Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Weak supervision approaches for Named Entity Recognition

Relatori:
Matteo Luigi Palmonari
Leonardo Rigutini

Relazione della prova finale di:
Vittorio Maggio
Matricola 817034

Anno Accademico 2021-2022

Ringraziamenti

Desidero ringraziare l'azienda Expert.ai che mi ha ospitato durante tutto il percorso di stage, in particolare Leonardo Rigutini mio supervisore e Andrea Zugarini che mi ha seguito per tutto il percorso, dando un apporto fondamentale al mio lavoro.
Un ringraziamento va anche al mio relatore Matteo Palmonari, per la sua disponibilità mostrata dall'inizio della mia esperienza di stage alla stesura dell'elaborato.
Ringrazio i miei genitori Francesco e Rita che mi hanno supportato per tutto il percorso universitario, mio zio Aurelio, è grazie a lui che mi sono appassionato a questo fantastico mondo che è l'informatica.
Ringrazio mio fratello Giovanni per essere sempre stato al mio fianco.
Ringrazio tutti gli amici e le persone che ho incontrato durante questi anni.
Infine, non per importanza, ringrazio i miei nonni, ai quali dedico qeusta tesi.

Acknowledgements

I would like to thank the Expert.ai company that hosted me throughout my internship, in particular Leonardo Rigutini, my supervisor, and Andrea Zugarini, who followed me throughout, giving a fundamental contribution to my work.
Thanks also go to my supervisor Matteo Palmonari, for his willingness shown from the beginning of my internship experience to the writing of the thesis.
I thank my parents Francesco and Rita, who supported me throughout my university journey, and my uncle Aurelio, it is thanks to him that I became passionate about this amazing world that is computer science.
I thank my brother Giovanni for always being by my side.
I thank all the friends and people I have met during these years.
Last but not least, I thank all my grandparents, to whom I dedicate this thesis.

Abstract

Over the last few years, the amount of textual information has rapidly increased. Named Entity Recognition is one of the most valuable and popular tasks to extract information from these texts. The main approaches to perform this task are: rule-based, machine learning, and hybrid approaches. In this thesis, I focus on supervised machine learning approaches, where the main problem of the traditional technique is that they don't take into account the noise of the supervision while, in the real world, we have to deal with noisy labels (because of annotator errors, lack of information or because tags were automatically extracted from knowledge bases). I analyze the state-of-the-art techniques found in the literature; I conduct experiments using a noise-robust loss function and propose an extension of it, demonstrating an improvement in F1 score compared both with the original loss and the cross-entropy loss (usually used for this task).

Contents

Chapter 1

Introduction

Over the last few years, the amount of textual information has rapidly increased. A report from the International Data Corporation (IDC) [1] shows that 80% of worldwide data will be unstructured by 2025. This large amount of data, when analyzed, can give a significant competitive advantage to companies to improve their businesses, speed up processes, analyze conversations made on the web, and be able to make better data-driven decisions. In this scenario, the role of Information Extraction (IE) has become crucial, and one of the most critical tasks in extracting information from unstructured text is the Named Entity Recognition (NER).

In this thesis, I delve into the main problem with state-of-the-art approaches for the NER task, namely, the fact that in the real world, companies have to deal with noisy datasets, in which the correctness of all labels is not certain. This happens because the labeling process is a very expensive process, so to reduce costs, companies often adopt automatic labeling techniques or do not employ domain experts for that process. The literature refers to this type of dataset as "Weakly labeled datasets", I then analyze the different approaches by focusing especially on the Weak Supervision approaches, which, unlike the other approaches, are mainly based on adapting the loss function used for Machine Learning models, to make it take into account the potential presence of mislabels, not requiring additional data (such as rules or textual patterns), thus reflecting the most common case where only a weakly labeled dataset is available. I dwell on a state-of-the-art approach to Weak supervision for the NER task (presented by Yu Meng et al. [2]) and make my contribution by proposing

two extensions of it: the first that simplifies the model training process (no longer necessitating the slef-training phase adopted by Yu Meng et al. [2]) and the second that considers the case where it is known that there are labels within the dataset that are noisier than others, by going to adapt the loss function according to the label and the respective level of noise.
Finally, I compare the results from the proposed losses with those from the original noise robust loss and Cross Entropy loss, a function usually used in the state-of-the-art for NER task, showing the improvements over both on the F1 score evaluation metric.

The thesis is structured as follows: in the introductory chapter, I introduce the Named Entity Recognition task, define it, analyze the state of the art, and then describe the main limitation of these approaches. In the second chapter, I describe how transformer-based language representation models work. They form the basis of all approaches not only for the task of Named Entity Recognition but also for other tasks in the field of Natural Language Processing. In the third chapter, I summarize the study done on the state of the art on the problem of Named Entity Recognition with Weak Labels, analyzing and comparing the three main approaches. Then in Chapter 4, I analyze the main limitations of the state the art of chosen approach after the comparison, and I propose a solution and extension. In Chapter 5, I present the experiments done and their objectives, then analyze the results in Chapter 6. The thesis concludes with an analysis of the results obtained and possible future developments.

1.1 Named Entity Recognition

Named entity recognition is a subtask of information extraction that seeks to segment and classify text fragments (named entities) into predefined classes. Entities can be, for example, names of people, organizations, locations, or any other class according to the text domain.

Example:

Apple CEO Tim Cook introduces the new iPhone at Cupertino

↓ NER

Apple[0RG] *CEO* ***Tim Cook[PERS]*** *introduces the new iPhone at* ***Cupertino[LOC]***

where ORG = Organization, PERS = Person, LOC = Location

The main approaches to NER are:

Lexicon-based approach : consists of applying dictionaries (called gazetteers) that contain a list of words associated with the corresponding entity class. This approach seems to be very simple but, at the same time, has many limitations: entities that don't appear in the dictionary aren't classified; it assigns the same class to a word regardless of context, while some words change in meaning when put in a different context;

Rule-based approaches : consist of applying rules to detect and classify the entities. In this approach, a complete and correct definition of the rules is crucial to find and disambiguate the entities;

Machine learning approaches : are based on probabilistic models. They use a set of examples to train and test a model that later can be used to make inferences on new sentences. Before the training phase, you need to obtain the word embeddings, which are vectors (typically real-valued) that encode the meaning of the word such that the words closer in the vector space are expected to be similar in meaning. According to the dataset that you have, you can use three main machine learning techniques:

- Unsupervised: if you have an unlabeled dataset, so you have a set of examples of sentences, but you don't know where the entities are and their class.
- Semi-supervised: if you have a small labeled dataset and a large set of unlabeled data.
- Supervised: if you have only a labeled dataset.

This approach is the one that, in many cases, provides the best performance, but the limitation is that the outputs of the models have low explainability;

Hybrid approaches : combine machine learning approaches with rule-based systems to improve not only the performance of the models but also the explainability of the results.

In this thesis, I will focus on supervised learning approaches, the most common scenario.

1.2 Supervised Named Entity Recognition

Currently, the state-of-the-art models for supervised NER are based on transformers [3] like BERT [4]. The introduction of transformers represents a turning point for all the Natural Language Processing tasks. Previously, were used non-contextualized embeddings such as Word2Vec [5], GloVe [6], and fastText [7], these models help lots of NLP tasks, but the main limitation is that they represent a word with the same vector, regardless of the context. While, in natural language, a word used in different contexts could have different meanings, so a different vector representation is needed.

Example:

1. *Stock **rally** follows biggest hedge-fund shorting binge since 2008*
2. *Petrucci exceeds expectations in maiden Dakar **Rally***

In the example the word *rally* ah two completely different meanings (the first refers to stock market performance, the second to sports), so you have to represent them with different and distant vectors.

Later, word embeddings models based on Long Short-Term Memory (LSTM)[8] were proposed. They are a type of recurrent neural network capable of learning order dependence in sequences, allowing information to persist. So with LSTM, you can obtain word embedding that includes information about the context of the word. In this way, the same word has a different representation depending on its context. A

pretrained contextualized word embedding model based on Bidirectional LSTM is ELMo[9], it significantly outperforms previous state-of-art approaches for NLP tasks. Recently Delvin et al. proposed BERT, which encodes contextualized word information by Transformers. A transformer is a deep learning model that adopts the self-attention mechanism. Like recurrent neural networks, transformers are designed to process sequential input data, such as natural language. However, unlike RNNs, they process the entire input all at once, not one token at a time. In this way, you can get an embedding that includes the contextual information of the whole text, not only of the tokens near the word (as in LSTM). As anticipated, the transformers-based models are involved in the current state-of-art for many NLP tasks, such as NER. In chapter 2 I go into more detail about how BERT and Transformers work.
After BERT, the research has focused on improving the BERT model performance, like with RoBERTa [10], which uses stronger masking strategies, or DistilBERT [11], which aims to reduce the model size while keeping the same performance. In literature, some other architectures were built on top of these models to get better results. Examples of the state-of-art are:

Automated Concatenation of Embeddings for Structured Prediction (2021) by Wang et al. [12]:
this work achieves state-of-the-art performance on different versions and languages of the CoNLL Dataset. They focused on obtaining better representation by concatenating different types of embeddings. To automate the process to get the best concatenation of different model embeddings (BERT, ElMo, GloVe, etc..) for a specific task, they propose a reward system inspired by reinforcement learning techniques. In the table 1.1 is reported the F1 score achieved.

Task	**Dataset**	**Model**	**F1**
NER	CoNLL 2003 (English)	ACE + document-context	94.6
NER	CoNLL 2003 (German)	ACE + document-context	88.38

Table 1.1: F1 scores ACE + document-context model

Dice Loss for Data-imbalanced NLP Tasks (2021) by Xiaoya Li et al. [13]:
The intuition behind this work is that the most commonly used cross-entropy loss is accuracy-oriented, which creates a discrepancy between training and test. Each training instance contributes equally to the objective function at training time, while the F1 score concerns more about positive examples at test time. So, They propose a new loss based on the Sørensen–Dice coefficient (Sorensen, 1948) or Tversky index (Tversky, 1977), which attaches similar importance to false positives and false negatives, and is more immune to the data-imbalance issue. Using this loss with dynamically adjusted weights, they achieve state-of-the-art for the NER task on the dataset Ontonotes v5 (English) in terms of F1 score, shown in table 1.2.

Task	**Dataset**	**Model**	**F1**
NER	Ontonotes v5 (English)	BERT-MRC+DSC	92.07
NER	CoNLL 2003 (English)	BERT-MRC+DSC	93.33

Table 1.2: F1 scores BERT-MRC+DSC model

The problem is that these models don't take into account the noise in the data, while in the real world, we have to deal with noisy labels (because of annotator errors, lack of information, or because labels were automatically extracted from knowledge bases). We call these labels Weak Labels. In the next paragraph, I give a more formal definition of them.

1.3 Problem definition

In the machine learning approach, the quality of the datasets is crucial to allow models to generalize as best as possible and perform well. But labeling a great quantity of data is a very expensive and time-consuming process, often requiring person-months or years to assemble, clean, and debug datasets, especially when domain expertise is required. Often, to speed up this process, more annotators are involved or are used auto-labeling algorithms based on knowledge bases. On top

of this, tasks change and evolve in the real world, for example, labeling guidelines, granularities, or downstream use cases often change, necessitating re-labeling. For all these reasons, labeling errors are frequent, and companies must deal with these weak labels. Trying to give a more formal definition, I define weak labels as labels with the following features:

- Incompleteness: some entity mentions may not be assigned with labels;
- Labeling Bias: some entity mentions may not be labeled with the correct types;
- Large-scale: often a large number of examples are available.

Examples:
Suppose the case where we have a dataset in which the following entity classes are labeled: Person (PERS), Organization (ORG) and Location (LOC). I mark with the label 'O' all words not belonging to any class.

- Incompleteness:

Bloomberg	is	actively	preparing	to	enter	the	Democratic	presidential	primary
-	O	O	O	O	O	O	ORG	O	O

In this example, the label on the token 'Bloomberg' is missing; note that in the case of using knowledge bases or automatic annotation methods, these may incorrectly label the entity as belonging to the class 'ORG' (thus referring to the company Bloomberg), while the correct label in this case the correct label is 'PERS'.

- Labeling Bias:

Rafael	Nadal	wins	his	14th	Rolland-Garros
O	PERS	O	O	O	ORG

In this example the entity 'Rafael' was labeled as not belonging to any of the classes (label 'O'), instead the correct label is 'PERS'. Such errors may be caused by annotator errors or because automatic labeling methods were used.

As seen in the previous example, they can also be a consequence of incomplete labels.

In my work, I focused on the case where you have a dataset with labeling errors. Before examining the approaches in the literature dealing with weak labels, in the next chapter, I delve into how BERT and Transformers work, elements underlying the models that represent state of the art.

Chapter 2

Language representation with transformers

As anticipated, Bidirectional Encoder Representations from Transformers (BERT) is a language model based on transformers. It is at the base of the state of art approaches for many Natural Language Processing tasks. Before explaining what transformers are, I need to introduce the concepts of encoder, decoder, attention, and self-attention.

2.1 Encoder-Decoder Model

Encoders and Decoders were initially introduced for the task of statistical machine translation in 2014 by Cho et al. [14]. It first uses an encoder to encode an input sequence into a context vector and uses this intermediate representation to generate an output sequence through the decoder. The encoder-decoder model proposed by Cho et al. is based on Recurrent Neural Networks, in detail:

Encoder: The encoder takes into input a sequence $x = (x_1, x_2, ..., x_i)$, compresses the first token x_1 into a fixed-length hidden state vector $\overline{h_1}$, then, for each following token, compresses the last hidden state $\overline{h_{i-1}}$ and the current token x_i to the respective hidden state $\overline{h_i}$:

$$\overline{h_i} = f(\overline{h_{i-1}}, x_i) \tag{2.1}$$

where f is a non-linear activation function. Applying the equation 2.1 to each token, you have the context vector c of the input sequence.

