my next steps is to thoroughly examine the results from context-based attacks presented in Sec 3.2.2 to investigate why random selection of influential words was more effective than systematic selection. I also plan to continue developing the related word selection with BERT by using suggested words with a lower confidence score in the hopes that they are more likely to fool the baseline model because they are less common words.

More generally, I plan to apply these adversarial augmentation techniques to other NER datasets. CoNLL03 is a limited dataset because it is relatively simplistic, containing only four categories of named entities. Further, the CoNLL03 dataset contains many labeling mistakes. In particular, because the dataset contains many sporting event results, there are many discrepancies as to whether a country name such as "Japan" should be counted as an LOC or ORG in the context of a sports team.

I would also like to run human experiments to ensure that the adversarial data is able to be labeled by humans. Evidence that humans can properly label the adversarial data would confirm that the adversarial data is in fact valid.

Chapter Four

Conclusion

In this thesis I present two approaches to increasing the efficiency of human annotations in named entity recognition tasks. First, I offer a novel technique for collecting human-annotated data by gathering both ground-truth entity labels and entity triggers. I present experiments which evidence the efficacy of entity trigger collection, showing that our framework trained on only 20% of the available trigger-annotated training sentences results in a comparable performance to conventional approaches using 70% of the training data.

I also investigate a variety of methods to adversarially augment human-annotated data, creating new examples which retain the original labels, are semantically valid, and are difficult for existing models to label correctly. I experimentally demonstrate that the augmentation algorithms I developed are effective at fooling state-of-the-art NER models. Further, I provide evidence that adversarial training using these same augmentation techniques is effective at making models more robust to adversarial attacks without damaging the models' performance on standard data.

My work on adversarial augmentation is ongoing. I plan to further explore ways to improve my adversarial attacks. I also plan to test my augmentation algorithms on datasets other than CoNLL03. Finally, I would like to combine the topics of entity triggers and adversarial examples by adversarially attacking the trigger matching module and using adversarial training to make this module more robust.

REFERENCES

Agarwal, Oshin et al. (2020a). *Entity-Switched Datasets: An Approach to Auditing the In-Domain Robustness of Named Entity Recognition Models*. arXiv: 2004.04123 [cs.CL].

— (2020b). *Interpretability Analysis for Named Entity Recognition to Understand System Predictions and How They Can Improve*. arXiv: 2004.04564 [cs.CL].

Alzantot, Moustafa et al. (2018). "Generating Natural Language Adversarial Examples". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 2890–2896. DOI: 10.18653/v1/D18-1316. URL: https://www.aclweb.org/anthology/D18-1316.

Araujo, Vladimir et al. (2020). *On Adversarial Examples for Biomedical NLP Tasks*. arXiv: 2004.11157 [cs.CL].

Athalye, Anish et al. (2017). *Fooling Neural Networks in the Physical World*. URL: https://www.labsix.org/physical-objects-that-fool-neural-nets/.

Blodgett, Su Lin, Lisa Green, and Brendan O'Connor (2016). "Demographic Dialectal Variation in Social Media: A Case Study of African-American English". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 1119–1130. DOI: 10.18653/v1/D16-1120. URL: https://www.aclweb.org/anthology/D16-1120.

Cao, Yixin et al. (2019). "Low-Resource Name Tagging Learned with Weakly Labeled Data". In: *Proc. of EMNLP*.

Devlin, Jacob et al. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Dos Santos, Cicero and Bianca Zadrozny (2014). "Learning Character-level Representations for Part-of-Speech Tagging". In: vol. 5.

Hadsell, Raia, Sumit Chopra, and Yann LeCun (2006). "Dimensionality reduction by learning an invariant mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE, pp. 1735–1742.

Hancock, Braden et al. (2018). "Training Classifiers with Natural Language Explanations". In: *Proceedings of the conference. Association for Computational Linguistics. Meeting* 2018, pp. 1884–1895.

Iyyer, Mohit et al. (2018). "Adversarial Example Generation with Syntactically Controlled Paraphrase Networks". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1875–1885. DOI: 10.18653/v1/N18-1170. URL: https://www.aclweb.org/anthology/N18-1170.

