**Doctoral Dissertation**

# Structured Representation Learning for Structured Prediction

Yiran Wang

Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Professor Taro Watanabe
Natural Language Processing Lab. (Division of Information Science)

Submitted on December 15, 2023

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of Engineering

Yiran Wang

Dissertation Committee:

| | |
|---|---|
| Professor Taro Watanabe | (Supervisor) |
| Professor Satoshi Nakamura | (Co-supervisor) |
| Associate Professor Hiroyuki Shindo | (Co-supervisor) |
| Doctor Yuji Matsumoto | (RIKEN AIP) |

# Structured Representation Learning for Structured Prediction[1]

Yiran Wang

## Abstract

Structured prediction tasks, e.g., part-of-speech tagging, (nested) named entity recognition, and constituency parsing, are considered to be fundamental and essential techniques of natural language processing. In recent years, emerging deep-learning models, especially pre-trained language models, have provided fantastic ways to obtain informative representation, and have been continuously refreshing the leaderboards of these tasks. However, existing models typically employ universally applicable representation learning techniques, without considering the unique characteristics inherent to each specific task. Furthermore, lack of interpretability also keeps them a black box to humans, and the inability to explain their decision-making mechanism hindered researchers from further improving them.

In this dissertation, I mainly focus on leveraging the task-specified characteristics of these structured prediction tasks to learn structured and interpretable representations for solving these issues.

First of all, I factorize representation according to the hierarchical structures of the nested named entity recognition task. A carefully designed algorithm is introduced to explicitly exclude the harmful influence from the best path of

[1]Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, December 15, 2023.

previous level. By additionally introducing the chunk selection strategies and switching the encoding to be the innermost first scheme, I obtained level-wise representation and also pushed the performance to better results.

Moreover, I also factorize the representation for the conventional structured prediction tasks. With the proposed contrastive hashing methods, narrowing the representation bottleneck to be only 24 bits (almost) without sacrificing performance becomes possible. These learned discrete bits are demonstrated having properly preserved the necessary features of the downstream tasks, therefore, can provide researchers with a more interpretable tool to analyze the internal mechanism of the black-box neural networks.

The main contribution of this dissertation is that I proposed two representation learning methods to learn structured representation for structured prediction tasks. Numerous experiments and discussions are provided to show the effectiveness and efficiency of my methods.

**Keywords:**
Representation Learning, Structured Prediction, Nested Named Entity Recognition, Contrastive Learning, Hashing, Interpretability

# Acknowledgements

First and foremost, I want to express my heartfelt gratitude to Professor Taro Watanabe. His invaluable advice and detailed suggestions on my research have been instrumental in helping me clarify my thoughts and identify hidden challenges in my ongoing projects. His unwavering encouragement, particularly during times when my experiments faced setbacks, has been a source of strength for me. It's hard to imagine how I could have completed this dissertation without his continuous support and guidance.

I would like to extend my sincere thanks to Professor Yuji Matsumoto. He welcomed me into his research lab years ago, providing an environment where I could receive solid and comprehensive doctoral training. His patience and meticulous attention to detail have been crucial in helping me navigate the complexities of my ongoing research projects and in refining my academic papers. Learning from him has given me a glimpse into the qualities that a true researcher should possess, and for that, I am truly honored to be one of his students.

I'm also very thankful to Associate Professor Hiroyuki Shindo. From the very beginning of my doctoral course, he has provided consistent guidance and mentorship. His teachings have equipped me with a variety of experimental skills and valuable knowledge that have been beneficial to my academic journey.

In addition, I'd like to extend my gratitude to Dr. Masao Utiyama for offering me the opportunity to work at NICT. His abundant encouragement and academic support have been invaluable. Furthermore, working at NICT has given me the chance to connect with a wide range of skilled researchers, adding great value to my academic journey.

Besides, I want to thank Professor Satoshi Nakamura for his constructive comments and insightful suggestions that have enriched the quality of this dissertation.

As a member of both the Matsumoto and Watanabe laboratories, I've had an enriching and challenging study abroad life. The feedbacks and intellectual discussions from all lab members have greatly enhanced my academic experience. A special mention must go to our lab secretary, Ms. Yuko Kitagawa, who has been providing remarkable helps in both research and daily life, particularly for international students like me.

Finally, big thanks to my parents and friends. They've always supported me and respected my choices, helping me stay strong on this journey.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Structured prediction tasks, such as part-of-speech tagging, nested named entity recognition, and constituency parsing, are considered fundamental and essential techniques in natural language processing. Generally speaking, these tasks aim to detect linguistic structures within given sentences. For example, part-of-speech tagging assigns a syntactic label to each token. Named entity recognition identifies the boundaries of existing entities in sentences and assigns a semantic label to each one. Nested named entity recognition allows for entities to be nested within other entities. Constituency parsing identifies the grammatical constituents of a sentence.

In recent years, emerging deep learning models, particularly pre-trained language models, have provided innovative methods for obtaining informative representations and have continually refreshed the leaderboards for these tasks. They offer a universal solution for natural language processing, which involves feeding the input sentences into a universal encoder to transform the original discrete symbolic tokens into informative high-dimensional continuous representations. These vectors are then used for downstream tasks. However, these models commonly do not take the task-specific characteristics of structured prediction tasks into consideration. As a result, they fail to leverage structural information for structured prediction, leading to performance loss. Moreover, hidden states from these models are typically unstructured continuous high-dimensional vectors consisting of only human-unreadable floating-point numbers. This issue makes the

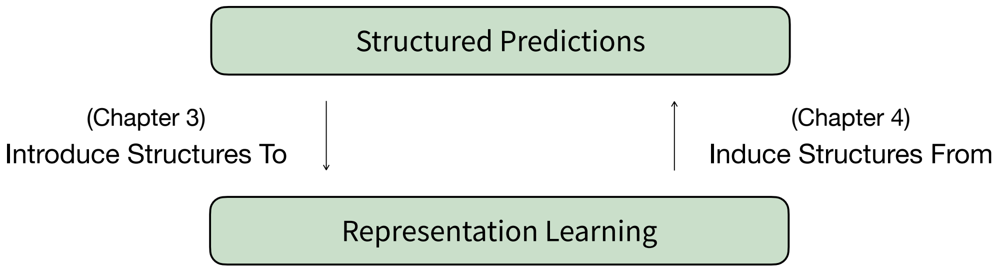


Figure 1.1: The overview of this dissertation.

internal mechanism *black boxes* to humans, and the inability to explain their decision-making mechanisms in ahead hinders researchers from further advancing them.

In this dissertation, I claim that incorporating structures into representation learning is beneficial for various structured prediction tasks. On the one hand, explicitly introducing target structures helps the model leverage the structural information to avoid suboptimal solutions. On the other hand, implicitly inducing target structures from downstream structured prediction tasks aids in reorganizing the task-specific features in a more interpretable manner, unveils the internal decision-making mechanisms, and offers potential inspiration for future researchers to design more efficient representation learning methods.

Firstly, in Wang et al. [101], I analyze the representation learning issues in existing methods and identify a problem where the best path exerts a harmful influence on the decoding of subsequent levels. Therefore, I propose a carefully designed algorithm to explicitly exclude the influence of the best path from previous levels. By introducing chunk selection strategies and altering the encoding scheme to prioritize innermost entities, the model is designed to learn *level-wise representations* to adapt to the hierarchical structures of nested named entity recognition, consequently achieving better performance.

Moreover, in Wang et al. [100], I focus on compressing and interpreting the continuous representations from pre-trained language models. Using the proposed contrastive hashing methods, it becomes feasible to narrow and discretize the representation bottleneck to only 24 bits without significantly compromising performance. These induced *bit-wise representations* preserve all the essential features required for downstream tasks. Compared with token-level tags from previous work, my method not only indicates whether the syntactic properties of two given tokens are different but also distinguishes exactly which bits they differ in. In this sense, my method can be considered as implicitly inducing target structures and representing them in a 24-bit format, which is equivalent but differs from the original structure format. In this way, my model provides researchers with a more interpretable tool for analyzing the internal mechanisms of black-box neural networks.

## 1.2 Contribution

The contributions are of this dissertation are the followings,

1. For nested named entity recognition tasks, I analyze how information from other levels may affect the entity detection on current level and propose a hierarchical representation learning method to explicitly exclude the influence of the best path. Besides, I propose an encoding scheme and chunk selection strategies. This method focuses on hierarchical structures and thus learns level-wise representations.

2. I provide experiments on nested named entity recognition to demonstrate that my methods successfully exclude the influence from previous levels and obtain effective level-wise representations.

3. For the issue of interpretability, I propose novel contrastive hashing methods to compress and discretize the continuous vectors from pre-trained langauge models down to only 24 bits. In addition, I introduce an instance selection strategy and transformer-based ego-attention to further ensure the task-relevant information. This method focuses on interpretability and thus learns bit-wise representations.

4. I conduct extensive experiments on various structured prediction tasks to demonstrate that my model completely preserves task-relevant information and obtains informative and compact bit-wise representations.

## 1.3 Organization

This dissertation organizes as follows.