Decoder: The objective of the decoder is to generate a new output sequence of a fixed length m. It computes a hidden state h_i that takes the previous decoder hidden state h_{i-1}, the previous output y_{i-1}, and the context vector c:

$$h_i = f(h_{i-1}, y_{i-1}, c) \tag{2.2}$$

where f in a non-linear activation function. The output of the hidden state h_i is fed into another activation function g; this produces a probability distribution over the set of possible output tokens:

$$y_i = g(h_i) \tag{2.3}$$

Applying the equation 2.3 m times, you have the output sequence of tokens. In figure 2.1 the representation of the encoder-decoder model.

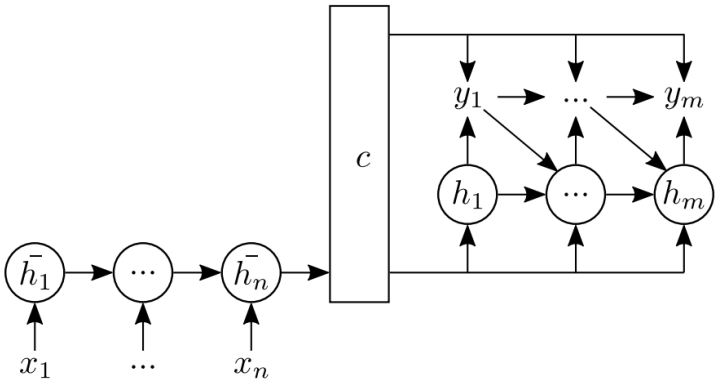

Figure 2.1: Encoder-Decoder model representation. On the left-hand side the encoder, on the right-hand side the decoder.

2.2 Attention mechanism

The Encoder-Decoder architecture's limitation is that it only passes the last hidden state from the encoder to the decoder. This leads to the problem that information has to be compressed into a fixed-length vector, and information can be lost in this compression, especially the information given by the first tokens in the sequence. A solution is to use bi-directional hidden layers to process the input in reversed order to represent both directions of the sentence. This helps for shorter sequences, but for the longer ones, the problem of loss of information persists. To solve this problem, Bahdanau, Cho, and Bengio [15] propose to use Bidirectional RNNs to compute the forward and the backward hidden states, so the input token representation is given by the concatenation of the forward and backward hidden states:

$$\overline{h_i} = [\overrightarrow{h_i}; \overleftarrow{h_i}] \tag{2.4}$$

So we have a representation for every token, and the context vector is computed as a weighted sum of the encoder hidden states h_i. The decoder learns the weight, in this way, we have m different context vectors (one for every element of the output), so the definition of decoder hidden states changes:

$$h_i = f(h_{i-1}, y_{i-1}, c) \tag{2.5}$$

With this new approach, the decoder decides which parts of the source sentence to pay attention to. For this reason, it is also known as 'global attention' [16]. Other attention mechanisms were later proposed for different tasks and with different architectures. A representation of the attention mechanism is in figure 2.2

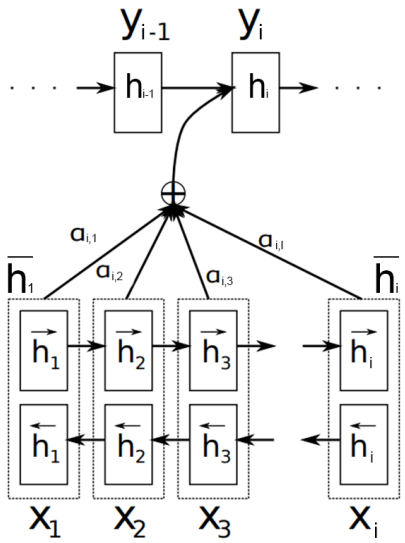

Figure 2.2: Attention mechanism representation. On the bottom the encoder and on the top the decoder.

2.2.1 Self-attention mechanism

The attention mechanism is restricted to the decoder step, so Lin et al. [17] propose a Self-attention mechanism that inglobes the attention in the embedding of the tokens, so in the encoder step. Lin et al. compute the hidden states with a Bi-RNN with u hidden units concatenating forward and backward hidden states (as in equation 2.4), obtained $H = (h_1, h_1, ..., h_n)$ with $H \in R^{n2u}$, the self-attention mechanism is given by:

$$a = softmax(w_{s2} tanh(W_{s1} H^T)) \tag{2.6}$$

where $W_{s1} \in R^{d_a 2u}$ is a learned matrix and w_{s2} is a learned vector of size d_a, where d_a is a abritrary hyperparameter. The final attention context vector is:

$$c = a^T H \tag{2.7}$$

Then Lin et al. extend this self-attention mechanism by replacing w_{s2} with a matrix $W_{s2} \in R^{r d_a}$, getting a matrix A:

$$A = softmax(W_{s2} tanh(W_{s1} H^T)) \tag{2.8}$$

so obtaining a context matrix C

$$C = AH \tag{2.9}$$

this allows to capture multiple parts of the input sentence.

2.3 Transformer Model

The use of self-attention in the encoder and the attention in the decoder significantly improve the performance of the basic encoder-decoder model, but the use of the attention mechanisms with the model based on RNN reduces the parallelization possibilities and the memory efficiency due to the sequential nature of RNN (it is processed one token at the time). To overcome these efficiency issues, Vaswani et al. propose a new architecture called Transformess [3]. A Transformer uses both in the encoder and in the decoder an extension of the attention mechanism: the Multi-Head Attention, which is described in the next subsection. In subsection 2.3.2 I describe the complete architecture of a Transformer.

2.3.1 Multi-Head Attention mechanism

Vaswani et al. propose an extension of the attention mechanism, the idea is to use a Scaled Dot-Product Attention (figure 2.4) which maps a query and a set of key-value pairs to an output:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.10}$$

Q is a set of queries, K is the keys, and V is the values. The scaling factor $\frac{1}{\sqrt{d_k}}$ is proved to improve the model's effectiveness. In a mechanism of self-attention, Query, Key, and Value are the same sentence (the input sentence) multiplied for the weight matrices, updated during the training. Figure 2.3 represent the computation of the matrices Q, K and V for the self-attention mechanism.

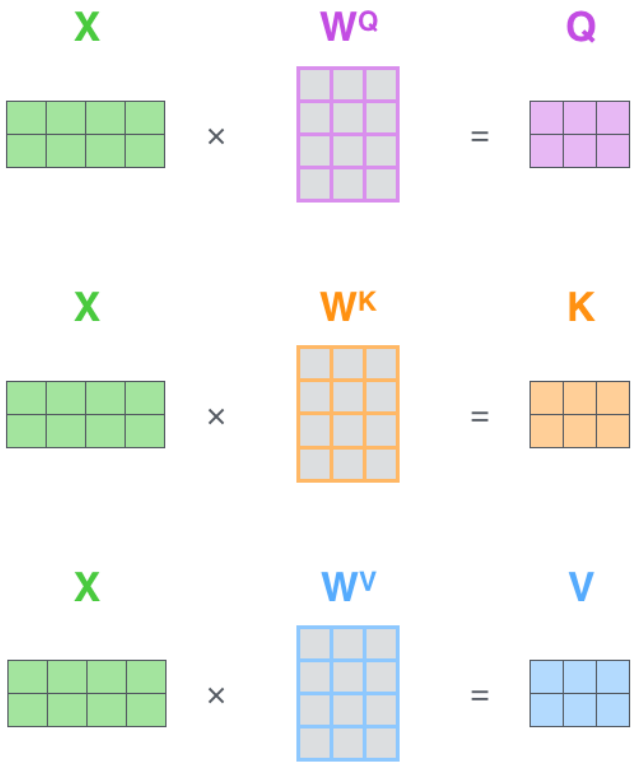

Figure 2.3: Input Transformation used in Transformers with self-attention, Q = Query, K = Key, V = Value [3].

These different transformations of the input are what enable the input to attend to itself. This, in turn, allows the model to learn about the context.
The multi-head component (figure 2.5) consists of dividing the embedding of each token of the sentences in h vectors, where h is the number of heads, computing in parallel the attention mechanism (equation 2.10), and at the end, concatenating the output vectors of each head.

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.11}$$

where the projections are learned parameters: $W_i^Q \in R^{d_{model}d_q}$, $W_i^K \in R^{d_{model}d_k}$,

$W_i^V \in R^{d_{model} d_v}$.

$$MultiHeadAttention(Q, K, V) = [head_1, ..., head_h]W^O \tag{2.12}$$

where the projection matrix $W^O \in R^{hd_v d_{model}}$ are learned parameters.

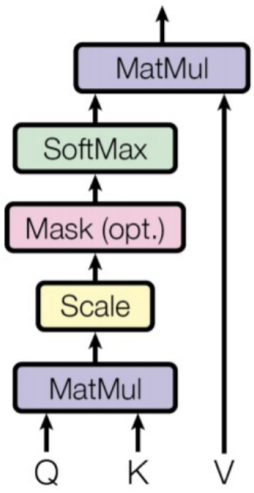

Figure 2.4: Scaled Dot Product [3]

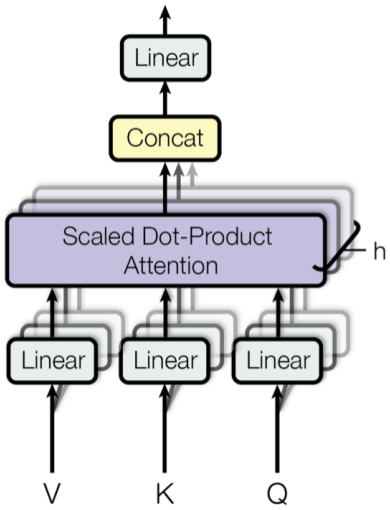

Figure 2.5: Multi-Head Attention [3]

2.3.2 Transformer architecture

The Transformer architecture comprises an encoder (left-hand side figure 2.6) and a decoder (right-hand side figure 2.6).

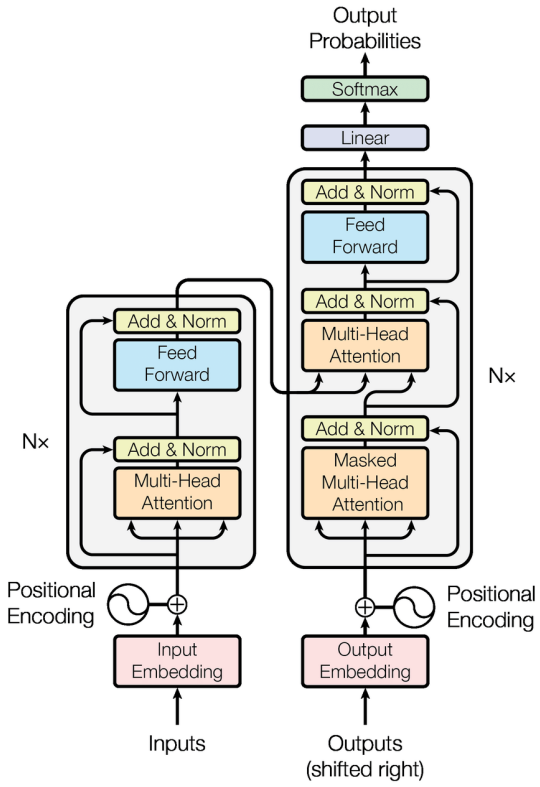

Figure 2.6: Transformer architecture [3]

The **encoder** is composed of N identical layers, each layer is composed of two sub-layers:

1. A Multi-Head self-attention mechanism, followed by a normalization layer;

2. A Feed-Forward neural network, followed by a normalization layer.

Each of the two components is surrounded by a residual connection. They allow gradients to flow through a network directly, without passing through non-linear activation functions. This help mitigates the vanishing gradient problem in deep networks.

The **decoder** also consists of a stack of N identical layers, each layer is composed of three sub-layers:

1. A Multi-Head self-attention that receives the previous output of the decoder stack. The decoder has to pay attention only to the preceding words (unlike the encoder, which considers the whole text) because a word's prediction can only depend on the previous words. For this reason, it is used a masking mechanism of the matrices Q and K (figure 2.4). Followed by a normalization layer.

2. A Multi-Head self-attention that receives the queries from the previous decoder sublayer and the keys and values from the output of the encoder. This allows the decoder to attend to all of the words in the input sequence. Followed by a normalization layer.

3. A Feed-Forward neural network, followed by a normalization layer.

A residual connection also surrounds the decoder's layers.
The encoder and the decoder input sequences are embedded and annotated with a positional encoding.

2.4 BERT Architecture and Training

BERT is a language model based on transformer with a variable number of layers and self-attention head. It has been trained on a massive dataset of 3.3 Billion words, specifically:

- Wikipedia dataset(2.5B words)

- Google's BooksCorpus (800M words)

This amount of information allows BERT to learn not only the English language but also information about the world. BERT has been trained using two strategies:

- Masked Language Modeling: it consists in masking 15% of tokens of the sentence and train the model to predict the masked words based on their context;

- Next Sentence Prediction: the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. In training, 50% correct sentence pairs are mixed in with 50% random sentence pairs to help BERT increase next sentence prediction accuracy.

The two tasks are trained together, so the objective function is the combined loss of the two strategies.

There are two original architectures of BERT, which differ in model size:

BERT base:

- Transformer layers: 12;
- Hidden size: 768;
- Attention Heads: 12;
- Parameters: 110M.

BERT large:

- Transformer layers: 24;
- Hidden size: 1024;
- Attention Heads: 16;
- Parameters: 340M.

Chapter 3

Related work

In this chapter, I analyze the main approaches present in literature about Named Entity Recognition with weak labels. In particular, they are:

- Weak supervision;
- Trigger NER;
- Prompting.

3.1 Weak Supervision

Weak supervision is the most popular approach to NER with weakly labeled data. It uses noise-robust loss functions to allow the model a better generalization and prevent overfitting to wrong labels.