Jia, Robin and Percy Liang (2017). *Adversarial Examples for Evaluating Reading Comprehension Systems*. arXiv: 1707.07328 [cs.CL].

Jia, Robin, Aditi Raghunathan, et al. (2019). "Certified Robustness to Adversarial Word Substitutions". In: *EMNLP/IJCNLP*.

Lample, Guillaume et al. (2016). "Neural Architectures for Named Entity Recognition". In: *CoRR* abs/1603.01360. arXiv: 1603.01360. URL: http://arxiv.org/abs/1603.01360.

Lee, Dong-Ho et al. (2020). "Entity Triggers: Learning with Explanations for Named Entity Recognition". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Li, Jiao et al. (2016). "BioCreative V CDR task corpus: a resource for chemical disease relation extraction". In: *Database : the journal of biological databases and curation* 2016.

Li, Shen, Hengru Xu, and Zhengdong Lu (2018). "Generalize Symbolic Knowledge With Neural Rule Engine". In: *ArXiv* abs/1808.10326.

Lin, Bill Yuchen et al. (2019). "AlpacaTag: An Active Learning-based Crowd Annotation Framework for Sequence Tagging". In: *ACL*.

Lin, Zhouhan et al. (2017). "A Structured Self-Attentive Sentence Embedding". In: *Proc. of ICLR*.

Lipton, Zachary Chase and Byron C. Wallace (2018). "Practical Obstacles to Deploying Active Learning". In: *EMNLP/IJCNLP*.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025*.

Ma, Xuezhe and Eduard H. Hovy (2016). "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF". In: *Proc. of ACL*.

Nadeau, David and Satoshi Sekine (2007). "A survey of named entity recognition and classification". In: *Linguisticae Investigationes* 30.1. Publisher: John Benjamins Publishing

Company, pp. 3–26. URL: http://www.ingentaconnect.com/content/jbp/li/2007/00000030/00000001/art00002.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "Glove: Global Vectors for Word Representation". In: *EMNLP*.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2018). "Semantically Equivalent Adversarial Rules for Debugging NLP models". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 856–865. DOI: 10.18653/v1/P18-1079. URL: https://www.aclweb.org/anthology/P18-1079.

Riggio, Christopher (2019). *What's the deal with Accuracy, Precision, Recall and F1?* URL: https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021.

Rome, Scott (2018). *Understanding Attention in Neural Networks Mathematically*. URL: https://srome.github.io/Understanding-Attention-in-Neural-Networks-Mathematically/.

Rosenberg, Chuck, Martial Hebert, and Henry Schneiderman (2005). "Semi-Supervised Self-Training of Object Detection Models". In: *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1* 1, pp. 29–36.

Shang, J. et al. (2018). "Learning named entity tagger using domain-specific dictionary". In: *Proc. of EMNLP*.

Shen, Yanyao et al. (2018). "Deep Active Learning for Named Entity Recognition". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ry018WZAZ.

Speer, Robyn, Joshua Chin, and Catherine Havasi (2017). "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, pp. 4444–4451.

Tjong Kim Sang, Erik F and Fien De Meulder (2003). "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition". In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics.

Viterbi, A. (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* 13.2, pp. 260–269.

Wallace, Eric et al. (2019). *AllenNLP Interpret: A Framework for Explaining Predictions of NLP Models*. arXiv: 1909.09251 [cs.CL].

Wang, Ziqi et al. (2020). "Learning from Explanations with Neural Execution Tree". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rJlUt0EYwS.

Yadav, Vikas and Steven Bethard (2019). "A Survey on Recent Advances in Named Entity Recognition from Deep Learning models". In: arXiv: 1910.11470 [cs.CL].

Yang, Y. et al. (2018). "Distantly Supervised NER with Partial Annotation Learning and Reinforcement Learning". In: *Proc. of ACL*.

Zhou, Wenxuan et al. (2020). "NERO: A Neural Rule Grounding Framework for Label-Efficient Relation Extraction". In: *Proc. of World Wide Web Conference (WWW)*.

Zhu, Mengdi et al. (2019). "Towards Open-Domain Named Entity Recognition via Neural Correction Models". In: *arxiv*.