**Chapter 1** introduces the motivation and contributions of this dissertation.

**Chapter 2** brings fundamental concepts and related work. On the one hand, I introduce various structured prediction tasks, such as part-of-speech tagging, (nested) named entity recognition, and constituency parsing. On the other hand, I describe the workflow of pre-trained language models, the interpretability issues, and the existing attempts on resolving them.

**Chapter 3** aims to enhance the performance of nested named entity recognition by explicitly incorporating its hierarchical structures to learn *level-wise* representation learning. It begins by explaining how does the best path influence the subsequent levels on decoding. After that, I propose a novel method to explicitly exclude the influence of the best path, along with chunk selection strategies and an innermost-first encoding scheme. I conduct numerous experiments to demonstrate the efficiency and effectiveness of my model. Finally, I also display the chunk distribution to intuitively illustrate the mechanism of level-wise representation learning.

**Chapter 4** focuses on improving the interpretability of conventional structured prediction tasks by implicitly inducing targets structures and compressing them as a novel *bit-wise* representation learning. In this chapter, I first introduce the existing attempts on compressing and interpreting continuous vectors from pre-trained language models, and then proposes a novel contrastive hashing method, by applying it along with instance selection and an ego-attention mechanism, my model becomes capable of hashing the continuous vectors to discrete codes. Experiments and discussions shows that using these binary as the sole inputs, performance on various tasks are still kept, and the outputs cases and bit distribution reveal more details about what information is preserved by these bits exactly.

**Chapter 5** gives a summary of this dissertation and discusses the limitations and potential future work.

# Chapter 2

# Background

## 2.1 Structured Prediction

Structured prediction tasks aim at detecting structures from plain sentences. The most commonly known tasks include part-of-speech tagging, named entity recognition, and constituency parsing. The training goal is generally minimizing the negative log-likelihood of the target structure of the given input sentence,

$$\mathcal{L} = -\log \frac{s\left(\boldsymbol{y} \mid \boldsymbol{x}\right)}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} s\left(\boldsymbol{y}' \mid \boldsymbol{x}\right)} \tag{2.1}$$

and inference aims at searching the most probable structure in the space of valid structures.

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}' \in \mathcal{Y}}{\arg\max}\, s\left(\boldsymbol{y}' \mid \boldsymbol{x}\right) \tag{2.2}$$

Where $\boldsymbol{x}$, $\boldsymbol{y}$, and $\mathcal{Y}$ are the input sentence, target structure, and the search space consist of all valid target structures, respectively.

Generally, enumerating all valid structures in the entire search space to compute the partition function, i.e., the denominator part of Equation 2.1, is intractable due to the exponential size. Therefore, structured prediction models commonly rely on probabilistic structured distributions, such as Conditional Random Field [42] (CRF) and Cocke–Kasami–Younger Algorithm [33] (CKY), for efficient training and inference. These probabilistic structured distributions build upon the sum-product algorithm [41] to significantly reduce time complexity by factorizing the probabilistic graph into a product of local functions, thus resulting in remarkable acceleration.

Additionally, Eisner [20] revealed the fact that the inside-outside algorithm [6] and the forward-backward algorithm [8], which are widely used in structured prediction for inference, are conceptually equivalent to the back-propagation algorithm [76]. This makes implementing probabilistic structured distributions much easier and also makes them more efficient.

### 2.1.1 Part-of-speech Tagging

Part-of-speech tagging aims to classify the syntactic categories of each token in given sentences. Before the era of deep learning, traditional statistics-based models [72, 62, 52] commonly followed the paradigm of feeding hand-crafted features into probabilistic structured distributions and then performing sequential labeling. However, these hand-crafted features are highly dependent on experience, thus limiting performance and application. Early-stage deep learning methods noticed that morphological features [30, 73, 10] are beneficial to part-of-speech tagging and can simultaneously address the out-of-vocabulary issue. During this stage, the standard architecture [54, 43] was commonly employing a bidirectional Long Short-term Memory [27] (LSTM) followed by CRF. After that, contextual information was demonstrated can remarkably enhance accuracy, and these work [2, 66] finally lead to the new paradigm of pre-trained language models [18, 49, 45].

### 2.1.2 Named Entity Recognition

Named entity recognition extends the task to span-level by simultaneously detecting the boundaries of named entities and assigning semantic labels to them. In the field of information extraction, named entities are often noun phrases that refer to real-world objects, such as persons, organizations, products, etc. Detecting these entities is considered the first step in linking them with real-world knowledge.

Existing named entity recognition models commonly utilize architecture similar to that used in part-of-speech tagging, therefore, both can be categorized as sequential labeling tasks. They have also followed similar developmental trajectories, such as employing CNNs and LSTMs to construct word representations from the character level, and then shifting to using pre-trained language models.

### 2.1.3 Nested Named Entity Recognition

However, these conventional models are not designed for nested named entity recognition, therefore, they are inherently incapable of detecting entities nested within other entities. Fortunately, a large number of recent papers across various types have proposed methods to address the case of nested entities. I introduce them in the following paragraphs.

**Layered Model** Models in this category employ an encoder to update the token representation from level to level. For example, Ju et al. [32] dynamically updates span-level representations for the next layer of recognition based on recognized inner entities. Fisher and Vlachos [22] proposed a *merge and label* method to further enhance this idea.

**Region-based Model** Lin et al. [47] proposed an anchor-region network to recognize nested entities by first detecting anchor words and entity boundaries, and then classifying each detected span.

**Hypergraph-based Model** Lu and Roth [51] proposed a hypergraph structure in which edges are connected to multiple nodes to represent nested entities. Muis and Lu [58] and Wang and Lu [95] addressed the spurious structures and ambiguity issues of the hypergraph structure.

**Parsing-based Model** Finkel and Manning [21] indicated that all these nested entities are located in some non-terminal nodes of the constituency parses of the original sentences. Therefore, they proposed using a CRF constituency parser to obtain them. However, its cubic time complexity limits its applicability.

### 2.1.4 Constituency Parsing

Constituency parsing, also known as phrase parsing, aims to extract constituency-based parses, which are designed to reflect the syntactic structures of the input sentences. Input tokens make up the terminal nodes of the parses, while other intermediate nodes consist of non-terminal nodes. To address this task, previous research has proposed a variety of solutions, which can primarily be categorized into methods represented by Cross and Huang [17], known as transition-based methods, and methods represented by Stern et al. [80], known as graph-based methods. Transition-based methods build phrases by iteratively shifting tokens from a buffer to a stack and reducing partially built parses to form larger parses. In contrast, graph-based methods formalize this task as dynamic programming decoding. The architectures of graph-based methods [91, 39, 83, 107] are generally similar to the sequential labeling models, but have the CRF decoder replaced with the CKY decoder. In this dissertation, I mainly focus on the graph-based method.

## 2.2 Interpretability

Deep learning methods, especially the recently popular pre-trained language models, have significantly improved the performance of various tasks, they have become the de facto new paradigm. The new paradigm involves introducing a neural encoder with strong capabilities to extract information from discrete input sentences and then produce differentiable continuous high-dimensional vectors for downstream inference. These vectors are supposed to contain as much as possible task-relevant information that is beneficial for task prediction, therefore, these kinds of methods are also commonly known as representation learning methods. Although the conventional approach is to randomly initialize the encoder and train it along with the task decoder, the recently popular pre-trained language models claim that pre-training the encoder with a huge amount of raw text corpus and then fine-tuning them on the downstream task leads to much better performance.

However, it is still difficult to explain what actually happens within these models and what preserved information leads in such extraordinary improvements. Being called *black boxes* is not only due to the fact that the continuous vectors of these models are incomprehensible to human, but also because the internal mechanism by which they make decisions is also still completely unknown. These characteristics make further improving their performance extremely difficult.

Therefore, improving the interpretability of neural networks has become an important and urgent task for researchers, and there is already much existing work in this line. Early work focused on probing syntactic and semantic properties from the continuous vectors of pre-trained models. Subsequent papers attempted clustering and hashing tricks to obtain meaningful discrete outputs. After that, some recently published papers have also tried directly compressing continuous vectors into interpretable discrete tokens.

### 2.2.1 Probing

Tenney et al. [85, 84] may be considered as the most effective approaches at the early stage of neural network interpretability field. They proposed a suite of probing tasks to measure what kind of linguistic information is captured by pre-trained language models. According to their results, they found that traditional pipelines are rediscovered by pre-trained language models. Such that vectors from low-level layers generally preserve morphological and syntactic information that helps in part-of-speech tagging, while higher-level information focuses more on semantic and task-relevant features.

Later work further found that syntactic trees can be recovered from continuous vectors. On the one hand, Vilares et al. [92], Kim et al. [36], and Bai et al. [5] successfully induced parses by simply calculating the syntactic distances among token representations according to their different distance definitions. Jawahar et al. [31] and Htut et al. [29], on the other hand, pointed out that syntactic structure is not directly encoded in self-attention weights. These papers provide some insight into how pre-trained language models encode syntactic information. However, all of them rely on the assumption that the desired features are preserved in these continuous vectors in a human-readable format and hypothesize that some simply defined measurements can effectively reveal the syntactic features. In practice, it is completely possible that information is stored in some hard-to-define format, such as nonlinear subspaces.