3.1.1 Supervised approaches

In *Distantly-Supervised Named Entity Recognition with Noise-Robust Learning and Language Model Augmented Self-Training* by Yu Meng et al. [2], the authors have adapted the Generalized Cross-Entropy (GCE) loss function for the task of NER. The Generalized Cross-Entropy loss was introduced by Zhilu Zhang et al. in *Generalized Cross-Entropy Loss for Training Deep Neural Networks with Noisy Labels* [18]. It is based on the observation that the most used loss function for classification problems is the Cross-Entropy loss (equation 3.1), that due to the term at the denominator,

the tokens on which the model's prediction is less congruent with the provided labels will be implicitly weighed more during the gradient update 3.2, this grants better convergence with clean data (It emphasizes difficult tokens), but with noise labels, the model overfits the wrongly-labeled tokens.

$$\mathcal{L}_{CE} = -\sum_{i=1}^{n} log f_{i,y_i}(x;\theta) \tag{3.1}$$

$$\nabla_\theta \mathcal{L}_{CE} = -\sum_{i=1}^{n} \frac{\nabla_\theta f_{i,y_i}(x;\theta)}{f_{i,y_i}(x;\theta)} \tag{3.2}$$

In contrast, a noise-robust loss is the Mean Absolute Error (equation 3.3)which treats every token equally for gradient update (equation 3.4, unlike the cross-entropy, it doesn't put more emphasis on difficult tokens) but has worse convergence efficiency and effectiveness.

$$\mathcal{L}_{MAE} = \sum_{i=1}^{n} (1 - f_{i,y_i}(x;\theta)) \tag{3.3}$$

$$\nabla_\theta \mathcal{L}_{MAE} = -\sum_{i=1}^{n} \nabla_\theta f_{i,y_i}(x;\theta) \tag{3.4}$$

Generalized Cross-Entropy balance model convergence and noise-robustness, it is defined as follows:

$$\mathcal{L}_{GCE} = \sum_{i=1}^{n} \frac{1 - f_{i,y_i}(x;\theta)^q}{q} \tag{3.5}$$

where $0 < q < 1$ is a hyperparameter:

- when $q \to 1$, $\mathcal{L}_{GCE}$ approximates $\mathcal{L}_{MAE}$;
- when q $\to$ 0, $\mathcal{L}_{GCE}$ approximates $\mathcal{L}_{CE}$ (using L'Hôpital's rule).

the gradient is computed as:

$$\nabla_\theta \mathcal{L}_{CE} = -\sum_{i=1}^{n} \frac{\nabla_\theta f_{i,y_i}(x;\theta)}{f_{i,y_i}(x;\theta)^{1-q}} \tag{3.6}$$

They extend this loss with a weight w:

$$\mathcal{L}_{GCE} = \sum_{i=1}^{n} w_i \frac{1 - f_{i,y_i}(x;\theta)^q}{q} \tag{3.7}$$

when the model prediction is greater than a threshold τ (optimal threshold found: 0.7) the weight will be 1, otherwise, 0.
This weight system is based on the following intuition: since the loss function is noise-robust, the learned model will be dominated by the correct majority in the distant labels instead of quickly overfitting to label noise; if the model prediction disagrees with some given labels, they are potentially wrong.
Moreover, they do the model ensemble with different seeds to get a more robust model, minimizing the Kullback–Leibler divergence. Then, they perform self-training by generating new examples by feeding a partially masked original sequence into a pre-trained RoBERTa model and sample from its output probability to obtain an augmented sequence. In the final step, the NER model is trained with both original and augmented sequences.

Evaluation

In table 3.1 the evaluation of the model (named RoSTER).

	Methods	CoNLL03			OntoNotes5.0			Wikigold		
		Pre.	**Rec.**	**F1**	**Pre.**	**Rec.**	**F1**	**Pre.**	**Rec.**	**F1**
Distant-Sup.	**Distant Match**	0.811	0.638	0.714	0.745*	0.693*	0.718*	0.479	0.476	0.478
	Distant RoBERTa	0.837*	0.633*	0.721*	0.760*	0.715*	0.737*	0.603*	0.532*	0.565*
	AutoNER	0.752	0.604	0.670	0.731*	0.712*	0.721*	0.435	0.524	0.475
	BOND	0.821	0.809	0.815	**0.774***	0.701*	0.736*	0.534	0.686	0.600
	RoSTER (Ours)	**0.859**	**0.849**	**0.854**	0.753	**0.789**	**0.771**	**0.649**	**0.710**	**0.678**
Sup.	**BiLSTM-CNN-CRF**	0.914	0.911	0.912	0.888*	0.887*	0.887*	0.554	0.543	0.549
	RoBERTa	0.906*	0.917*	0.912*	0.886*	0.890*	0.888*	0.853	0.876	0.864

Table 3.1: Performance of Yu Meng et al. model (RoSTER) compared with other weak supervision approaches [2]

Another recent approach to weak supervision was proposed by Yue Yu et al. in *Fine-Tuning Pre-trained Language Model with Weak Supervision: A Contrastive-Regularized Self-Training Approach*[19]. After a fine-tuning process of BERT using the cross-entropy loss, they perform self-training minimizing the Kullback–Leibler

(KL) divergence, regularized by a contrastive regularizer and a confidence regularizer.

$$\mathcal{L}(\theta; \tilde{y}) = \mathcal{L}_c(\theta; \tilde{y}) + R_1(\theta; \tilde{y}) + \lambda R_2(\theta; \tilde{y}) \tag{3.8}$$

$\mathcal{L}_c$ and R_1 are applied to the high-confidence samples C based on the entropy H and a threshold ξ in the batch β:

$$C = \{x \in \beta \mid w(x) \geq \xi\} \tag{3.9}$$

$$w = 1 - \frac{H(\tilde{y})}{\log(C)}, \quad H(\tilde{y}) = -\sum_{i=1}^{C} \tilde{y}_i \log \tilde{y}_i, \tag{3.10}$$

where $0 \leq H(\tilde{y} \leq log(C)$ is the entropy of $\tilde{y}$.
For each pair $x_i, x_j \in C$ their silimarity is defined by:

$$W_{i,j} = \begin{cases} 1 & \text{if } argmax_{k \in y}[\tilde{y_i}]_k = argmax_{k \in y}[\tilde{y_j}]_k \\ 0 & otherwise \end{cases} \tag{3.11}$$

It is used in the contrastive regularizer:

$$R_1(\theta; \tilde{y}) = \sum_{(x_i, x_j) \in CxC} l(v_i, v_j, W_{i,j}) \tag{3.12}$$

where $v = BERT(x)$ and l:

$$l = W_{i,j} d_{i,j}^2 + (1 - W_{i,j})[max(0, \gamma - d_{i,j})]^2 \tag{3.13}$$

where d is the distance, γ a fixed margin.
The contrastive regularizer ($R1$) encourages data within the same class to have similar representations and keeps data in different classes separated. For samples from the same class, i.e. $W_{ij} = 1$, it penalizes the distance between them; for samples from different classes, the contrastive loss is large if their distance is small. In this way, the regularizer enforces similar samples to be close while keeping dissimilar samples apart by at least γ.
As shown, the KL loss and the contrastive learning are applied to high-confidence examples. This strategy relies on the assumption that wrongly-labeled samples have

low confidence, which may not always be true. Therefore, they add a confidence-based reweighting.

$$R_2(\theta) = \frac{1}{|C|} \sum_{x \in C} D_{KL}(u||f(x;\theta)) \tag{3.14}$$

where D_{KL} is the KL divergence and $u_i = \frac{1}{C}$ for $i = 1, 2, ..., C$.
$R2$ measures the difference between the probability distribution of the prediction and all probability distributions in set C (samples with high confidence). Such term constitutes a regularization to prevent over-confident predictions and leads to better generalization.

Evaluation

In table 3.2 the evaluation of the model (named COSINE).

Method	AGNews	IMDB	Yelp	MIT-R	TREC	Chemprot	WiC (dev)
ExMatch	52.31	71.28	68.68	34.93	60.80	46.52	58.80
Fully-supervised Result							
RoBERTa-CL$^{\diamond}$ (Liu et al., 2019)	91.41	94.26	97.27	88.51	96.68	79.65	70.53
Baselines							
RoBERTa-WL† (Liu et al., 2019)	82.25	72.60	74.89	70.95	62.25	44.80	59.36
Self-ensemble (Xu et al., 2020)	85.72	86.72	80.08	72.88	66.18	44.62	62.71
FreeLB (Zhu et al., 2020)	85.12	88.04	85.68	73.04	67.33	45.68	63.45
Mixup (Zhang et al., 2018)	85.40	86.92	92.05	73.68	66.83	51.59	64.88
SMART (Jiang et al., 2020)	86.12	86.98	88.58	73.66	68.17	48.26	63.55
Snorkel (Ratner et al., 2020)	62.91	73.22	69.21	20.63	58.60	37.50	–*
WeSTClass (Meng et al., 2018)	82.78	77.40	76.86	–$^{\otimes}$	37.31	–$^{\otimes}$	48.59
ImplyLoss (Awasthi et al., 2020)	68.50	63.85	76.29	74.30	80.20	53.48	54.48
Denoise (Ren et al., 2020)	85.71	82.90	87.53	70.58	69.20	50.56	62.38
UST (Mukherjee and Awadallah, 2020)	86.28	84.56	90.53	74.41	65.52	52.14	63.48
Our COSINE Framework							
Init	84.63	83.58	81.76	72.97	65.67	51.34	63.46
COSINE	**87.52**	**90.54**	**95.97**	**76.61**	**82.59**	**54.36**	**67.71**

$^{\diamond}$: RoBERTa is trained with clean labels. †: RoBERTa is trained with weak labels. *: unfair comparison. $^{\otimes}$: not applicable.

Table 3.2: Performance of Yue Yu et al. model (COSINE) compared with other approaches [19]

3.1.2 Semi-supervised approaches

Another common case is when you have a small set of well-labeled data and a large set of unlabeled data. To deal with this scenario, Haoming Jiang et al. in *Named Entity Recognition with Small Strongly Labeled and Large Weakly Labeled Data* [20]

propose a framework composed by 3 stages (figure 3.3):

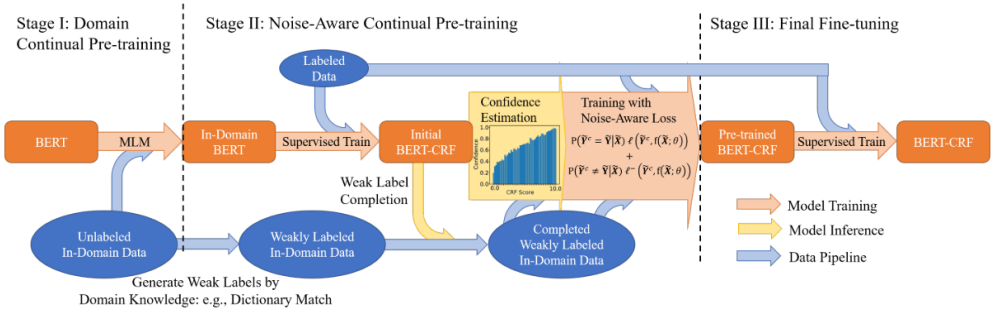

Figure 3.3: Pipeline NEEDLE [20]

Stage 1: In-domain continual pre-training of pre-trained masked language models (trained on open-domain datasets) on the large in-domain unlabeled data.

Stage 2: In this stage, firstly, knowledge bases are used to convert unlabeled data to weakly labeled data, then to find the missing entities: first train an initial NER model with the encoder initialized with the weight obtained in Stage 1 and a Conditional Random Fields randomly initialized on the top; the predicted entities y^p that were missed from the initial weak labels y^w are used to create the new corrected weak labels:

$$y_i^c = \begin{cases} y_i^p & \text{if } i_i^w = O \text{ (non-entity)} \\ y_i^w & \text{otherwise} \end{cases} \tag{3.15}$$

Now, weakly labeled examples and strongly labeled examples are used to train the same model with a noise-aware loss function:

$$min_\theta \frac{1}{M+\widetilde{M}}[\sum_{m=1}^{M} l(Y_m, f(X_m;\theta)) + \sum_{m=1}^{\widetilde{M}} l_{NA}(\widetilde{Y}_m^c, f(\widetilde{X}_m;\theta))] \tag{3.16}$$

where

$$\begin{aligned} l_{NA}(\widetilde{Y}^c, f(\widetilde{X};\theta)) &= \mathbb{E}_{\widetilde{Y}_m=\widetilde{Y}_m^c|\widetilde{X}_m} \mathcal{L}(\widetilde{Y}_m^c, f(\widetilde{X}_m;\theta), \mathbb{1}(\widetilde{Y}_m = \widetilde{Y}_m^c)) = \\ &= \hat{P}(\widetilde{Y}^c = \widetilde{P}|\widetilde{X}) l(\widetilde{Y}^c, f(\widetilde{X};\theta)) + \hat{P}(\widetilde{Y}^c \neq \widetilde{P}|\widetilde{X}) l^-(\widetilde{Y}^c, f(\widetilde{X};\theta)) \end{aligned} \tag{3.17}$$

The model tends to overfit the noise of weak labels using negative log-likelihood (usually used with Conditional Random Fields). The noise-aware loss function makes the fitting to the labels more conservative/aggressive when the confidence is lower/higher. Specifically, when $Y_c = Y$, let the loss function log-likelihood; when $Y_c \neq Y$, let the loss function be the negative log-unlikelihood, which can be viewed as a regularizer, and the confidence of weak labels can be viewed as an adaptive weight.

Stage 3: In the final stage, they use strongly labeled data to perform the fine-tuning of the model obtained in the second stage to get a better fitting to well-labeled examples.