On the semantic level, pre-trained language models do not perform as well as they do on syntactic tasks. They fail at capturing numbers [94] and struggle with understanding named entities [7], fortunately, these issues can be significantly improved with sufficient fine-tuning. Therefore, although they are generally considered capable of extracting syntactic-level information, their proficiency at the semantic level is deficient.

### 2.2.2 Clustering and Hashing

Unlike the aforementioned indirect probing approaches, providing a direct geometric perspective for explanation is always much more intuitive. However, high dimension makes visualization become extremely complicated. Although some useful tools such as t-SNE [56] and UMAP [56] have been proposed, they often fail on capturing the desired linguistic features from such high-dimensional spaces and then commonly result in hard-to-understand token distributions in projected 3-dimensional spaces.

Reif et al. [74], Wiedemann et al. [102], and Pasini et al. [61] proposed employing clustering approaches on the continuous representations of pre-trained language models. Their results show that the representation space is divided into multiple subspaces, where different syntactic and semantic features are distributed. Work in this line also provides some insights on the famous $king - man + woman = queen$ equation [57]. However, they generally only offer coarse-grained qualitative explanations rather than fine-grained quantitative ones.

Different from clustering, hashing is a far more straightforward method that discards and quantizes the continuous representations of pre-trained language models, replacing them with discrete labels instead. These discrete labels are often called codebook and are associated with an additional trainable embedding weight. These methods assume that vectors often contain subtle and irrelevant information that should be eliminated. However, the discrete nature of labels naturally prevents them from being applied to end-to-end deep learning frameworks, making them difficult to train and hard to retain desired information. Recently, Ou et al. [60] and Qiu et al. [69] proposed solving these issues by maximizing mutual information and achieved significant improvements in performance on document retrieval tasks. Xue and Aletras [104], on the other hand, replaced the input embedding layer with a hash function and trainable codebook, thereby extending the transformer to be vocabulary-independent.

### 2.2.3 Compressing

In recent, Li and Eisner [46] propose to maximize the mutual information between discrete tokens and the targets, while at the same time minimizing the mutual information between discrete tokens and inputs. This method is generally known as the information bottleneck [86], and Li and Eisner [46] extended it by introducing a stochastic variational inference method to estimate mutual information efficiently and applied it to obtaining discrete tags.

Kitaev et al. [40] on the other hand directly quantize the hidden states as discrete tags and then use them in downstream training to achieve satisfactory performance. However, these obtained discrete tags are distinct and independent of each other. They only indicate whether two vectors contain different information, but cannot quantify their differences or pinpoint exactly where they differ.

# Chapter 3

# Learning Level-wise Representation

## 3.1 Introduction

Named Entity Eecognition (NER), as a key technique in natural language processing, aims to detect entities and assign semantic category labels to them. Early research [30, 54, 43] proposed employing deep learning methods and achieved significant performance improvements. However, in many specialized domains, particularly in scientific fields like biomedical science [99], entities are allowed to nest within other entities, which is known as Nested Named Entity Recognition (nested NER). Figure 3.1 illustrates an example of the nested NER task. Correctly and efficiently detecting all existing nested entities is one of the most important prerequisites for various downstream tasks, such as relation extraction and entity linking. Therefore, nested NER plays a critical role in natural language processing.

However, existing methods primarily focus on flat named entity recognition, where entity nesting does not exist, therefore, they cannot satisfactorily handle nested entities. Recently, a large number of papers have proposed novel methods [22, 98] for the nested NER task. Among them, layered approaches solve this task through multi-level sequential labeling. In these approaches, entities are divided into several levels, where the term *level* indicates the depth of entity nesting, and sequential labeling is performed repeatedly. As a special case of the layered method, Shibuya and Hovy [78] observed that short entities are always

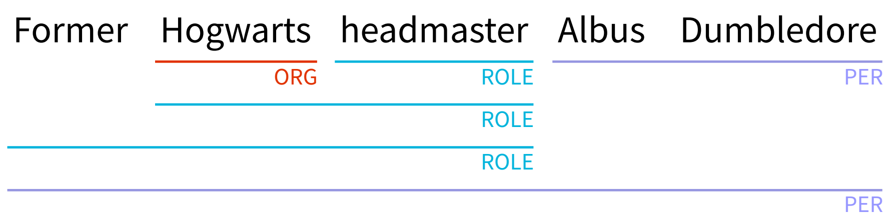


Figure 3.1: An example of the nested NER task. The underlines of different colors show the boundaries of entities, while `ORG`, `ROLE`, and `PER` indicate the organization, role and person entity categories, respectively.

nested within longer entities. Once the outer entity is detected, inner entities are considered to exist within the span of the detected outer entity. Thus, they search for inner entities only within these spans, rather than throughout the entire sentence. Additionally, they found that searching for the best path within the span of the detected entity results only in identifying the known outer entity itself, as it is located along the best path. Therefore, they instead detect inner entities by searching for the second-best path, applying a conventional Conditional Random Field [42] (CRF) with the same potential function.

We argue that inner entities are unlikely to be located on the second-best path. Because Shibuya and Hovy [78] continue to use the same potential function, any path in the remaining search space that shares the many labels with the best path will obtain the high emission scores. Therefore, searching for entities along the second-best path is likely to yield a path with many overlaps with the best path, making it unlikely to contain the expected inner entities. For this reason, we claim that even though Shibuya and Hovy [78] have the best path excluded from the search space, the influence of the best path is not excluded. Thus, forcing inner entities to be located on the second-best path of the current level, as done by Shibuya and Hovy [78], leads to contradictory optimization goals and ultimately harms performance.

In this research, we employ a different potential function at each level to address this issue. We aim to achieve this by introducing an encoder that generates multiple hidden states at each time step. At each level, we select some of these hidden states for entity recognition and then remove those hidden states that interact with the labels of the best path before proceeding to the next level. In this manner, the emission scores for the best path labels differ from level to level, allowing us to explicitly exclude the influence of the best path. Furthermore, we also propose three different selection strategies to leverage the information among hidden states.

Moreover, Shibuya and Hovy [78] proposed recognizing entities from outermost to inner. We empirically demonstrate that extracting the innermost entities first yields better performance. This may be because some long entities do not contain any inner entities, so using an outermost-first encoding mixes these entities with other existing shorter entities at the same levels, thereby causing the encoder representations to become dislocated. In this research, we convert entities to the commonly used `IOBES` encoding scheme [71], and address nested NER by applying CRF level by level.

Our contributions are considered as the follows,

1. We design a novel nested NER algorithm that explicitly excludes the influence of the best path by using a different potential function at each level.

2. We propose three different selection strategies to fully utilize the information among hidden states.

3. We empirically demonstrate that recognizing entities from the innermost to the outer layers results in better performance.

4. We provide extensive experimental results to demonstrate the effectiveness and efficiency of our proposed method on standard benchmark datasets across diverse domains, namely ACE2004, ACE2005, GENIA, and NNE.

## 3.2 Baselines

As a special case of the layered method, Shibuya and Hovy [78] detect entities from level to level. They first search the best path by applying a CRF to detect the outermost entities. After obtaining these longer entities in the given sentence, they narrow their search space by searching only on the spans of the detected entities. This is because shorter entities can only be nested within longer entities. From the second level onwards, they only focus on the second-best path, as they demonstrate that searching the best path within the span of a detected entity only yields that known outer entity. They continue to detect inner entities using the same potential function until no more specific inner entities can be found. In addition, they also designed a special algorithm to efficiently search for the second-best path by excluding the known best path.

On the other hand, Wang et al. [98] proposed another layered method. They enumerate all $l$-gram spans by iteratively merging neighboring spans using a convolutional neural network. At each level, they first predict the entity category of each span according to its representation. Then, they merge the representations of neighboring $l$-gram spans to construct the representations of all $(l+1)$-gram spans for the next level. In this way, the method reduces the sentence length level by level, arranged in a pyramid shape, which accounts for the name.

## 3.3 Proposed Methods

Formally speaking, named entity recognition task aims at recognizing entities in a given sequence, i.e., $x_1, \ldots, x_n$. For nested NER, entities may be nested within longer entities, whereas for flat named entity recognition, no such nesting occurs. Existing algorithms address flat named entity recognition by applying a sequential labeling method, which assigns each token a label, i.e., $y_1, \ldots, y_n$ where $y_t \in \mathcal{Y}$, to simultaneously determine the span and category of each entity and non-entity. To tackle nested NER, our method builds upon previous layered methods and extends the sequential labeling approach with a multi-level encoding scheme. In this encoding scheme, we divide entities into various levels according to their nesting depths and apply the sequential labeling method level by level to detect all entities.

### 3.3.1 Encoding Schemes

Different from part-of-speech tagging, which is only a token-level task, named entity recognition further detects the boundaries of entities. Therefore, sequential labeling also needs to extend the label set through an encoding scheme to give them the ability to recognize boundaries. Currently, one of the most commonly used encoding scheme is `BIOES` [71], which contains several different labels. `O` indicates the current token is not an entity, `S-` means this is a single token entity, while `B-`, `I-`, and `E-` are used for multi-token entities and stand for the beginning, intermediate, and end of an entity, respectively.