Evaluation

In table 3.5 the evaluation of the model (called NEEDLE) on the datasets described in table 3.4.

Dataset	Number of Samples				Weak Label	
	Train	Dev	Test	Weak	Precision	Recall
E-commerce Query Domain						
En	187K	23K	23K	22M	84.62	49.52
E-commerce Query Domain (Multilingual)						
Mul-En	257K	14K	14K			
Mul-Fr	79K	4K	4K			
Mul-It	52K	3K	3K	17M	84.62	49.52
Mul-De	99K	5K	5K			
Mul-Es	64K	4K	4K			
Biomedical Domain						
BC5CDR Chem	5K	5K	5K	11M	92.08	77.40
BC5CDR Disease	5K	5K	5K	15M	94.46	81.34
NCBI Disease	5K	1K	1K			

Table 3.4: Datasets used for evaluation

Method	BC5CDR chemical	BC5CDR disease	NCBI disease
NEEDLE	**93.74**	**90.69**	**92.28**
w/o NAL	93.60	90.07	92.11
w/o WLC/NAL	93.08	89.83	91.73
w/o FT	82.03	87.86	89.14
w/o FT/NAL	81.75	87.85	88.86
Supervised Baseline			
BioBERT-CRF	92.96	85.23	89.22
Semi-supervised Baseline			
SST	93.06	85.56	89.42
Weakly-supervised Baseline			
WSL	85.41	88.96	78.84
Reported F1-scores in Gu et al. (2020).			
BERT	89.99	79.92	85.87
BioBERT	92.85	84.70	89.13
SciBERT	92.51	84.70	88.25
PubMedBERT	93.33	85.62	87.82
Reported F1-scores in Nooralahzadeh et al. (2019).			
NER-PA-RL†	89.93		-

Table 3.5: Performance of Haoming Jiang et al. model (NEEDLE) compared with other models [20]

An interesting approach based on the Teacher-Student network (pipeline in figure 3.6) was proposed by Giannis Karamanolakis et al. in *Self-Training with Weak Supervision*[21]. Beyond the strongly labeled dataset and the unlabeled data, they use a set of rules.

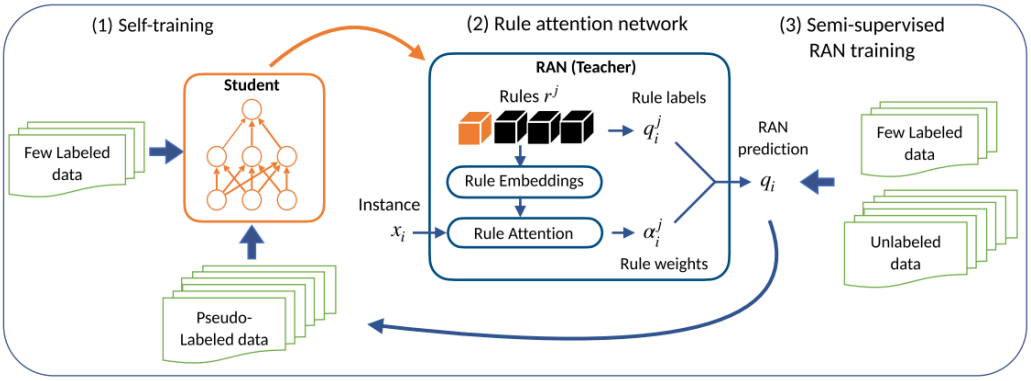

Figure 3.6: Pipeline Student - Teacher training [21]

The student model is trained on the strongly labeled data and computes the predictions on the unlabeled dataset. Then the teacher model aggregate the weak label predicted by all the rules and the student pseudo-lables $p_\theta(y\|x_i)$ to compute a soft label q_i:

$$q_i = \frac{1}{Z_i}\Big(\sum_{j:r^j \in R_i} a_i^j q_i^j + a_i^S p_\theta(y|x_i) + a_i^u u \Big) \tag{3.18}$$

where:

- a^j and a^s are the weight for the rule labels and student labels;
- u is a uniform rule distribution that assigns equal probabilities for all the K classes;
- a^u is the weight assigned to the "uniform rule" for xi, which is computed as a function of the rest of the rule weights $(1 - a^j - a^s)$;

To compute a^j consider the embedding e of the rules:

$$a_i^j = \sigma(f(h_i)^T \cdot e_j) \in [0, 1] \tag{3.19}$$

Where f project h (the embedding of the sentence) to a d-dimension space (attention mechanism), and σ is the sigmoid function.
The loss function is:

$$\mathcal{L}^{RAN} = -\sum_{(x_i,y_i)\in D_L} y_i \log q_i - \sum_{x_i \in D_U} q_i \log q_i \tag{3.20}$$

The first term considers the manually labeled data, the second term minimizes the entropy of the aggregated pseudo-label q_i on unlabeled data (regularization).

Evaluation
In table 3.7 the evaluation of the model (called ASTRA).

	TREC (Acc)	**SMS (F1)**	**YouTube (Acc)**	**CENSUS (Acc)**	**MIT-R (F1)**	**Spouse (F1)**
Majority	60.9 (0.7)	48.4 (1.2)	82.2 (0.9)	80.1 (0.1)	40.9 (0.1)	44.2 (0.6)
LabeledOnly	66.5 (3.7)	93.3 (2.9)	91.0 (0.7)	75.8 (1.7)	74.7 (1.1)	47.9 (0.9)
Snorkel+Labeled	65.3 (4.1)	94.7 (1.2)	93.5 (0.2)	79.1 (1.3)	75.6 (1.3)	49.2 (0.6)
PosteriorReg	67.3 (2.9)	94.1 (2.1)	86.4 (3.4)	79.4 (1.5)	74.7 (1.2)	49.4 (1.1)
L2R	71.7 (1.3)	93.4 (1.1)	92.6 (0.5)	82.4 (0.1)	58.6 (0.4)	49.5 (0.7)
ImplyLoss	75.5 (4.5)	92.2 (2.1)	93.6 (0.5)	80.5 (0.9)	75.7 (1.5)	49.8 (1.7)
Self-train	71.1 (3.9)	95.1 (0.8)	92.5 (3.0)	78.6 (1.0)	72.3 (0.6)	51.4 (0.4)
ASTRA (ours)	**80.3** (2.4)	**95.3** (0.5)	**95.3** (0.8)	**83.1** (0.4)	**76.9** (0.6)	**62.3** (1.1)

Table 3.7: Performance of Giannis Karamanolakis et al. compared to other models[21]

3.1.3 Weak supervision models comparison

Below is a comparison of the weak supervision (supervised and semi-supervised) approaches seen.

	Weak Supervision - Supervised approaches									
Model	**RoSTER**			**COSINE**						
Dataset	CoNLL03	Ontonote v5	Wikigold	AGNews	IMDB	Yelp	MIT-R	TREC	Chemprot	WiC(dev)
F1	85,4	77,1	67,8	87,5	90,5	95,9	76,6	82,5	54,3	67,7

Table 3.8: Supervised models for weak supervision F1 comparison

	Weak supervision - semi-supervised					
Model	**NEEDLE**			**ASTRA**		
Dataset	BC5CDR chemical	BC5CDR disease	NCBI disease	SMS	MIT-R	Spouse
F1	93,7	90,6	92,28	95,3	76,9	62,3

Table 3.9: Semi-supervised models for weak supervision F1 comparison

All models analyzed are tested on different datasets, so it is impossible to make a direct comparison against the same dataset.

3.2 Trigger NER

Trigger NER is an approach that uses triggers to help the model recognize the entities. Where the entity triggers are a group of words that can help explain the recognition process of a particular entity in the same sentence. 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.[22] Therefore, to use this approach, we need, beyond the dataset, a set of triggers associated with the entities. In *TriggerNER: Learning with Entity Triggers as Explanations for Named Entity Recognition* by Bill Yuchen Lin at al. [22], they use as input:

$$D_T = \{(x_i, y_i, T(x_i, y_i))\} \tag{3.21}$$

$T(x, y)$ represents the set of annotated entity triggers, where each trigger $t_i \in T(x, y)$ 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, $t = (w_1, ..., w_i, \rightarrow e)$, where e and w_i are integers in the range of $[1, |x|]$. The training phase is composed of two stages (figure 3.8):

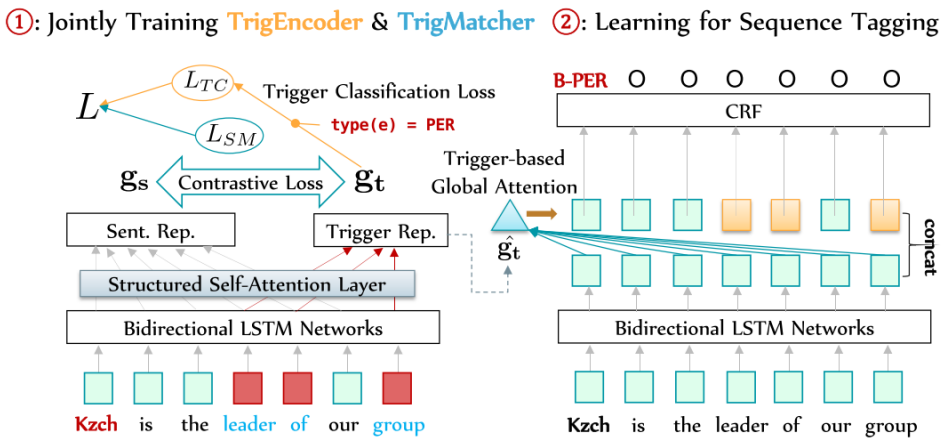

Figure 3.8: Training Trigger NER [22]

Stage 1: TrigEncoder and TrigMatcher The objective of this stage is to obtain the representation of the examples and of the triggers. To do that, it is used a bidirectional LSTM with output: H, the matrix containing the hidden vectors of all of the tokens; After that, the vector representation of the sentences and of the triggers are given in input to their respective attentions layers:

$$\begin{aligned} \overrightarrow{a}_{sent} &= softmax(W_2 tanh(W_1 H^T) \\ g_s &= \overrightarrow{a}_{sent} H \end{aligned} \tag{3.22}$$

where W_2 is a trainable parameters for computing self-attention score vectors.

$$\begin{aligned} \overrightarrow{a}_{trig} &= softmax(W_2 tanh(W_1 Z^T) \\ g_t &= \overrightarrow{a}_{trig} Z \end{aligned} \tag{3.23}$$

where W_2 is a trainable parameters for computing self-attention score vectors. To guide the trigger representation the trigger vector g_t is further fed into a multi-class classifier to predict the type of the associated entity e:

$$\mathcal{L}_{TC} = -\sum \log P(type(e)|g_t; \theta_{TC}) \tag{3.24}$$

While, to learn to match triggers and sentence, a contrastive loss is applied:

$$\mathcal{L}_{SM} = (1 - \mathbb{1}_{matched})\frac{1}{2}(d)^2 + \mathbb{1}_{matched}\frac{1}{1}\{max(0, m - d)\}^2 \tag{3.25}$$

where $d = ||g_s - g_t||_2$.
So, the model use a Joint Loss:

$$\mathcal{L} = \mathcal{L}_{TC} + \lambda\mathcal{L}_{SM} \tag{3.26}$$

Stage 2: Trigger-Enhanced Sequence Tagging The learning objective in this stage is to output the tag sequence y. It is used Bidirectional LSTM (the same used in the *TrigEncoder* and *TrigMatcher*), which output H, the matrix containing the hidden vectors of all of the tokens. Then, use the *TrigMatcher* to compute the mean of all the trigger vectors $\hat{gt}$ associated with the sentence and create a sequence of attention-based token representation H':

$$\begin{aligned} \overrightarrow{\alpha} &= softmax\Big(v^T tanh(U_1 H^T + U_2 \hat{g}_t^T)^T\Big) \\ H\prime &= \overrightarrow{\alpha} H \end{aligned} \tag{3.27}$$

where U_1, U_2, and v are trainable parameters for computing the trigger-enhanced attention scores for each token.
Therefore, concatenate the original token representation H with the trigger-enhanced one H' as the input ($[H; H']$) to the final CRF tagger (learning objective is the same as conventional NER, which is to correctly predict the tag for each token).

Trained the model, to make inferences on new examples (without knowing the triggers, representation in figure 3.9):

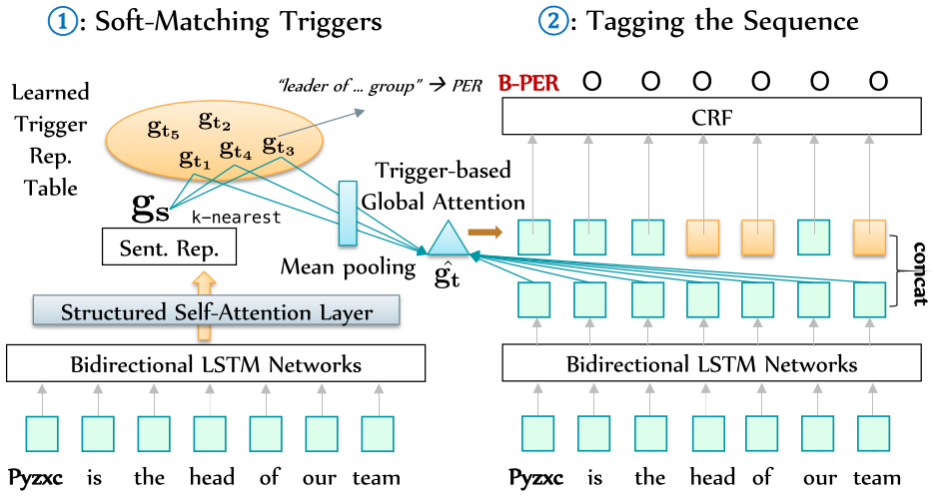

Figure 3.9: Inference Trigger NER [22]

- Use the *TriggerMatcher* (contrastive loss) to compute the similarity between the self-attended sentence representation (g_s) and the trigger representation (g_t);
- Retrieve the most similar triggers in the shared embedding;
- Calculate the mean of the k nearest matched triggers (g_t);
- Use g_t in the SeqTagger (concatenating it to the vector of the sentence).