Shibuya and Hovy [78] suggested identifying the outermost entities first and then recursively detecting the nested inner entities. However, our findings indicate that starting with the innermost entities yields better performance. We use the sentence illustrated in Figure 3.1, as an example to illustrate the details of these two encoding schemes. The outermost-first encoding scheme are presented below.

| | Former | Hogwarts | headmaster | Albus | Dumbledore |
|---|---|---|---|---|---|
| (level 1) | `B-PER` | `I-PER` | `I-PER` | `I-PER` | `E-PER` |
| (level 2) | `B-ROLE` | `I-ROLE` | `E-ROLE` | `B-PER` | `E-PER` |
| (level 3) | `O` | `B-ROLE` | `E-ROLE` | `O` | `O` |
| (level 4) | `O` | `S-ORG` | `S-ROLE` | `O` | `O` |
| (level 5) | `O` | `O` | `O` | `O` | `O` |
| (level 6) | `O` | `O` | `O` | `O` | `O` |

In this example, the outermost entity *Former Hogwarts headmaster Albus Dumbledore* appears at the first level, while the innermost entities *Hogwarts* and *headmaster* appear at the fourth level. Besides, we keep all examples to have the same number of levels by filling the remaining levels, which contain no entities, with the label `O`. For the ACE2004, ACE2005, and NNE datasets, the maximal depth of entity nesting is 6, thus, examples in these datasets have 6 levels. For the GENIA dataset, the maximal level is only 4.

| | Former | Hogwarts | headmaster | Albus | Dumbledore |
|---|---|---|---|---|---|
| (level 1) | `O` | `S-ORG` | `S-ROLE` | `B-PER` | `E-PER` |
| (level 2) | `O` | `B-ROLE` | `E-ROLE` | `O` | `O` |
| (level 3) | `B-ROLE` | `I-ROLE` | `E-ROLE` | `O` | `O` |
| (level 4) | `B-PER` | `I-PER` | `I-PER` | `I-PER` | `E-PER` |
| (level 5) | `O` | `O` | `O` | `O` | `O` |
| (level 6) | `O` | `O` | `O` | `O` | `O` |

However, we find that detecting from the innermost entities results in better performance as follows. In this encoding scheme, the innermost entities *Hogwarts*, *headmaster*, and *Albus Dumbledore* appear at the first level. Note that the innermost-first encoding scheme is not the simple reverse of the outermost-first encoding scheme. For example, the entity *Former Hogwarts headmaster* and the entity *Albus Dumbledore* appear at the same level in the outermost-first scheme, but they appear at different levels in the innermost-first scheme.

## 3.3.2 Influence of the Best Path

Although the second-best path searching algorithm is proposed as the main contribution of Shibuya and Hovy [78], we claim that forcing inner entities to be located on the second-best path of the current level is not optimal. To clarify the issue, consider the target paths at levels 3 and 4 from the innermost-first encoding example above. The best path at level 3 is `B-ROLE`, `I-ROLE`, `E-ROLE`, `O`, `O`. Therefore, the path most likely to be the second-best is one that shares many labels with the best path, rather than those that have no overlap with the best path at all. For example, the path `B-ROLE`, `I-ROLE`, `E-ROLE`, `O`, `S-ORG` is more likely to be selected than the actual target label sequence at level 4, i.e., `B-PER`, `I-PER`, `I-PER`, `I-PER`, `E-PER`. In addition, Shibuya and Hovy [78] reuse the same potential function at all higher levels. This indicates that, for instance, at level 3 and time step 1, the target label is `B-ROLE`, therefore, their model encourages the emission scores of label `B-ROLE` to be larger than the emission score of the non-target label `B-PER`,

$$\boldsymbol{h}_1^\top \boldsymbol{v}_{\texttt{B-ROLE}} > \boldsymbol{h}_1^\top \boldsymbol{v}_{\texttt{B-PER}} \tag{3.1}$$

while at level 4, the target at the first time step turns to be `B-PER`, their model naturally shifts to optimizing the opposite objective.

$$\boldsymbol{h}_1^\top \boldsymbol{v}_{\texttt{B-ROLE}} < \boldsymbol{h}_1^\top \boldsymbol{v}_{\texttt{B-PER}} \tag{3.2}$$

These two optimization goals are contradictory and eventually hurt performance. Therefore, we claim that although they exclude the best path from the search space, the influence of the best path is not excluded.

To separate the influence between different levels, the crux of the matter is to introduce different emission scores for different levels and to ensure that each emission score is used and only used at one level. For example, by introducing level-wise hidden states, the optimization goals are no longer contradictory.

$$\boldsymbol{h}_1^{3\top} \boldsymbol{v}_{\texttt{B-ROLE}} > \boldsymbol{h}_1^{3\top} \boldsymbol{v}_{\texttt{B-PER}} \tag{3.3}$$

$$\boldsymbol{h}_1^{4\top} \boldsymbol{v}_{\texttt{B-ROLE}} < \boldsymbol{h}_1^{4\top} \boldsymbol{v}_{\texttt{B-PER}} \tag{3.4}$$

This is because $\boldsymbol{h}_1^3$ and $\boldsymbol{h}_1^4$ are two distinctive hidden states to be used only at levels 3 and 4, respectively.

To achieve this goal, we introduce a novel encoder (§3.3.4) to produce $m$ hidden states at each time step, such as $\boldsymbol{h}_t^1, \ldots, \boldsymbol{h}_t^m$, where $m$ is the number of levels. We use it to replace the conventional encoder that can only output a single hidden state $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ at each time step. We maintain these chunks by removing the used chunk from them to ensure that each chunk is *used and only used at one level*. To distinguish between our multiple hidden states and the conventional single hidden state, we use the term *chunks* from now on to refer to these hidden states $\boldsymbol{h}_t^l \in \mathbb{R}^{d_h/m}$. The chunk dimension is restricted to be $d_h/m$ to keep the total number of parameters unchanged.

### 3.3.3 Chunk Selection

After obtaining $m$ chunks from our specialized encoding layer, our method repeatedly applys conditional random fields to detect entities from level to level. At each time step, our algorithm selects one chunk from the remains for detection and removes the selected chunk before moving to the next level. From the representation learning aspect, the complete information of all these $m$ levels are distributed in these $m$ chunks, each chunk preserve only a fraction of it. Therefore, we need a chunk strategy to select the most relevant chunk from the remains the current level. For clarity, we use the notation $\mathcal{H}_t^l$ to denote the chunk set at level $l$, and use $\mathcal{H}^l$ to refer to all of these chunk sets at level $m$ across time steps, i.e., $\mathcal{H}_1^l, \ldots, \mathcal{H}_n^l$.

#### The *Naive* Selection Strategy

First of all, the most intuitive selection strategy is to follow the original chunk order and simply select the $l$-th chunk for level $l$. At level $l$, regardless of the label, the emission score is calculated by multiplying the selected chunk $\boldsymbol{h}_t^l$ with the embedding of each label. In this manner, this *naive* potential function can be defined as follows,

$$\phi\,(y_{t-1}^{l}, y_{t}^{l}, \mathcal{H}_{t}^{l}) = A_{y_{t-1}^{l}, y_{t}^{l}} + \boldsymbol{h}_{t}^{l\top} \boldsymbol{v}_{y_{t}^{l}} \tag{3.5}$$

where $\mathbf{A} \in \mathbb{R}^{|\mathcal{Y}| \times |\mathcal{Y}|}$ is the transition matrix, $\mathcal{Y}$ is the label set, $A_{y_{t-1}^{l}, y_{t}^{l}}$ indicates the transition score from label $y_{t-1}^{l}$ to label $y_{t}^{l}$, and $\boldsymbol{v}_{y_{t}^{l}} \in \mathbb{R}^{d_h/m}$ is the embedding of label $y_{t}^{l}$. In this case, the $l$-th chunk $\boldsymbol{h}_{t}^{l} \in \mathcal{H}_{t}^{l}$ is just the chunk that have an interaction with the best path, thus its influence to the best path should be removed as we discussed above. We achieve this by removing it from the chunk set $\mathcal{H}_{t}^{l}$.

$$\mathcal{H}_{t}^{l+1} = \mathcal{H}_{t}^{l} \setminus \{\boldsymbol{h}_{t}^{l}\} \tag{3.6}$$

### The *Max* Selection Strategy

However, one concern with the *naive* potential function is that it implicitly assumes the outputs of the encoder are arranged in level order rather than in some other particular order, such as syntactic or semantic order. For example, it is possible that the encoder gathers `ROLE`-relevant information in the 3rd chunk but remains `PER` relevant information to the 5th chunk. More specifically, at level 3 time step 1, *naive* potential function forces

$$\boldsymbol{h}_{1}^{3\top} \boldsymbol{v}_{\texttt{B-ROLE}} > \boldsymbol{h}_{1}^{3\top} \boldsymbol{v}_{\texttt{B-PER}} \tag{3.7}$$

However, if there exists another chunk, say $\boldsymbol{h}_{1}^{5}$, which contains more `B-PER` relevant information. Then optimizing the following target is more reasonable.

$$\boldsymbol{h}_{1}^{3\top} \boldsymbol{v}_{\texttt{B-ROLE}} > \boldsymbol{h}_{1}^{5\top} \boldsymbol{v}_{\texttt{B-PER}} \tag{3.8}$$

From the perspective of representation learning, this means that we are not only increasing the confidence of assigning $\boldsymbol{h}_{1}^{3}$ a `B-ROLE` tag but also encouraging it to be the most salient chunk. More specifically, this strategy ensures that the confidence of assigning a `B-PER` label to the current level is always lower than `B-ROLE`, regardless of any remaining level. Therefore, the emission score $\boldsymbol{h}_{1}^{3\top} \boldsymbol{v}_{\texttt{B-ROLE}}$ should not only be larger than the emission scores of other labels with respect to it, e.g., $\boldsymbol{h}_{1}^{3\top} \boldsymbol{v}_{\texttt{B-PER}}$, but also be larger than the emission scores of all possible labels with respect to all remaining chunks, e.g., $\boldsymbol{h}_{1}^{5\top} \boldsymbol{v}_{\texttt{B-PER}}$.

In addition, from the optimization perspective, optimizing Equation 3.8 is much harder than forcing Equation 3.7, due to $\boldsymbol{h}_1^{5\top}\boldsymbol{v}_{\texttt{B-PER}} > \boldsymbol{h}_1^{3\top}\boldsymbol{v}_{\texttt{B-PER}}$. This encourages each label embedding to be well-separated [48]. In other words, this new selection strategy leads to the following sequences.

$$\boldsymbol{h}_t^{\sigma_1\top}\boldsymbol{v}_{y_t^1} > \boldsymbol{h}_t^{\sigma_2\top}\boldsymbol{v}_{y_t^2} > \ldots > \boldsymbol{h}_t^{\sigma_m\top}\boldsymbol{v}_{y_t^m} \tag{3.9}$$

Where $\sigma_l$ is the index of the selected chunk at level $l$. However, for the naive potential function, the above inequality does not always hold. In summary, at each level, the max function selects the most salient chunk from the remaining chunks.

Therefore, rather than adhering to the original chunk orders, we propose allowing each label $y_j$ to select the most salient chunk when computing the emission score. We define the *max* potential function as follows,

$$\phi\,(y_{t-1}^l, y_t^l, \mathcal{H}_t^l) = A_{y_{t-1}^l, y_t^l} + \max_{\boldsymbol{h}\in\mathcal{H}_t^l} \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l} \tag{3.10}$$

After decoding, we update the chunk sets by removing the chunks that have interactions with the labels on the best path.

$$\mathcal{H}_t^{l+1} = \mathcal{H}_t^l \setminus \{\arg\max_{\boldsymbol{h}\in\mathcal{H}_t^l} \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l}\} \tag{3.11}$$

## The *LogSumExp* Selection Strategy

Furthermore, the log-sum-exp operation is a well-known differentiable approximation of the max operation. We introduce it as the third potential function,

$$\phi\,(y_{t-1}^l, y_t^l, \mathcal{H}_t^l) = A_{y_{t-1}^l, y_t^l} + \log \sum_{\boldsymbol{h}\in\mathcal{H}_t^l} \exp \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l} \tag{3.12}$$

The chunk set is updated in the same manner as Equation 3.11. We refer to this potential function definition as *logsumexp* for the remainder of this dissertation. One advantage of the log-sum-exp operation is that it back-propagates gradients to all of its inputs, as indicated in Equation 3.14. This is in contrast to the max operation, which only accumulates gradients to the maximal input, as shown in Equation 3.13. The max operator could lead to excessive gradient accumulation for specific chunks, which might result in sub-optimal performance.

$$\frac{\partial \max_{i=1}^{n} x_i}{\partial x_i} = \begin{cases} 1 & (i = \arg\max_{i=1}^{n} x_i) \\ 0 & (\text{otherwise}) \end{cases} \tag{3.13}$$

$$\frac{\partial \log \sum_{i=1}^{n} \exp x_i}{\partial x_i} = \frac{\exp x_i}{\sum_{i=1}^{n} \exp x_i} \tag{3.14}$$

### 3.3.4 Neural Network Architecture

The neural network architecture of our model is mainly based on previous work, with only a few modifications, which we will briefly introduce in this section. Figure 3.2 displays the details of the architecture.

#### Embedding Layer

Following previous work of Shibuya and Hovy [78], we convert words to embeddings $\boldsymbol{w}_t \in \mathbb{R}^{d_w}$ and employ a character-level bidirectional Long Short-term Memory [27] (LSTM) to obtain character-based word embeddings $\boldsymbol{c}_t \in \mathbb{R}^{d_c}$ for leveraging the morphological information. The concatenation of them is fed into the encoding layer as the token representation $\boldsymbol{x}_t = [\boldsymbol{w}_t, \boldsymbol{c}_t] \in \mathbb{R}^{d_x}$.

#### Encoding Layer

We employ a three-layered bidirectional LSTM to encode sentences and to obtain contextual information,

$$[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n] = \text{LSTM}\left([\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]\right) \tag{3.15}$$

where $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is the hidden state. In contrast to the encoders used in previous work, which can only output a single hidden state at each time step, we divide the hidden state $\boldsymbol{h}_t$ into $m$ chunks,

$$[\boldsymbol{h}_t^1, \ldots, \boldsymbol{h}_t^m] = \boldsymbol{h}_t \tag{3.16}$$

where $\boldsymbol{h}_t^j \in \mathbb{R}^{d_h/m}$, and then define the collection of them as the first level chunk set, i.e., $\mathcal{H}_t^1 = \{\boldsymbol{h}_t^j\}_{j=1}^m$ for entity detecting.

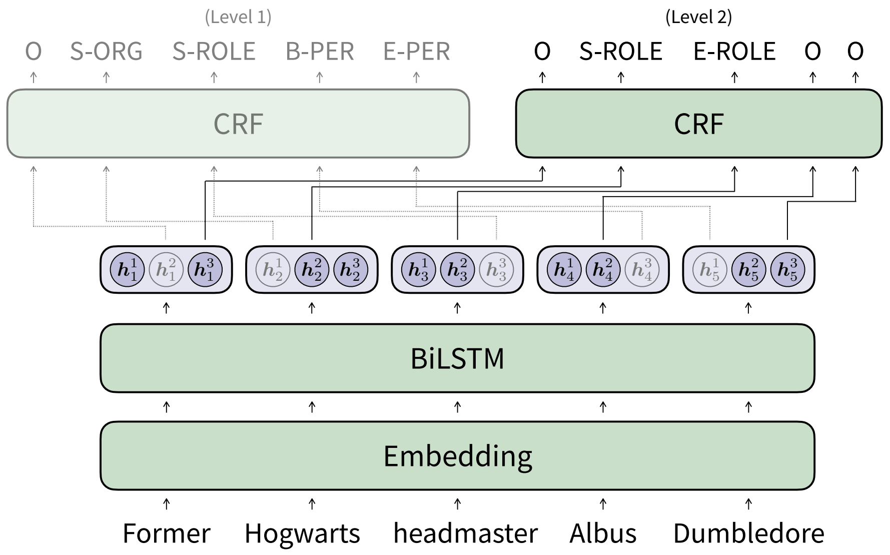

(Level 1)
O S-ORG S-ROLE B-PER E-PER
CRF
(Level 2)
O S-ROLE E-ROLE O O
CRF
BiLSTM
Embedding
Former Hogwarts headmaster Albus Dumbledore

Figure 3.2: The architecture of the proposed nested NER model. These cells indicate chunks obtained from the proposed multi-output bidirectional LSTM. At each level, the CRF layer selects the most salient chunks from the remaining chunks and utilizes them for decoding. Transparent cells indice used chunks from previous levels. To save space, we only display the decoding procedures for the first two levels.

**Decoding Layer**

At each level, we run a shared conventional CRF with its corresponding potential function $\phi\,(y_{t-1}^l, y_t^l, \mathcal{H}_t^l)$ and update the chunk sets until finishing all $m$ levels. During the training stage, we remove chunks based on the selections of the ground-truth target labels, while during the decoding stage, the removal depends on the selections of the predicted labels.

### 3.3.5 Training and Inference

Following the definition of CRF, the conditional probability function for a given label sequence at the $l$-th level, i.e., $\boldsymbol{y}^l = \{y_t^l\}_{t=1}^n$, is defined as,

$$p\,(\boldsymbol{y}^l \mid \mathcal{H}^l) = \frac{1}{Z(\mathcal{H}^l)} \exp \sum_{t=1}^{n} \phi\,(y_{t-1}^l, y_t^l, \mathcal{H}_t^l) \tag{3.17}$$

**Algorithm 1:** Training (with *max* or *logsumexp* potential function)

```
input  : first level chunk sets $\mathcal{H}^1$
input  : target label sequences $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^m$
output: negative log-likelihood $\mathcal{L}$
$\mathcal{L} \leftarrow 0$
for $l = 1$ to $m$ do
    $\mathcal{L} \leftarrow \mathcal{L} - \log p\,(\boldsymbol{y}^l \mid \mathcal{H}^l)$
    for $t = 1$ to $n$ do
        $\mathcal{H}_t^{l+1} \leftarrow \mathcal{H}_t^l \setminus \{\arg\max_{\boldsymbol{h} \in \mathcal{H}_t^l} \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l}\}$
    end
end
```

$$Z(\mathcal{H}^l) = \sum_{\boldsymbol{y}' \in \mathcal{Y}^n} \exp \sum_{t=1}^{n} \phi\,(y'^l_{t-1}, y'^l_t, \mathcal{H}_t^l) \tag{3.18}$$

where $Z(\mathcal{H}^l)$ is the sum of the scores of all paths and is commonly known as the partition function.