Evaluation

In figure 3.10 the evaluation of the model (named TMN).

CONLL 2003				
	BLSTM-CRF		TMN	TMN + S.T.
sent.	F1	**trig.**	F1	F1
5%	69.04	3%	75.33	77.68
10%	76.83	5%	80.2	81.57
20%	81.3	7%	82.02	82.43
30%	83.23	10%	83.53	83.53
40%	84.18	13%	84.22	84.33
50%	84.27	15%	85.03	85.38
60%	85.24	17%	85.36	85.52
70%	**86.08**	**20%**	**86.01**	**86.5**

Table 3.10: Performance of Bill Yuchen Lin at al. model (TMN) compared to a BLSTM-CRF [22]

3.3 Prompting

The prompting approach applied to the Named Entity recognition task consists of constructing an input for a Masked Language Model to predict context-dependent hypernyms of the mention, which can be used as type labels (example in table 3.12). In *Ultra-Fine Entity Typing with Weak Supervision from a Maske d Language Model* by Hongliang Dai, Yangqiu Song and Haixun Wang [23] use prompting for a task of ultra-fine entity typing (more than 10.000 types), as Masked Language Model, they use BERT. The work is composed of two steps:

In the first step, they generate the weak labels using Hears Patterns. Hearst patterns (examples in table 3.11) are a set of lexico-syntactic patterns that are easily recognizable, that occur frequently and across text genre boundaries and that indisputably indicate the lexical relation of interest.

Hearst pattern
M and any other H
M and some other H
H such as M
such H as M
H including M
H especially M

Table 3.11: Hearst patterns example

The workflow to labels generation is:

1. Insert into the sentence a few tokens to create an artificial Hearst pattern;
2. Applying the BERT MLM on sentences with artificial Hearst patterns, deriving the top k type labels from the prediction of 'MASK';
3. Singularize the most probable words that are plural;
4. Remove thoe that are non in the vocabulary of the dataset;
5. Use the most probable k words.

Input	**Top Words for [MASK]**
In late 2015, [MASK] **such as** Leonardo DiCaprio starred in the Revenant.	actors, stars, actor, directors, filmmakers
At sum clinics, **they and some other** [MASK] are told the doctors don't know how to deal with AIDS, and to go someplaces else.	patients, people, doctors, kids, children
Finkelstein says he expects the company to "benefit from some of the distruption faced by our competitors **and any other** [MASK]"	company, business, companies, group, investors

Table 3.12: Prompting example [23]

Not all the patterns perform well for all the sentences. For this reason, a set of patterns are tested, and are selected the patterns that reach an F1 score greater than a fixed threshold.

Generated the labels and completed them with labels obtained by headwords,

knowledge base, and human-annotated labels. The training procedure consists of:

1. Pretrainig on automatically labeled samples;
2. Fine-tuning on human-annotated samples;
3. Self-training.

The loss function is computed by subdividing the labels in: general (T_g), fine (T_f) and ultra-fine (T_u).

$$\begin{aligned}\mathcal{J}(x) = \mathcal{L}(x, T_g)\mathbb{1}(\mathfrak{L}, T_g) + \mathcal{L}(x, T_f)\mathbb{1}(\mathfrak{L}, T_f) \\ + \mathcal{L}(x, T_u)\mathbb{1}(\mathfrak{L}, T_u)\end{aligned} \tag{3.28}$$

In the pretraining phase:

$$\mathcal{L}(x, T) = -\sum_{t \in T} \alpha(t)[y_t \cdot \log(p_t) + (1 - y_t) \cdot \log(1 - p_t)] \tag{3.29}$$

It is the binary cross-entropy multiplied for the value $\alpha(t)$ that indicates how confident we are about the label t for x. It is equal to a predefined constant value larger than 1 when t is a positive type for x obtained through entity linking or headword supervision, otherwise, it is equal to 1.
In the fine-tuning phase it is used the binary cross entropy:

$$\mathcal{L}(x, T) = -\sum_{t \in T} [y_t \cdot \log(p_t) + (1 - y_t) \cdot \log(1 - p_t)] \tag{3.30}$$

For the self-training step, it is used the student-teacher approach. Denote:

- h: the ultra-fine entity typing model obtained after pretraining on the automatically labeled data;
- m: the model obtained after fine-tuning h with human-annotated data.

By using m as a teacher model, our self-training step fine-tunes the model h again with a mixture of the samples from the automatically labeled data and the human-annotated data. This time, for the automatically annotated samples, use pseudo

labels generated based on the predictions of m instead of their original weak labels

$$\mathcal{J}_{ST} = \frac{1}{|H|} \sum_{x \in H} \mathcal{J}(x) + \lambda \frac{1}{|A|} \sum_{x \in S} \mathcal{L}_{ST}(x) \tag{3.31}$$

Where A is the automatically annotated samples, and H is the human-annotated samples. λ is a hyperparameter that controls the strength of the supervision from the automatically labeled data.

For automatically labeled data, compute the binary cross-entropy loss with only the types in $Y^+(x)$ and $Y^-(x)$, which are respectively positive or negative strong prediction, i.e. above or below a fixed threshold P or $1 - P$:

$$\mathcal{L}_{ST}(x) = - \sum_{t \in \hat{Y}^+(x)} \log(p_t) - \sum_{t \in \hat{Y}^-(x)} \log(1 - p_t) \tag{3.32}$$

Evaluation

In table 3.13 the evaluation of the model.

Method	Acc	Macro F1	Micro F1
UFET	59.5	76.8	71.8
LTRFET	63.8	82.9	77.3
LDET	64.9	84.5	79.2
DSAM	66.06	83.07	78.19
BERT-Direct	63.25	80.84	75.90
Ours	**67.44**	**85.44**	**80.35**

Table 3.13: Performance of Hongliang Dai et al. model compared to other metods[23]

3.4 Comparison of approaches

The choice of the right approach depends on the data you have to work on, the domain, and the number of labels you have to deal with.
Below I summarize the pros and cons of every approach:

Weak Supervison

Pros:

- Does not require extra data;
- Approaches both with datasets with weak labels and for small well labeled datasets flanked by large unlabeled datasets.

Cons:

- Without well-labeled data could more easily overfit errors;
- Less effective with a large number of classes (fine-grained).

Trigger NER

Pros:

- Less importance to unlabeled entities and errors (thank the use of triggers);
- Triggers could be both rules and natural language.

Cons:

- Require triggers.

Prompting

Pros:

- Potentially does not need much labeled data;

- Effective also for fine-grained classes.

Cons:

- Requires patterns;
- Need to find the best patterns;
- Need to find the position of the entities.

Trigger NER and Prompting approaches have an important limitation: they require rules or textual patterns to be effective; finding such rules or patterns could be an expensive process for companies that often have to deal with weakly labeled datasets precisely to keep labeling costs down. For this reason, I focused on Weak supervision approaches, particularly the one proposed by Yu Meng et al. [2]. In that, It is based on a loss that is easily adaptable both to different models and to the case where we have different levels of label noise in the dataset.
In the next chapter I discuss the limitations of this model and propose solutions to overcome these limitations.

Chapter 4

Research objectives and thesis contributions

The most common scenario in which companies have to deal is when they have only a dataset with weak labels because, as seen in section 1.3, getting well-labeled data is an expensive process, so they decide to use techniques to lower labeling costs, at the expense of label quality. The approach that better fits this scenario is weak supervision. In section 3.1, I presented the state-of-art of this approach, and I decided to explore and extend the one based on Generalized Cross-Entropy (GCE) loss adapted for NER (proposed by Yu Meng et al. [2]). The advantage of this approach, compared to the other, is that it can be easily adapted to different models, it allows us to exploit any information about the amount of noise, and as we shall see, it can also be adapted to the scenario where it is known that there are well-labeled classes and noisier classes.

4.1 Overcoming the limitations of Weighted Generalized Cross Entropy

I introduced and elaborated on how Generalized Cross-Entropy (GCE) loss works in subsection 3.1.1, summarizing:
The GCE loss is a compromise between the Cross-Entropy loss, which has the advantage of reaching convergence quickly, but the disadvantage of giving more

importance to wrong predictions, so it tends to overfit errors; and the Mean Absolute Error Loss, which is a loss robust to noise, but reaches convergence with more difficulty.

$$\mathcal{L}_{GCE} = \sum_{i=1}^{n} \frac{1 - f_{i,y_i}(x;\theta)^q}{q} \tag{4.1}$$

where $0 < q < 1$ is a hyperparameter:

- when $q \to 1$, $\mathcal{L}_{GCE}$ approximates $\mathcal{L}_{MAE}$;
- when $q \to 0$, $\mathcal{L}_{GCE}$ approximates $\mathcal{L}_{CE}$.

Yu Meng et al. [2], extend the GCE implementing a mechanism of noise-removal adding a weight w to the GCE: when the model prediction is greater than a threshold τ (optimal threshold found: 0.7) the weight will be 1, otherwise, 0, and it is updated after several batches.

$$\mathcal{L}_{WGCE} = \sum_{i=1}^{n} w_i \frac{1 - f_{i,y_i}(x;\theta)^q}{q} \tag{4.2}$$

However, this adaptation of the GCE has some limitations:

- The weighted version (WGCE) is based on the intuition that since the loss function is noise-robust, the learned model will be dominated by the correct majority in the distant labels instead of quickly overfitting to noisy labels; if the model prediction disagrees with some given labels, they are potentially wrong and does not consider them in the computation of the loss (giving weight 0).

  My hypothesis is that this assumption is too strong and that it goes and eliminates too many entities (in fact, Yu Meng et al.[2] perform a step of generating new examples through Masked Language Modeling).

  Thus, the **first objective** is to modify such loss in such a way that it is not so punitive with examples that the model gets wrong and thus does not need a phase of generating new examples.

- Yu Meng et al. [2], do not deal with the case where we are aware that some labels are noisier than others, e.g., a common case is one in which several

annotators who are experts in different domains are employed, so they label with more precision the classes related to their domain and with less the others.

The **second objective**, then, is to create a loss function that adapts to the different noise levels of the classes.

4.1.1 Better weighting for Weighted Generalized Cross Entropy

Assigning a weight w to the loss function based on the confidence level of a specific prediction is a very useful technique to make that example weigh less in the training phase of the model, being potentially incorrect.
The intuition is that assigning as weight 0 to examples that have low confidence is overly penalizing: eliminating examples from the loss calculation would be correct only if one was certain of the label error, besides the fact that this could cause the elimination of a large number of examples, thus requiring a subsequent step of generating new examples, which in turn might contain noise. Low predictions can also be caused not only by errors but also by the ambiguity of some classes, e.g., classes such as Location (LOC) and Geopolitical Entities (GPE) appear in similar contexts, are more difficult for the model to learn, and consequently may be predicted with less confidence. Instead, assigning a low weight (such as 0.3) would allow the model not to ignore examples about which it is low in confidence, but at the same time gives it less importance than examples about which it is more confident about prediction.
Therefore, to achieve the **first objective**, my proposal is to moodify the original weight system, going to give a low weight to predictions with low confidence, so that these labels are not completely ignored by the loss function, but at the same time have a lower weight than predictions with high confidence.

$$w_i = \begin{cases} 1 & \text{if } pred > \tau \text{ or } epoch \leq 3 \\ 0.3 & otherwise \end{cases} \tag{4.3}$$

Where $\tau = 0.7$, the weight is equal to 1 for the first three epochs, and then it is updated every epoch.

4.1.2 Adaptive Generalized Cross Entropy loss

The main feature of Generalized Cross Entropy is that through the parameter q, it fluctuates between Cross Entropy (suitable for the case where there are no mislabels in the dataset) and Mean Absolute Error (suitable for noisier datasets but difficult to use because of the difficulty in achieving convergence). This feature makes it easily adaptable to the case where we have labels that we know are more or less noisy than others, going to use q levels of verses depending on the label under consideration and its noise level.

Therefore, to achieve the **second objective**, I propose an *Adaptive Generalized Cross-Entropy* (AGCE). It is based on the intuition that based on the noise level of the label, you can adjust the value of q:

$$\mathcal{L}_{AGCE} = \sum_{i=1}^{n} \frac{1 - f_{i,y_i}(x;\theta)^q}{q} \qquad q = \begin{cases} q_{weak} & \text{if } y_i \in weak\_labels \\ q_{base} & otherwise \end{cases} \tag{4.4}$$

Where $weak\_labels$ is a dictionary $\{class : q_{weak}\}$, q_{weak} is the q value for the respective weak class known and q_{base} is the q value for all the other classes.

A higher q value will be used for noisier classes, and a lower q value will be used for less noisy classes.

Chapter 5

Experimental setup

5.1 Model

Transformer-based models, like BERT, are at the base of the current state-of-art, but as shown in section 2.4, they are very large models, so finetune them requires a lot of computational power, and it is very expensive. Therefore, for computational reasons, I use a lighter version of BERT: DistilBERT [11] with on the top a classification layer.

5.1.1 DistilBERT for token classification

DistilBERT is a distilled version of BERT, it has the same general architecture as BERT. The token-type embeddings and the pooler are removed while the number of layers is reduced by a factor of 2. **DistilBERT base:**

- Transformer layers: 6;
- Hidden size: 768;
- Attention Heads: 12;
- Parameters: 66 M.

DistilBERT's 66 million parameters make it 40% smaller and 60% faster than BERT base, all while retaining more than 95% of BERT's performance.