During the training stage, we optimize our model by minimizing the sum of the negative log-likelihoods across all levels.

$$\mathcal{L} = -\sum_{l=1}^{m} \log p\,(\boldsymbol{y}^l \mid \mathcal{H}^l) \tag{3.19}$$

During the decoding stage, we iteratively apply the Viterbi algorithm [23] at each level to search for the most probable label sequences.

$$\hat{\boldsymbol{y}}^l = \arg\max_{\boldsymbol{y}' \in \mathcal{Y}^n} p\,(\boldsymbol{y}' \mid \mathcal{H}^l) \tag{3.20}$$

The pseudo-code for the training and decoding algorithms using either the *max* or *logsumexp* potential functions can be found in Algorithms 1 and 2, respectively.

**Algorithm 2:** Decoding (with *max* or *logsumexp* potential function)

```
input  : first level chunk sets $\mathcal{H}^1$
output: recognized entity set $\mathcal{E}$
$\mathcal{E} \leftarrow \varnothing$
for $l = 1$ to $m$ do
    $\hat{\boldsymbol{y}}^l \leftarrow \arg\max_{\boldsymbol{y}' \in \mathcal{Y}^n} p\left(\boldsymbol{y}' \mid \mathcal{H}^l\right)$
    for $t = 1$ to $n$ do
        $\mathcal{H}_t^{l+1} \leftarrow \mathcal{H}_t^l \setminus \{\arg\max_{\boldsymbol{h} \in \mathcal{H}_t^l} \boldsymbol{h}^\top \boldsymbol{v}_{\hat{y}_t^l}\}$
    end
    $\mathcal{E} \leftarrow \mathcal{E} \bigcup$ label-to-entity $(\hat{\boldsymbol{y}}^l)$
end
```

## 3.4 Experiments

### 3.4.1 Datasets

Following the standard benchmark settings, we conduct experiments on four nested NER datasets in English with diverse domains, namely, ACE2004 [19], ACE2005 [93], GENIA [35], and NNE [75]. We split all these datasets into training, development, and test sets in accordance with the methodology followed by Shibuya and Hovy [78] and Wang et al. [98]. Among these, ACE2004, ACE2005, and NNE datasets are in the news domain and have up to 6 levels, while the GENIA dataset belongs to the biomedical domain and is relatively shallow, with a maximum of 4 levels. Specifically, the NNE dataset contains 427 distinct labels, making it far more fine-grained than the other datasets, which have up to 29 distinct labels. Detailed dataset statistics can be found in Table 3.1, where $|\mathcal{Y}|$ represents the size of the label set and $m$ is the maximum depth of entity nesting. The number of entities is displayed according to level and data split.

### 3.4.2 Settings

For the initialization of word embeddings, we utilize 100-dimensional pre-trained GloVe embeddings [65] for the ACE2004, ACE2005, and NNE datasets. For

| Dataset | (Level) | Sentences Train | Sentences Dev | Sentences Test | Entities Train | Entities Dev | Entities Test | $\lvert\mathcal{Y}\rvert$ | $m$ |
|---|---|---|---|---|---|---|---|---|---|
| ACE2004 | 1 | 3,480 | 448 | 421 | 17,305 | 2,002 | 2,379 | 29 | 6 |
| | 2 | 1,994 | 216 | 295 | 3,967 | 413 | 546 | | |
| | 3 | 620 | 66 | 85 | 804 | 84 | 100 | | |
| | 4 | 91 | 10 | 7 | 104 | 13 | 8 | | |
| | 5 | 11 | 2 | 1 | 13 | 2 | 1 | | |
| | 6 | 2 | 0 | 0 | 2 | 0 | 0 | | |
| | Overall | 6,198 | 742 | 809 | 22,195 | 2,514 | 3,034 | | |
| ACE2005 | 1 | 4,488 | 616 | 719 | 19,911 | 2,651 | 2,467 | 29 | 6 |
| | 2 | 2,096 | 277 | 265 | 3,939 | 479 | 466 | | |
| | 3 | 610 | 69 | 62 | 751 | 82 | 83 | | |
| | 4 | 85 | 6 | 11 | 91 | 6 | 12 | | |
| | 5 | 4 | 0 | 1 | 6 | 0 | 1 | | |
| | 6 | 2 | 0 | 0 | 2 | 0 | 0 | | |
| | Overall | 7,285 | 968 | 1,058 | 24,700 | 3,218 | 3,029 | | |
| GENIA | 1 | 11,800 | 1,341 | 1,407 | 42,967 | 4,072 | 5,007 | 21 | 4 |
| | 2 | 3,123 | 327 | 433 | 3,939 | 388 | 574 | | |
| | 3 | 98 | 1 | 15 | 99 | 1 | 15 | | |
| | 4 | 1 | 0 | 0 | 1 | 0 | 0 | | |
| | Overall | 15,022 | 1,669 | 1,855 | 47,006 | 4,461 | 5,596 | | |
| NNE | 1 | 14,851 | 697 | 1,273 | 164,907 | 7,055 | 13,963 | 427 | 6 |
| | 2 | 17,417 | 873 | 1,474 | 64,836 | 2,775 | 5,544 | | |
| | 3 | 9,442 | 354 | 817 | 16,444 | 560 | 1,465 | | |
| | 4 | 1,644 | 62 | 186 | 1,842 | 70 | 212 | | |
| | 5 | 101 | 3 | 12 | 105 | 3 | 12 | | |
| | 6 | 2 | 0 | 0 | 2 | 0 | 0 | | |
| | Overall | 43,457 | 1,989 | 3,762 | 248,136 | 10,463 | 21,196 | | |

Table 3.1: Statistics on the ACE2004, ACE2005, GENIA, and NNE datasets. $|\mathcal{Y}|$ means the label size, and $m$ shows the maximal depth of entity nesting.

the GENIA dataset, we use 200-dimensional biomedical domain-specific word embeddings[1] [14]. In addition, we randomly initialize 30-dimensional vectors for character embeddings. The hidden state dimension $d_c$ of the character-level LSTM is set to 100, i.e., 50 in each direction. Therefore, the dimension $d_x$ of the token representation is 200. We apply dropout [79] to the token representations before feeding them into the encoder.

The hidden state dimension of the three-layered bidirectional LSTM is set to 600 for the ACE2004, ACE2005, and NNE datasets, i.e., 300 in each direction. For the GENIA dataset, it is set to 400. The choice of different dimensions is influenced by the maximal depth of entity nesting $m$ in each dataset. In other words, we keep the chunk dimensions across all datasets to be 100. We apply layer normalization [4] and a dropout ratio of 0.5 after each bidirectional LSTM layer.

Differing from the approach of Shibuya and Hovy [78], we employ a single CRF rather than using different CRFs for various entity types. Furthermore, our CRF is shared across different levels, meaning that the same CRF is used for entity detection at all levels.

Our model is optimized using stochastic gradient descent (SGD), with a decaying learning rate given by the formula $\eta_\tau = \frac{\eta_0}{1 + \gamma \cdot \tau}$, where $\tau$ is the index of the current epoch. For the ACE2004, ACE2005, GENIA, and NNE datasets, the initial learning rates $\eta_0$ are 0.2, 0.2, 0.1, and 0.2, and the decay rates $\gamma$ are 0.01, 0.02, 0.02, and 0.02 respectively. We set the weight decay rate, the momentum, the batch size, and the number of epochs to be $10^{-8}$, 0.5, 32, and 100 respectively. Specifically, we use a batch size of 64 for the GENIA dataset. Gradients exceeding a value of 5 are clipped.

Besides, we conduct experiments to assess how our model performs when incorporating contextual word representations. Both BERT [18] and Flair [2] are popular choices for this purpose in previous works and have demonstrated their effectiveness in significantly enhancing model performance. In these configurations, we concatenate the contextual word representations with the word and character representations to form a more comprehensive token representation, i.e., $\boldsymbol{x}_t = [\boldsymbol{w}_t, \boldsymbol{c}_t, \boldsymbol{e}_t]$, where $\boldsymbol{e}_t$ denotes the contextual word representation. It's worth noting that we do not fine-tune these pre-trained language models in any

[1] `https://github.com/cambridgeltl/BioNLP-2016`

of our experiments.

**BERT** The contextual word vectors are obtained from a pre-trained transformer-based model [89, 18]. In our experiments, for the ACE2004, ACE2005, and NNE datasets, we use the general domain checkpoint `bert-large-uncased`. For the GENIA dataset, we use the biomedical domain checkpoint `BioBERT large v1.1` [2] [44]. We average all BERT subword embeddings from the last four layers to build 1024-dimensional vectors.

**Flair** The embeddings are obtained from a pre-trained character-level bidirectional LSTM language model. We concatenate these vectors obtained from the `news-forward` and `news-backward` checkpoints for ACE2004, ACE2005, and NNE, and use the `pubmed-forward` and `pubmed-backward` checkpoints for GENIA to build 4096-dimensional vectors.

Experiments are all evaluated using precision, recall, and $F_1$. All of our experiments were run four times with different random seeds, and the averaged scores are reported in the following tables.

Our model is implemented with `PyTorch` [63] and we run experiments on a `GeForce GTX 1080Ti` with 11 GB memory.

### 3.4.3 Main Results

Table 3.2 shows the performance of previous work and our model on the ACE2004 datasets. Our model substantially outperforms most of the previous work, especially when compared with our baseline Shibuya and Hovy [78]. When utilizing only word embeddings and character-based word embeddings, our method exceeds theirs by a 2.64 $F_1$ score and also achieves comparable results with the recent competitive method [98]. In the case of utilizing BERT and further employing Flair, our method consistently outperforms [78] by 1.09 and 0.60 $F_1$ scores, respectively.

On the ACE2005 dataset, as shown in Figure 3.3, our method improves the $F_1$ scores by 1.98, 0.72, and 0.59, respectively, compared with Shibuya and Hovy [78]. Although our model's performance is generally inferior to Wang et al. [98],

[2]`https://github.com/naver/biobert-pretrained`

| Method | Precision | Recall | $F_1$ |
|---|---|---|---|
| Wang et al. [96] | 74.9 | 71.8 | 73.3 |
| Wang and Lu [95] | 78.0 | 72.4 | 75.1 |
| Straková et al. [81] | 78.92 | 75.33 | 77.08 |
| Shibuya and Hovy [78] | 79.93 | 75.10 | 77.44 |
| Wang et al. [98] | 80.83 | 78.86 | 79.83 |
| Our Method (Naive) | 81.12 | 77.71 | 79.38 (0.31) |
| Our Method (Max) | 81.90 | 78.05 | <u>79.92</u> (0.10) |
| Our Method (LogSumExp) | 81.24 | 78.96 | **80.08** (0.22) |
| with Bert | | | |
| Straková et al. [81] | 84.71 | 83.96 | 84.33 |
| Shibuya and Hovy [78] | 85.23 | 84.72 | 84.97 |
| Wang et al. [98] | 86.08 | 86.48 | **86.28** |
| Our Method (Naive) | 86.19 | 85.28 | 85.73 (0.24) |
| Our Method (Max) | 86.27 | 85.09 | 85.68 (0.09) |
| Our Method (LogSumExp) | 86.42 | 85.71 | <u>86.06</u> (0.10) |
| with Bert and Flair | | | |
| Straková et al. [81] | 84.51 | 84.29 | 84.40 |
| Shibuya and Hovy [78] | 85.94 | 85.69 | 85.82 |
| Wang et al. [98] | 87.01 | 86.55 | **86.78** |
| Our Method (Naive) | 86.56 | 85.65 | 86.11 (0.24) |
| Our Method (Max) | 86.96 | 85.45 | 86.19 (0.17) |
| Our Method (LogSumExp) | 86.74 | 86.11 | <u>86.42</u> (0.31) |

Table 3.2: Experiments on the ACE2004 dataset. The **bold** numbers indicate the best performance, while the <u>underlined</u> numbers represent the second-best results. Numbers in parentheses are the standard variance.

our proposed method with the *max* selection strategy is slightly superior to them by 0.05 in $F_1$ score when adding BERT in.

Furthermore, on the biomedical domain dataset GENIA, as shown in Figure 3.4, our method consistently outperforms Shibuya and Hovy [78] by 0.18, 1.62,

| Method | Precision | Recall | F$_1$ |
|---|---|---|---|
| Ju et al. [32] | 74.2 | 70.3 | 72.2 |
| Wang et al. [96] | 74.5 | 71.5 | 73.0 |
| Wang and Lu [95] | 76.8 | 72.3 | 74.5 |
| Luo and Zhao [53] | 75.0 | 75.2 | 75.1 |
| Lin et al. [47] | 76.2 | 73.6 | 74.9 |
| Straková et al. [81] | 76.35 | 74.39 | 75.36 |
| Shibuya and Hovy [78] | 78.27 | 75.44 | 76.83 |
| Wang et al. [98] | 79.27 | 79.37 | **79.32** |
| Our Method (Naive) | 79.45 | 77.22 | 78.32 (0.26) |
| Our Method (Max) | 80.68 | 77.03 | <u>78.81</u> (0.04) |
| Our Method (LogSumExp) | 79.49 | 77.65 | 78.55 (0.12) |
| with Bert | | | |
| Straková et al. [81] | 82.58 | 84.29 | 83.42 |
| Shibuya and Hovy [78] | 83.30 | 84.69 | 83.99 |
| Wang et al. [98] | 83.95 | 85.39 | <u>84.66</u> |
| Our Method (Naive) | 84.23 | 84.17 | 84.20 (0.30) |
| Our Method (Max) | 85.28 | 84.15 | **84.71** (0.09) |
| Our Method (LogSumExp) | 83.95 | 84.67 | 84.30 (0.13) |
| with Bert and Flair | | | |
| Straková et al. [81] | 83.48 | 85.21 | 84.33 |
| Shibuya and Hovy [78] | 83.83 | 84.87 | 84.34 |
| Wang et al. [98] | 84.90 | 86.08 | **85.49** |
| Our Method (Naive) | 84.17 | 84.88 | 84.52 (0.21) |
| Our Method (Max) | 84.70 | 84.76 | 84.73 (0.21) |
| Our Method (LogSumExp) | 84.81 | 85.06 | <u>84.93</u> (0.24) |

Table 3.3: Experiments on the ACE2005 dataset.

and 1.57 in F$_1$ scores, respectively. Although the low scores of Shibuya and Hovy [78] are due to their use of the general domain checkpoint `bert-large-uncased`, instead of our biomedical domain checkpoint, our model is still superior to Straková

| Method | Precision | Recall | $F_1$ |
|---|---|---|---|
| Ju et al. [32] | 78.5 | 71.3 | 74.7 |
| Wang et al. [96] | 78.0 | 70.2 | 73.9 |
| Wang and Lu [95] | 77.0 | 73.3 | 75.1 |
| Luo and Zhao [53] | 77.4 | 74.6 | 76.0 |
| Lin et al. [47] | 75.8 | 73.9 | 74.8 |
| Straková et al. [81] | 79.60 | 73.53 | 76.44 |
| Shibuya and Hovy [78] | 78.70 | 75.74 | 77.19 |
| Wang et al. [98] | 77.91 | 77.20 | **77.55** |
| Our Method (Naive) | 78.83 | 75.32 | 77.03 (0.13) |
| Our Method (Max) | 78.80 | 75.71 | 77.22 (0.10) |
| Our Method (LogSumExp) | 78.58 | 76.21 | $\underline{77.37}$ (0.15) |
| with Bert | | | |
| Straková et al. [81] | 79.92 | 76.55 | 78.20 |
| Shibuya and Hovy [78] | 77.46 | 76.65 | 77.05 |
| Wang et al. [98] | 79.45 | 78.94 | **79.19** |
| Our Method (Naive) | 78.83 | 78.07 | 78.45 (0.32) |
| Our Method (Max) | 79.20 | 78.16 | $\underline{78.67}$ (0.18) |
| Our Method (LogSumExp) | 78.83 | 78.27 | 78.54 (0.02) |
| with Bert and Flair | | | |
| Straková et al. [81] | 80.11 | 76.60 | 78.31 |
| Shibuya and Hovy [78] | 77.81 | 76.94 | 77.36 |
| Wang et al. [98] | 79.98 | 78.51 | **79.24** |
| Our Method (Naive) | 79.28 | 78.31 | 78.79 (0.17) |
| Our Method (Max) | 79.51 | 78.25 | 78.87 (0.04) |
| Our Method (LogSumExp) | 79.20 | 78.67 | $\underline{78.93}$ (0.26) |

Table 3.4: Experiments on the GENIA dataset.

et al. [81] by 0.47 and 0.62 in $F_1$ scores, who used the same checkpoint as us.