It has been trained using the knowledge distillation technique [24, 25] (student-teacher learning). BERT was used as a teacher, and the student model was trained to optimize the linear combination of 3 losses:

$$\mathcal{L} = \frac{\mathcal{L}_{mlm} + \mathcal{L}_{distil} + \mathcal{L}_{cosine}}{3} \tag{5.1}$$

Where $\mathcal{L}_{mlm}$ is the Cross Entropy loss of the Masked Language Model task:

$$\mathcal{L}_{mlm} = -\sum_{i=1}^{n} y_i \log(s_i) \tag{5.2}$$

where n is the number of samples, y the truth value of the labels and s the student model prediction.

$\mathcal{L}_{distil}$ is the distillation loss over the soft target probabilities of the teacher, it is the student-teacher Cross Entropy:

$$\mathcal{L}_{distil} = -\sum_{i=1}^{n} t_i \log(s_i) \tag{5.3}$$

where t_i is the output of the teacher model.

$\mathcal{L}_{cosine}$ is the Cosine loss to align the directions of the student and teacher hidden states vectors:

$$\mathcal{L}_{cosine} = 1 - cos(T(x), S(x)) \tag{5.4}$$

where $T(x)$ and $S(x)$ are the hidden state vectors for the input x, respectively of the teacher model and of the student model.

The output distribution of the models was controlled by a softmax-temperature:

$$p_i = \frac{exp(z_i/temp)}{\sum_j exp(z_j/temp)} \tag{5.5}$$

where $temp$ controls the smoothness of the output distribution and z_i is the model score for the class i. The same temperature $temp$ is applied to the student and the

teacher at training time, while at inference, *temp* is set to 1 to recover a standard softmax.

For the Named Entity recognition task, I use DistilbertForTokenClassification implemented by hugginface [26], which consists of a DstilBert Encoder with a token classification head on top (a linear layer on top of the hidden-states output).

5.2 Datasets

I conducted the experiments on two of the most popular and widely used datasets in the literature: CoNLL-2003 and Ontonotes v5.

5.2.1 CoNLL-2003

he Conll2003 was released in 2003 for the CoNLL-2003 shared task: language-independent named entity recognition [27]. It is available in two languages: English and German; I used the English version for the experiments.
The English data was taken from the Reuters Corpus. This corpus consists of Reuters news stories between August 1996 and August 1997. For the training and validation set, ten days worth of data were taken from the files representing the end of August 1996. For the test set, the texts were from December 1996.

There are 4 named entities:

- Person (tag: PERS)
- Location (tag: LOC)
- Organization (tag: ORG)
- Names of Miscellaneous entities (tag: MISC)

The number of samples for training, validation and test set are:

- Train: 14.041
- Validation: 3.250
- Test: 3.453

5.2.2 Ontonotes

OntoNotes v5 [28] is the final release of the OntoNotes project, a collaborative effort between BBN Technologies, the University of Colorado, the University of Pennsylvania, and the University of Southern Californias Information Sciences Institute. The project's goal was to annotate a large corpus comprising various genres of text (news, conversational telephone speech, weblogs, Usenet newsgroups, broadcast, talk shows).
It was released in 2012, and it is available in three languages: English, Chinese, and Arabic. For the experiments, I used the English Language. The dataset was also used for the CoNLL-2012 shared task: Modeling Multilingual Unrestricted Coreference in OntoNotes [29].

There are 18 named entities are:

- Person (tag: PERSON)
- Nationalities or religious or political groups (tag: NORP)
- Facility (tag: FAC)
- Organization (tag: ORG)
- Geopolitical Entities (tag: GPE)
- Location (tag: LOC)
- Product (tag: PRODUCT)
- Date (tag: DATE)
- Time (tag: TIME)
- Percent (tag: PERCENT)
- Money (tag: MONEY)
- Quantity (tag: QUANTITY)
- Ordinal (tag: ORDINAL)

- Cardinal (tag: CARDINAL)
- Event (tag: EVENT)
- Worf of art (tag: WORK_OF_ART)
- Law (tag: LAW)
- Langauge (tag: LANGUAGE)

The number of samples for training, validation and test set are:

- Train: 115.812
- Validation: 15.680
- Test: 12.217

5.2.3 BIO scheme

Both the dataset are used with the BIO tagging format [30]. The BIO or IOB format (short for inside, outside, beginning) is a common tagging format for tagging tokens in a chunking task (ex. named-entity recognition). It consist in:

- The B- prefix before a tag indicates that the tag is the beginning of a chunk;
- The I- prefix before a tag indicates that the tag is inside a chunk;
- The O tag indicates that a token belongs to no entity/chunk.

This format of tagging is essential for contextualized word embeddings.

Example:

It	has	nearly	reached	200	bilion	between	China	and	Japan
O	O	O	O	B-MONEY	I-MONEY	O	B-GPE	O	B-GPE

(Ontonotes V5)

5.2.4 Noise simulation

Linguistic datasets often contain inherent noise due to text ambiguity, especially for the Named Entity Recognition task. Ontonotes dataset, in particular, is known to have a high level of noise.
However, to have more control and simulate different noise levels, I implemented a function to add a given percentage of noise in the dataset. By noise, I mean the presence of mislabels, thus simulating weak labels.

The function takes in input:

- A list of entities to mask;
- A list of entities to use for replacement;
- Percentage of entities to mask.

Example:

- List of entities to mask: all;
- List of entities to use for replacement: ['O'].

Keane	signs	four-year	contract	with	Manchester	United
B-PER	O	O	O	O	B-LOC	I-LOC
			↓ Add noise			
O	O	O	O	O	O	I-LOC

(CoNLL2003)

Chapter 6

Experiments and results

6.1 Experiments objectives

The objectives of the experiments are:

- To verify the performance of GCE, I expect it to perform better than Cross-Entropy, even in the case where the datasets are not injected with noise, this is because of the inherent noise in the datasets;

If this hypothesis is verified, then I proceed with the other two objectives:

- Test the hypothesis underlying the WGCE, namely that giving the loss a low weight if the model has low confidence in a certain prediction will yield better results than the GCE.
- To test the hypothesis underlying AGCE, i.e., that assigning different q values to the classes, based on their noise level, improves performance over WGCE and GCE.

I conducted such experiments on the Ontonotes v5 (Train: 115.812, Validation: 15.680, Test: 12.217, Classes: 18) and CoNLL2003 (Train: 14.041, Validation: 3.250, Test: 3.453, Classes: 4) datasets (analyzed in section 5.2).
For computational reasons, it was not possible to run many experiments on the Ontonotes dataset (having a high number of examples), so I used Ontonotes to test the effectiveness of GCE, while on CoNLL2003 I compared all losses: GCE, WGCE, and AGCE.

6.2 Experiments configurations

I performed the experiments using the following configurations:

Hyperparameters:

- Epochs: 20;
- Train batch size: 256;
- Eval batch size: 256;
- Warmup steps: 1000;
- Early stopping:
  - Patience: 3
  - Monitor: validation loss

I applied Early Stopping to avoid overfitting.

Noise simulation:
To the datasets, I added noise by going to mask the labels with 'O' class labels at different percentages: 10%, 20%, 30%, 50%, and 70%.
Note that the 50% and 70% noise thresholds are extreme cases where random behavior is expected. In the real world, at such noise levels, one must apply techniques to improve data quality before proceeding with model training.

Loss configuration:
The q values tested for GCE and WGCE are: 0 (equivalent to cross entropy), 0.2, 0.5 and 0.7. For AGCE, q_{base} values of 0.1, 0.2, 0.5, and 0.7 were used for low noise percentages (10% and 20%), while the q_{weak} value assigned to the 'O' labels is equal to the q_{base} value increased by a delta of 0.1. As the noise increased, I decreased the values of q_{base} to 0.001, 0.01, 0.1, and 0.2 and increased the delta to 0.15.
I gave higher q values to the 'O' label because as the noise increases, the incorrect 'O' labels increase while the number of the other labels decrease, so the hypothesis is that the probability of errors in the rest of the classes decreases.

Evaluation measures:

The commonly used metric to evaluate a Named Entity Recognition model is the F1 score due to the strong imbalance between positive and negative examples for each class.

$$F1 = 2 \cdot \frac{Precision * Recall}{(Precision + Recall)} \tag{6.1}$$

It is the harmonic mean of precision and recall, where:

$$Precision = \frac{\#True_Positive}{(\#True_Positive + \#False_Positive)} \tag{6.2}$$

It is the ratio between the correctly identified positives (true positives) and all identified positives. The precision metric reveals how many of the predicted entities are correctly labeled.

$$Recall = \frac{\#True_Positive}{(\#True_Positive + \#False_Negatives)} \tag{6.3}$$

It measures the model's ability to predict actual positive classes. It is the ratio between the predicted true positives and what was actually tagged. The recall metric reveals how many of the predicted entities are correct.

The results I report are the mean and standard deviation of the F1 metric obtained from training the models on 3 different seeds. In Appendix A and Appendix B, I report the metrics obtained from each individual seed of both datasets.

6.3 Results and discussion

6.3.1 Ontonotev v5

		AVERAGE GCE		DEV STD GCE	
Noise %	**q**	**Eval F1**	**Test F1**	**Eval F1**	**Test F1**
0	0	83,36	81,10	0,010	0,013
0	0,2	84,17	81,88	0,005	0,003
0	0,5	**84,92**	82,80	0,003	0,002
0	0,7	0,00	0,00	0,000	0,000
10	0	83,70	81,33	0,004	0,005
10	0,2	84,07	81,60	0,004	0,002
10	0,5	**84,27**	81,97	0,002	0,001
10	0,7	0,00	0,00	0,000	0,000
20	0	82,92	80,53	0,003	0,003
20	0,2	**83,44**	81,05	0,002	0,003
20	0,5	81,88	79,49	0,006	0,007
20	0,7	0,00	0,00	0,000	0,000
30	0	81,79	79,02	0,003	0,001
30	0,2	**82,02**	79,38	0,005	0,005
30	0,5	76,49	73,36	0,022	0,021
30	0,7	0,00	0,00	0,000	0,000
30	0	53,12	49,86	0,028	0,020
50	0,2	**58,07**	54,56	0,037	0,040
50	0,5	50,36	48,16	0,031	0,033
50	0,7	0,00	0,00	0,000	0,000
70	0	6,33	5,47	0,088	0,077
70	0,2	5,71	5,01	0,086	0,069
70	0,5	**20,59**	19,67	0,044	0,031
70	0,7	0,00	0,00	0,000	0,000

Table 6.1: Ontonotes v5 Generalized Cross Entrpèy (GCE) evaluation

Table 6.1 reports the results obtained on the Ontonotes dataset, where the Generalized Cross Entropy (GCE) improves performance in terms of F1 score compared to the values achieved by the Cross Entropy ($q = 0$).
One might have expected a correlation between q value and noise percentage. But this was not the case, for example, in the case of no noise, the optimal q value was found to be 0.5, while in the case of 50% noise, the optimal q was 0.2. This is probably due to the inherent noise within the dataset (known to be high in the Ontonotes dataset) caused by other annotation errors or entity ambiguity.
Furthermore, as can be seen from the increase in the Standard Deviation value, the hypothesis of increased randomness among the different seeds at higher noise levels (50% and 70%) was confirmed.

6.3.2 CoNLL2003

		AVERAGE GCE		DEV STD GCE		AVERAGE WGCE		DEV STD WGCE				AVERAGE AGCE		DEV STD AGCE	
Noise %	q	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	q_base	delta	Eval F1	Test F1	Eval F1	Test F1
0	0	92,64	88,33	0,005	0,005										
0	0,2	93,08	88,98	0,005	0,005	93,00	88,81	0,005	0,005						
0	0,5	**93,52**	89,14	0,004	0,004	93,27	89,10	0,005	0,005						
0	0,7	93,34	88,85	0,000	0,001	**93,47**	89,05	0,002	0,003						
10	0	92,43	88,02	0,006	0,008					0,1	0,1	92,68	88,60	0,004	0,004
10	0,2	92,63	88,58	0,006	0,004	92,47	88,25	0,005	0,002	0,2	0,1	92,89	88,76	0,002	0,004
10	0,5	**92,99**	88,57	0,002	0,001	**93,15**	88,75	0,003	0,003	0,5	0,1	**93,05**	88,55	0,007	0,005
10	0,7	92,86	88,47	0,000	0,001	92,76	88,24	0,002	0,003	0,7	0,1	0,00	0,00	0,000	0,000
20	0	91,73	87,63	0,002	0,004					0,1	0,1	**92,17**	88,14	0,005	0,002
20	0,2	91,86	87,90	0,004	0,003	91,90	87,88	0,004	0,002	0,2	0,1	92,10	88,05	0,007	0,007
20	0,5	**91,88**	87,87	0,006	0,003	**92,04**	88,00	0,006	0,006	0,5	0,1	92,01	87,96	0,002	0,002
20	0,7	81,67	79,43	0,057	0,070	80,98	78,68	0,065	0,082	0,7	0,1	0,00	0,00	0,000	0,000
30	0	90,15	86,35	0,005	0,007					0,001	0,15	**90,93**	86,67	0,010	0,006
30	0,2	**90,67**	86,41	0,009	0,008	**90,68**	86,47	0,008	0,008	0,01	0,15	90,39	86,33	0,011	0,011
30	0,5	88,71	84,48	0,006	0,006	88,12	84,21	0,020	0,028	0,1	0,15	90,27	85,99	0,014	0,012
30	0,7	11,16	10,04	0,193	0,174	0,00	0,00	0,000	0,000	0,2	0,15	90,59	86,30	0,008	0,006
50	0	**72,98**	69,57	0,058	0,049					0,001	0,15	**72,65**	69,39	0,047	0,031
50	0,2	72,07	68,44	0,045	0,029	**73,61**	69,77	0,049	0,033	0,01	0,15	72,54	69,20	0,048	0,031
50	0,5	67,05	63,53	0,028	0,042	65,42	62,63	0,032	0,034	0,1	0,15	71,22	67,47	0,039	0,021
50	0,7	0,00	0,00	0,000	0,000	0,00	0,00	0,000	0,000	0,2	0,15	69,95	65,23	0,003	0,010
70	0	23,86	20,57	0,136	0,121					0,001	0,15	25,90	23,07	0,165	0,152
70	0,2	**28,78**	26,39	0,137	0,120	**28,34**	26,16	0,137	0,118	0,01	0,15	25,93	23,22	0,166	0,152
70	0,5	0,00	0,00	0,000	0,000	0,00	0,00	0,000	0,000	0,1	0,15	36,94	33,43	0,048	0,051
70	0,7	0,00	0,00	0,000	0,000	0,00	0,00	0,000	0,000	0,2	0,15	**38,23**	34,62	0,100	0,101

Table 6.2: CoNLL2003 Generazlized Cross Entropy (GCE), Weighted Generalized Cross Entropy (WGCE) and Adaptive Generalized Cross Entropy (AGCE) evaluation

In bold is the maximum F1 value obtained on the validation set, relative to the loss for each noise threshold, in green the absolute maximum for the respective noise threshold. In the AGCE, *delta* is the difference between q_{base} and q_{weak}, e.g. if we have $q_{base} = 0.2$ and $delta = 0.1$, then $q_{weak} = 0.3$.

Table 6.2 reports the results obtained on the CoNLL2003 dataset, where the Generalized Cross Entropy (GCE) improves performance in terms of F1 obtained from Cross Entropy ($q = 0$).
The addition of the weight (WGCE) still improved the F1 values, thus increasing the gap with Cross Entropy, except for the 50% and 70% noise thresholds, in these cases, as expected, and as shown by the higher values of Standard Deviation between the different seeds, the models start to have random performance.
When it comes to the Adaptive Generalized Cross Entropy (GCE), this always improves over the GCE, but it turns out to be equivalent to the WGCE in the case of low noise (10%), while it still improves in the cases of 20% and 30% noise, again

the considerations made earlier about the 50% and 70% noise thresholds apply.

As in Ontonotes also in CoNLL2003, a close correlation between noise percentage and q value did not emerge, this did not allow to find a general optimum q value, but it has to be determined from time to time according to the dataset.
In the case of AGCE on CoNLL2003, was confirmed the hypothesis that as the percentage of noise increases, it is necessary to increase the difference between q_{weak} (value of q for the class known to be noisy) and q_{base} (value of q for all other classes).

6.3.3 Results analysis

The results obtained from the implemented loss functions always improve the predictions made by the models based on Generalized Cross Entropy Loss and Cross Entropy Loss. In particular with the latter, we note that in the CoNLL2003 dataset, the model trained with a 20% injected noise rate based on the Adaptive Generalized Cross Entropy loss performed similar to the Cross Entropy loss based model trained on the dataset without added noise. The same is true for the Weighted Generalized Cross Entropy loss based model with 10% injected noise, which performed even better than the version based on Cross Entropy without noise innjection. Below is an example where you can see how in some cases the model with 20% noise trained with AGCE makes better predictions than the model trained on the clean dataset based on Cross Entropy.

Example:

- Dataset: CoNLL2003, true labels:

Our	statistics	are	the	highest	for	everything	,	Hakme	said	.
O	O	O	O	O	O	O	O	B-PER	O	O

- Prediction model loss: AGCE $q_{base} : 0.1$, $q_{weak} : 0.2$;
  Dataset: ConLL2003
  Noise: 20%

Our	statistics	are	the	highest	for	everything	,	Hakme	said	.
O	O	O	O	O	O	O	O	B-PER	O	O

- Prediction model loss: GCE $q : 0$ (Cross Entropy);
  Dataset: ConLL2003
  Noise: 20%

Our	statistics	are	the	highest	for	everything	,	Hakme	said	.
O	O	O	O	O	O	O	O	B-PER	B-PER	B-PER

- Prediction model loss: GCE $q : 0$ (Cross Entropy);
  Dataset: ConLL2003
  Noise: 0%

Our	statistics	are	the	highest	for	everything	,	Hakme	said	.
O	O	O	O	O	O	O	O	B-PER	B-PER	B-PER

In the example, the AGCE manages to correctly classify the 'O' labels in the final part of the sentence despite it being trained with numerous wrong examples for that label, unlike models trained with Cross Entropy with or without noise.
Instead an example where with 20% noise the Cross Entropy based model correctly labels all tokens unlike the AGCE based model is:

Example:

- Dataset: CoNLL2003, true labels:

USDA	gross	cutout	hide	and	offal	value	.
B-ORG	O	O	O	O	O	O	O

- Prediction model loss: AGCE $q_{base} : 0.1$, $q_{weak} : 0.2$;

Dataset: ConLL2003
Noise: 20%

USDA	gross	cutout	hide	and	offal	value	.
B-MICS	O	O	O	O	O	O	O

- Prediction model loss: GCE $q : 0$ (Cross Entropy);
  Dataset: ConLL2003
  Noise: 20%

USDA	gross	cutout	hide	and	offal	value	.
B-ORG	O	O	O	O	O	O	O

- Prediction model loss: GCE $q : 0$ (Cross Entropy);
  Dataset: ConLL2003
  Noise: 0%

USDA	gross	cutout	hide	and	offal	value	.
B-ORG	O	O	O	O	O	O	O

The au AGCE based model correctly classifies the 'O' labels (which are the noisy ones) and correctly identifies the position of the principlae entity within the sentence, but misclassifies its classification (B-MICS instead of B-ORG); this may be because the noise injection removed a significant number of B-ORG labels,making them more difficult to recognize with few examples at a noise-robust loss, unlike Cross Entropy which tends to converge, thus learn, more quickly.

In addition, we note that in both datasets all noise-robust loss functions (Generalized Cross and Weighted Generalized Cross Entropy) performed better than Cross Entropy even in the case where no noise was injected, this is because of the inherent noise present in the datasets for the Named Entity Recognition task.
As for the WGCE this has always performed better than the basic GCE, so I can say that between the two it is the best to use in any case. Instead, in the case where we know that there are labels within the dataset that are more or less noisy than others,

and we know their noise level, using AGCE allows us to improve the results obtained by the model over WGCE in the case of high noise rates. However, its use should be examined on a case-by-case basis as finding the best combination of q values may require several experiments, thus needing greater computational capabilities.

Chapter 7

Conclusion

To evaluate the results obtained, these were compared with the results achieved by Cross entropy (loss generally used for this task), a direct comparison with the work proposed by Yu Meng et al.[2] could not be made, as they use a different model and a different noise injection system (not specified).
First, the basic Generalized Cross Entropy loss was evaluated, which was confirmed to be a good noise-robust loss, performing better than the Cross Entropy loss in terms of F1 score. The results obtained from the proposed new weighting system showed a further improvement over the Generalized Cross Entropy (thus going to increase the gap with the Cross Entropy). This meant that the first goal of obtaining a loss that reinforces the results of the basic Generalized Cross Entropy and does not require an additional step of generating new examples was achieved, thus simplifying the training phase compared to the model proposed by Yu Meng et al. [2] and allowing the use of this loss on any model.
Regarding the Adaptive Generalized Cross Entropy, this did not show improvement over the Weighted Generalized Cross Entropy in the case of low noise, while it improved the performance of the Weighted Generalized Cross Entropy in the presence of higher noise percentages.
However, there are still some limitations: the optimal q parameter depends on the dataset and the amount of noise, so it is not possible to identify a parameter that performs well in all cases. This means that several trainings are required to find the optimal q parameter, this limitation is accentuated in the Adaptive Generalized

Cross Entropy version, where we have two parameters to optimize.
In conclusion, the use of this loss depends on the computational capabilities one has available: it improves the performance obtained by Cross Entropy, but several training steps must be performed to find the optimal q parameters and achieve significant improvements.

7.1 Future work

As reported in the conclusions, the main limitation of this loss is that it requires training several models to find the optimal q value for the dataset one has available. Thus, the most immediate future development is to extend this loss so that it adapts dynamically during training to the data's noise level. A first idea is to start with an intermediate q (0.5) and increase or decrease it over the epochs according to the confidence of the predictions on the labels: if the model classifies a certain label with high average confidence, then that class is likely to be low in noise, so the value of q can be lowered; on the other hand, if the model classifies a label with low average confidence predictions, then the value of q will be decreased as that class is likely to be noisier.
Further future development is to test this loss in a hybrid context: by flanking the model with rules or patterns or by combining it with the seen approaches of Trigger NER or Prompting.
Finally, it would be interesting to test the performance of this loss on other models, as it is easy to implement and does not require other self-training steps.

Bibliography

[1] Eric Burgener. Meeting the new unstructured storage requirements for digitally transforming enterprises, 2021.

[2] Yu Meng, Yunyi Zhang, Jiaxin Huang, Xuan Wang, Yu Zhang, Heng Ji, and Jiawei Han. Distantly-supervised named entity recognition with noise-robust learning and language model augmented self-training. *arXiv preprint arXiv:2109.05003*, 2021.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[8] Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[9] Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.

[10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[11] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[12] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Automated concatenation of embeddings for structured prediction. *arXiv preprint arXiv:2010.05006*, 2020.

[13] Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li. Dice loss for data-imbalanced nlp tasks. *arXiv preprint arXiv:1911.02855*, 2019.

[14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[16] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[17] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[18] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.

[19] Yue Yu, Simiao Zuo, Haoming Jiang, Wendi Ren, Tuo Zhao, and Chao Zhang. Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. *arXiv preprint arXiv:2010.07835*, 2020.

[20] Haoming Jiang, Danqing Zhang, Tianyu Cao, Bing Yin, and Tuo Zhao. Named entity recognition with small strongly labeled and large weakly labeled data. *arXiv preprint arXiv:2106.08977*, 2021.

[21] Giannis Karamanolakis, Subhabrata Mukherjee, Guoqing Zheng, and Ahmed Hassan Awadallah. Self-training with weak supervision. *arXiv preprint arXiv:2104.05514*, 2021.

[22] Bill Yuchen Lin, Dong-Ho Lee, Ming Shen, Ryan Moreno, Xiao Huang, Prashant Shiralkar, and Xiang Ren. Triggerner: Learning with entity triggers as explanations for named entity recognition. *arXiv preprint arXiv:2004.07493*, 2020.

[23] Hongliang Dai, Yangqiu Song, and Haixun Wang. Ultra-fine entity typing with weak supervision from a masked language model. *arXiv preprint arXiv:2106.04098*, 2021.

[24] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. pages 535–541, 2006.

[25] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[26] Distilbert hugginface. URL https://huggingface.co/docs/transformers/model_doc/distilbert.

[27] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[28] Mitchell Marcus Eduard Hovy Sameer Pradhan Lance Ramshaw Nianwen Xue Ann Taylor Jeff Kaufman Michelle Franchini Mohammed El-Bachouti Robert Belvin Ann Houston Ralph Weischedel, Martha Palmer. Ontonotes release 5.0. URL https://catalog.ldc.upenn.edu/docs/LDC2013T19/OntoNotes-Release-5.0.pdf.

[29] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint Conference on EMNLP and CoNLL-Shared Task*, pages 1–40, 2012.

[30] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer, 1999.

Appendix A

Ontonotes v5 evaluation

Dataset: Ontonotes v5

- Train: 115.812
- Validation: 15.680
- Test: 12.217
- Classes: 18

Model: DistilBERTForTokenClassification
Hyperparameters:

- Epochs: 20;
- Train batch size: 256;
- Eval batch size: 256;
- Warmup steps: 1000;
- Early stopping:
  - Patience: 3
  - Monitor: validation loss

Seeds: 6, 42, 100

		Ontonotes v5 - Generalized Cross Entropy									
		SEED 6		SEED 42		SEED 100		AVERAGE		DEV STD	
Noise %	q	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1
0	0	0,832089552	0,809292069	0,824529158	0,798936978	0,844258689	0,824858757	0,8336258	0,811029268	0,009954077	0,013047914
0	0,2	0,836437635	0,816138021	0,844027026	0,817424532	0,844665233	0,822735036	0,841709965	0,818765863	0,004577109	0,003497074
0	0,5	**0,851600864**	0,827401575	**0,846507308**	0,826799827	**0,849609756**	0,829920275	**0,849239309**	0,828040559	0,002566905	0,001655453
0	0,7	0	0	0	0	0	0	0	0	0	0
10	0	0,832746778	0,808004327	0,841651499	0,817837321	0,836649366	0,814108456	0,837015881	0,813316701	0,00446366	0,004964081
10	0,2	0,837359413	0,813617553	0,844397084	0,818167726	0,840259079	0,81635342	0,840671859	0,816046233	0,003536947	0,002290588
10	0,5	**0,841117852**	0,820416764	**0,844459688**	0,820363979	**0,842491023**	0,818235642	**0,842689521**	0,819672128	0,001679737	0,001244313
10	0,7	0	0	0	0	0	0	0	0	0	0
20	0	0,826407309	0,802398965	0,829509931	0,80439175	0,831585381	0,809074608	0,82916754	0,805288441	0,00260596	0,003426966
20	0,2	**0,835303192**	0,809535199	**0,832341583**	0,808402193	**0,835581104**	0,813550008	**0,834408626**	0,8104958	0,001795497	0,002705007
20	0,5	0,818463848	0,798586289	0,824929207	0,798916507	0,813047614	0,787254586	0,818813556	0,794919127	0,005948511	0,00663974
20	0,7	0	0	0	0	0	0	0	0	0	0
30	0	0,820459342	0,79031423	0,818892573	0,791458177	0,814405678	0,788705278	0,817919198	0,790159229	0,003142022	0,00138298
30	0,2	**0,82521157**	0,798914584	**0,819368633**	0,792099042	**0,816027031**	0,790334332	**0,820202411**	0,793782653	0,004648691	0,004531125
30	0,5	0,754908403	0,721191489	0,749388399	0,721700905	0,790298976	0,757819104	0,764865259	0,733570499	0,022198492	0,021001452
30	0,7	0	0	0	0	0	0	0	0	0	0
50	0	0,517019374	0,490212351	0,513249651	0,484285785	**0,563245121**	0,52122086	0,531171382	0,498572999	0,02784055	0,019836211
50	0,2	**0,623103653**	0,591763405	**0,559233092**	0,522790519	0,559643935	0,522277722	**0,580660227**	0,545610549	0,036757659	0,039970368
50	0,5	0,503920171	0,479848683	0,534605643	0,515055315	0,472270228	0,449754332	0,503598681	0,481552777	0,031168951	0,032683827
50	0,7	0	0	0	0	0	0	0	0	0	0
70	0	0,164844551	0,143573318	0,004973426	0,005805743	0,020160151	0,014646426	0,063326043	0,054675163	0,088244914	0,077114856
70	0,2	0,156028984	0,129492299	0,003240059	0,003476612	0,01193621	0,0174268	0,057068418	0,050131903	0,085812593	0,069081157
70	0,5	**0,220430108**	0,203934945	**0,240407117**	0,223550216	**0,156761052**	0,162532769	**0,205866092**	0,196672644	0,043683509	0,031150249
70	0,7	0	0	0	0	0	0	0	0	0	0

Appendix B

CoNLL2003 evaluation

Dataset: CoNLL2003

- Train: 14.041
- Validation: 3.250
- Test: 3.453
- Classes: 4

Model: DistilBERTForTokenClassification
Hyperparameters:

- Epochs: 20;
- Train batch size: 256;
- Eval batch size: 256;
- Warmup steps: 1000;
- Early stopping:
  - Patience: 3
  - Monitor: validation loss

Seeds: 6, 42, 100

		CoNLL2003 - Generalized Cross Entropy									
		SEED 6		SEED 42		SEED 100		AVERAGE		STD DEV	
Noise %	q	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1
0	0	0,927998664	0,887681794	0,929862739	0,884107408	0,921299759	0,878231352	0,926387054	0,883340185	0,004503234	0,004771707
0	0,2	0,924916388	0,889336721	0,932364729	0,884581805	0,935136952	0,895377558	0,930806023	0,889765362	0,005285561	0,005410626
0	0,5	0,930256065	0,888594399	**0,93907051**	0,896223924	**0,93617736**	0,889473223	**0,935167978**	0,891430515	0,004493078	0,004174405
0	0,7	**0,933747586**	0,890199877	0,933087681	0,887547038	0,933221336	0,887776607	0,933352201	0,88850784	0,000348874	0,001469835
10	0	0,924413263	0,885461397	**0,93063342**	0,884083348	0,91776616	0,870956522	0,924270948	0,880167089	0,00643481	0,008006289
10	0,2	0,919745862	0,884278711	0,929169457	0,882826163	0,930103922	0,8902364	0,926339747	0,885780425	0,005729555	0,003926737
10	0,5	0,928169369	0,88608261	0,930543933	0,884107408	**0,931135962**	0,88687544	**0,929949755**	0,885688486	0,001570017	0,001425482
10	0,7	**0,92849322**	0,884791814	0,928300615	0,885966461	0,928895613	0,883437748	0,928563149	0,884732008	0,0003036	0,001265417
20	0	**0,917215746**	0,880106809	0,919789724	0,87177196	0,914829153	0,87696659	0,917278208	0,876281786	0,002480875	0,004209412
20	0,2	0,91439133	0,877319497	0,918489737	0,87723036	**0,922985782**	0,882447092	0,918622283	0,878998983	0,004298759	0,002986483
20	0,5	0,911511883	0,874878758	**0,92286988**	0,88040802	0,922156698	0,880960791	**0,918846153**	0,87874919	0,006361667	0,003363267
20	0,7	0,847978192	0,832955404	0,851137714	0,836394784	0,75092735	0,713598188	0,816681085	0,794316126	0,056966314	0,069924934
30	0	0,897374292	0,858565737	0,899785684	0,860761784	0,907372723	0,871237308	0,901510899	0,86352161	0,005217704	0,006771607
30	0,2	**0,900984168**	0,862078273	**0,902571158**	0,857403355	**0,916503865**	0,87270447	**0,906686397**	0,864062033	0,008539124	0,007841078
30	0,5	0,888313034	0,840756226	0,880679847	0,841797054	0,892326305	0,851851852	0,887106396	0,844801711	0,005916247	0,00612774
30	0,7	0,334748993	0,301292308	0	0	0	0	0,111582998	0,100430769	0,193267421	0,173951195
50	0	0,763393277	0,719496353	0,662507311	0,639859053	**0,763458401**	0,727886563	**0,72978633**	0,695747323	0,058265348	0,048582126
50	0,2	**0,767752872**	0,715487878	**0,678677528**	0,659150924	0,71569805	0,678708157	0,720709483	0,684448987	0,044748633	0,028603862
50	0,5	0,641296961	0,590660757	0,673598639	0,6414418	0,696679797	0,673748402	0,670525133	0,635283653	0,027819049	0,041884738
50	0,7	0	0	0	0	0	0	0	0	0	0
70	0	0,081614078	0,065808764	0,309302326	0,267670431	0,324880559	0,283485389	0,238598987	0,205654861	0,136175867	0,121368144
70	0,2	**0,137609494**	0,135455774	**0,319889085**	0,283909128	**0,405887807**	0,372463588	**0,287795462**	0,26394283	0,136988371	0,119758781
70	0,5	0	0	0	0	0	0	0	0	0	0
70	0,7	0	0	0	0	0	0	0	0	0	0

		CoNLL2003 - Weighted Generalized Cross Entropy									
		SEED 6		SEED 42		SEED 100		AVERAGE		DEV STD	
Noise %	q	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1
0	0,2	0,92458194	0,889200561	0,931233772	0,882687429	0,934239222	0,892429577	0,930018311	0,888105856	0,004942042	0,004962474
0	0,5	0,926469355	0,885044349	**0,934921167**	0,895703365	0,936577181	0,892199825	0,932655901	0,890982513	0,005421311	0,005432775
0	0,7	**0,932459677**	0,887832365	0,934811828	0,893137169	**0,936817342**	0,890608512	**0,934696282**	0,890526015	0,002181129	0,002653364
10	0,2	0,920234114	0,883403361	0,929492547	0,883602973	0,924390857	0,880539499	0,924705839	0,882515278	0,004637247	0,001713983
10	0,5	**0,92788704**	0,888732765	**0,932872905**	0,883585256	**0,933670459**	0,890088496	**0,931476801**	0,887468839	0,003134296	0,003430913
10	0,7	0,927616646	0,878710473	0,925707547	0,883696229	0,929340511	0,884676424	0,927554901	0,882361042	0,001817269	0,003199248
20	0,2	**0,915188703**	0,877594091	0,919179123	0,877983849	0,922777307	0,880967197	0,919048378	0,878848379	0,003795991	0,00184527
20	0,5	0,913417722	0,874006008	**0,922481924**	0,87943761	**0,925204486**	0,886532344	**0,920368044**	0,879991987	0,006171169	0,006281542
20	0,7	0,846558925	0,834845049	0,848084066	0,833380428	0,734782609	0,69207604	0,809808533	0,786767172	0,064978831	0,082008196
30	0,2	**0,900410678**	0,862667506	**0,903763533**	0,858263408	**0,916305548**	0,873050027	**0,906826586**	0,864660313	0,008378452	0,007592068
30	0,5	0,865218529	0,814290963	0,875314098	0,841278539	0,903098784	0,870617998	0,88121047	0,8420625	0,019616418	0,0281717
30	0,7	0	0	0	0	0	0	0	0	0	0
50	0,2	**0,765112099**	0,711039614	0,680076628	0,65962632	**0,763165306**	0,722427388	**0,736118011**	0,697697774	0,048543021	0,033458886
50	0,5	0,627564808	0,594450941	**0,689799014**	0,66221336	0,64530408	0,622383253	0,654222634	0,626349185	0,03206134	0,03405485
50	0,7	0	0	0	0	0	0	0	0	0	0
70	0,2	**0,133845935**	0,13453997	**0,312744727**	0,281735839	**0,403580146**	0,368650942	**0,283390269**	0,26164225	0,137242119	0,118341882
70	0,5	0	0	0	0	0	0	0	0	0	0
70	0,7	0	0	0	0	0	0	0	0	0	0

			CoNLL2003 - Adaptive Generalized Cross Entropy (delta 0,1)									
			SEED 6		SEED 42		SEED 100		AVERAGE		STD DEV	
Noise %	q	delta	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1
10	0,1	0,1	0,921894579	0,883541867	0,929139128	0,883468835	0,929391071	0,891035209	0,926808259	0,886015304	0,004257237	0,004347519
10	0,2	0,1	**0,928739101**	0,88692301	0,927147881	0,883777875	0,930806005	0,892212934	0,928897663	0,88763794	0,001834209	0,004262734
10	0,5	0,1	0,922587131	0,879192274	**0,934784443**	0,888300688	**0,934258397**	0,888888889	**0,930543324**	0,885460617	0,006895283	0,005436505
10	0,7	0,1	0	0	0	0	0	0	0	0	0	0
20	0,1	0,1	0,916519546	0,878675823	0,924740265	0,88270837	0,923948904	0,88294849	**0,921736239**	0,881444228	0,004535083	0,002400513
20	0,2	0,1	0,912627872	0,87202381	**0,925052809**	0,883993605	**0,925287356**	0,885400035	0,920989346	0,880472483	0,007242198	0,007350481
20	0,5	0,1	**0,92237597**	0,878827994	0,920163071	0,881881347	0,917783962	0,878190973	0,920107668	0,879633438	0,002296506	0,00197263
20	0,7	0,1	0	0	0	0	0	0	0	0	0	0
30	0,1	0,1	**0,901917152**	0,862699199	0,875300998	0,826785228	**0,915513289**	0,872948982	0,897577146	0,85414447	0,020454433	0,024241712
30	0,2	0,1	0,901668806	0,863341736	**0,888736146**	0,850027367	0,914067148	0,871022676	**0,9014907**	0,861463926	0,01266644	0,01062287
30	0,5	0,1	0,842069025	0,809429191	0,869580268	0,842403214	0,87964571	0,849785408	0,863765001	0,833872604	0,019451602	0,021488009
30	0,7	0,1	0	0	0	0	0	0	0	0	0	0
50	0,1	0,1	**0,770605947**	0,717789569	0,678605631	0,660520717	**0,718337182**	0,679874461	**0,722516253**	0,686061582	0,046142312	0,029131438
50	0,2	0,1	0,682492582	0,630288167	**0,690753912**	0,672893442	0,703543971	0,660539461	0,692263488	0,65457369	0,010606572	0,0219202
50	0,5	0,1	0,585589338	0,540205729	0	0	0	0	0,195196446	0,180068576	0,338090162	0,311887923
50	0,7	0,1	0	0	0	0	0	0	0	0	0	0
70	0,1	0,1	0,139252995	0,135054022	**0,317264008**	0,283245579	0,402413553	0,36990908	0,286310185	0,262736227	0,134283186	0,118763209
70	0,2	0,1	**0,417091246**	0,395193591	0,291521935	0,251355484	**0,441531798**	0,3965496	**0,38338166**	0,347699558	0,080485974	0,08343917
70	0,5	0,1	0	0	0	0	0	0	0	0	0	0
70	0,7	0,1	0	0	0	0	0	0	0	0	0	0

			ConLL2003 - Adaptive Generalized Cross Entropy (delta 0,15)									
			SEED 6		SEED 42		SEED 100		AVERAGE		STD DEV	
Noise %	q	delta	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1	Eval F1	Test F1
30	0,001	0,15	0,898414059	0,862501127	**0,912352992**	0,863607451	**0,917035775**	0,873967684	**0,909267609**	0,866692087	0,00968668	0,006325086
30	0,01	0,15	0,897473233	0,861111111	0,897842968	0,853977845	0,916411304	0,874775422	0,903909169	0,863288126	0,010828745	0,010568318
30	0,1	0,15	**0,902269807**	0,861566157	0,888927009	0,846769791	0,916942957	0,871381196	0,902713258	0,859905714	0,014013237	0,012389436
30	0,2	0,15	0,897976607	0,859836581	0,904798695	0,85875193	0,914824889	0,87047191	0,905866731	0,86302014	0,008474767	0,006476169
50	0,001	0,15	0,773568944	0,726308416	0,679093005	0,665720081	**0,726806484**	0,689588	**0,726489478**	0,693872166	0,047238767	0,03052052
50	0,01	0,15	**0,774205678**	0,726327367	0,679020643	0,664706479	0,723028855	0,685033216	0,725418392	0,692022354	0,047637486	0,031399355
50	0,1	0,15	0,753039514	0,698707593	0,674800653	0,658338412	0,708834969	0,667127345	0,712225045	0,67472445	0,039229444	0,021229807
50	0,2	0,15	0,70277386	0,647453217	**0,69878391**	0,664125965	0,696844478	0,645401489	0,699467416	0,652326891	0,003023207	0,010269665
70	0,001	0,15	0,071880937	0,062816616	**0,320740365**	0,270736381	0,384479718	0,358671029	0,259033673	0,230741342	0,165182585	0,151928144
70	0,01	0,15	0,071689136	0,064527027	0,320070958	0,272205521	0,386067785	0,359974827	0,259275959	0,232235792	0,165772457	0,151725196
70	0,1	0,15	0,372042492	0,343666284	0,32004013	0,27907605	0,416233091	0,38018175	0,369438571	0,334308028	0,048149317	0,051198373
70	0,2	0,15	**0,427078495**	0,403665241	0,267328025	0,229573085	**0,452611219**	0,405249601	**0,382339246**	0,346162642	0,100417457	0,100972626