On the NNE dataset, as shown in Table 3.5, our method slightly but consistently outperforms Wang et al. [98] by 0.08, 0.19, and 0.22 in $F_1$ scores, respec-

| Method | Precision | Recall | $F_1$ |
|---|---|---|---|
| Wang et al. [96] | 77.4 | 70.1 | 73.6 |
| Wang and Lu [95] | 91.8 | 91.0 | 91.4 |
| Wang et al. [98] | 93.37 | 93.91 | 93.64 |
| Our Method (Naive) | 93.65 | 93.69 | <u>93.67</u> (0.05) |
| Our Method (Max) | 94.04 | 93.23 | 93.63 (0.04) |
| Our Method (LogSumExp) | 93.97 | 93.52 | **93.75** (0.05) |
| with Bert | | | |
| Wang et al. [98] | 93.97 | 94.79 | 94.37 |
| Our Method (Naive) | 94.49 | 94.52 | 94.50 (0.03) |
| Our Method (Max) | 94.84 | 93.56 | **94.58** (0.07) |
| Our Method (LogSumExp) | 94.67 | 94.47 | <u>94.56</u> (0.03) |
| with Bert and Flair | | | |
| Wang et al. [98] | 93.97 | 94.98 | 94.47 |
| Our Method (Naive) | 94.59 | 94.65 | <u>94.62</u> (0.03) |
| Our Method (Max) | 94.79 | 94.44 | 94.61 (0.04) |
| Our Method (LogSumExp) | 94.67 | 94.71 | **94.69** (0.06) |

Table 3.5: Experiments on the NNE dataset.

tively. We hypothesize that this is because the NNE dataset contains many more sentences and entity types than the other three datasets. Our method is much more capable of learning meaningful representations when data are sufficient.

As for these three potential functions, we notice that the *max* and *logsumexp* potential functions generally work better than the *naive* potential function. These results demonstrate that the chunk selection strategies of the *max* and *logsumexp* can leverage information from all remaining chunks and constrain the hidden states of the LSTM to be more semantically ordered. When we use BERT and Flair, the advantage of the *max* and *logsumexp* potential functions is less obvious compared with the case when we use only word embeddings and character-based word embeddings, especially on the GENIA dataset. We hypothesize that BERT and Flair can provide rich contextual information, therefore, selecting chunks in

| Encoding Scheme | $\phi$ | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| Outermost First | Naive | 79.08 | 76.57 | 77.80 (0.26) |
| | Max | 79.07 | 75.11 | 77.04 (0.20) |
| | LogSumExp | 79.05 | 76.39 | 77.70 (0.32) |
| Innermost First | Naive | 81.12 | 77.71 | 79.38 (0.31) |
| | Max | 81.90 | 78.05 | <u>79.92</u> (0.10) |
| | LogSumExp | 81.24 | 78.96 | **80.08** (0.22) |

Table 3.6: Ablation study on the influence of the encoding schemes and potential functions.

the original order is sufficient. Thus, our dynamic selection mechanism can only slightly improve model performance.

### 3.4.4 Ablation Studies

#### Influence of the Encoding Scheme

We also conduct experiments on the ACE2004 dataset to measure the influence of the outermost-first and innermost-first encoding schemes. As shown in Table 3.6, the innermost-first encoding scheme consistently works better than the outermost-first encoding scheme across all potential functions. We hypothesize that outermost entities do not necessarily contain inner entities, especially for longer ones, and that placing those diversely nested outermost entities at the same level would disrupt the encoding representation. Furthermore, when we use the outermost-first encoding scheme, our method is superior to Shibuya and Hovy [78], which further demonstrates the effectiveness of excluding the influence of the best path.

#### Level-wise Performance

We display the performance on the ACE2005 dataset at each level in Table 3.7. The *max* potential function at the first three levels consistently achieves higher precision scores than the *naive* and *logsumexp* potential functions, while simul-

| Level | Naive |  | Max |  | LogSumExp |  |
|---|---|---|---|---|---|---|
|  | Precision | Recall | Precision | Recall | Precision | Recall |
| 1 | 80.83 | 80.12 | **82.14** | 79.51 | <u>80.98</u> | 80.12 |
| 2 | <u>73.91</u> | 68.67 | **74.76** | 70.76 | 73.85 | 70.76 |
| 3 | 60.09 | 48.80 | **65.26** | 49.10 | <u>60.17</u> | 53.01 |
| 4 | **100.00** | 16.67 | 37.50 | 10.42 | <u>66.67</u> | 14.58 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | - | - | - | - | - | - |
| Overall | 79.45 | 77.22 | **80.68** | 77.03 | <u>79.49</u> | 77.65 |

Table 3.7: Ablation study of level-wise performance.

taneously obtaining the lowest recall scores. The *logsumexp* potential function, on the contrary, achieves the highest recall scores but fails to obtain satisfactory precision scores. Because most entities are located at the first two levels, the *max* and *logsumexp* achieve the best overall precision and recall scores, respectively.

Because the max operator back-propagates gradients to only one of its inputs, while the log-sum-exp operator back-propagates gradients to all of its inputs according to their normalized weights, it is possible for the max operator to accumulate excessive gradients to some particular chunks. As a result, it might be prone to converging to sub-optimal model performance.

### Connection to Attention Mechanism

The heat maps of chunk selection in Figure 3.4 raises a question about the connection between our methods and the attention mechanism [91]. In this section, we will briefly explain the connections and present the results of ablation studies to demonstrate that our methods are either equivalent to or better than the attention mechanism.

As mentioned in Section 3.3.3, our dynamic chunk selection strategies enable each label to choose the most similar chunk to it. Therefore, the concept of replacing our dynamic selection strategies with an attention mechanism could be realized by allowing each label embedding to attend to all the remaining chunks and constructing a chunk vector by aggregating information from all of them.

$$\tilde{\boldsymbol{h}}_t^l = \text{attention}\left(\texttt{query} = \boldsymbol{v}_{y_t^l}, \texttt{key} = \mathcal{H}^l, \texttt{value} = \mathcal{H}^l\right) \tag{3.21}$$

Subsequently, $\tilde{\boldsymbol{h}}_t^l$ is used to compute the dot product with the label embedding, similar to Equation 3.5.

$$\phi\left(y_{t-1}^l, y_t^l, \mathcal{H}_t^l\right) = A_{y_{t-1}^l, y_t^l} + \tilde{\boldsymbol{h}}_t^{l\top} \boldsymbol{v}_{y_t^l} \tag{3.22}$$

Attention mechanisms can be roughly divided into two categories: hard attention and soft attention. On the one hand, the hard attention mechanism only selects the chunk with the highest score.

$$\begin{aligned} \tilde{\boldsymbol{h}}_t^l &= \texttt{attention}\left(\texttt{query} = \boldsymbol{v}_{y_t^l}, \texttt{key} = \mathcal{H}^l, \texttt{value} = \mathcal{H}^l\right) && (3.23) \\ &= \arg\max_{\boldsymbol{h} \in \mathcal{H}^l} \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l} && (3.24) \end{aligned}$$

Therefore,

$$\begin{aligned} \phi\left(y_{t-1}^l, y_t^l, \mathcal{H}_t^l\right) &= A_{y_{t-1}^l, y_t^l} + \tilde{\boldsymbol{h}}_t^{l\top} \boldsymbol{v}_{y_t^l} && (3.25) \\ &= A_{y_{t-1}^l, y_t^l} + \left(\arg\max_{\boldsymbol{h} \in \mathcal{H}^l} \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l}\right)^\top \boldsymbol{v}_{y_t^l} && (3.26) \\ &= A_{y_{t-1}^l, y_t^l} + \max_{\boldsymbol{h} \in \mathcal{H}^l} \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l} && (3.27) \end{aligned}$$

It is easy to notice that Equation 3.27 is equivalent to our *max* potential function, i.e., Equation 3.10.

On the other hand, soft attention aggregates information from all chunks by considering normalized similarities as weights.

$$\begin{aligned} \tilde{\boldsymbol{h}}_t^l &= \texttt{attention}\left(\texttt{query} = \boldsymbol{v}_{y_t^l}, \texttt{key} = \mathcal{H}^l, \texttt{value} = \mathcal{H}^l\right) && (3.28) \\ &= \sum_{\hat{\boldsymbol{h}} \in \mathcal{H}^l} \text{softmax}\left(\hat{\boldsymbol{h}}^\top \boldsymbol{v}_{y_t^l}\right) \cdot \hat{\boldsymbol{h}} && (3.29) \\ &= \sum_{\hat{\boldsymbol{h}} \in \mathcal{H}^l} \left(\frac{\exp \hat{\boldsymbol{h}}^\top \boldsymbol{v}_{y_t^l}}{\sum_{\boldsymbol{h} \in \mathcal{H}^l} \exp \boldsymbol{h}^\top \boldsymbol{v}_{y_t^l}}\right) \cdot \hat{\boldsymbol{h}} && (3.30) \end{aligned}$$

Similarly,

$$\begin{aligned}
\phi\left(y_{t-1}^{l}, y_{t}^{l}, \mathcal{H}_{t}^{l}\right) &= A_{y_{t-1}^{l}, y_{t}^{l}} + \tilde{\boldsymbol{h}}_{t}^{l\top} \boldsymbol{v}_{y_{t}^{l}} && (3.31)\\
&= A_{y_{t-1}^{l}, y_{t}^{l}} + \left(\sum_{\hat{\boldsymbol{h}} \in \mathcal{H}^{l}} \left(\frac{\exp \hat{\boldsymbol{h}}^{\top} \boldsymbol{v}_{y_{t}^{l}}}{\sum_{\boldsymbol{h} \in \mathcal{H}^{l}} \exp \boldsymbol{h}^{\top} \boldsymbol{v}_{y_{t}^{l}}}\right) \cdot \hat{\boldsymbol{h}}\right)^{\top} \boldsymbol{v}_{y_{t}^{l}} && (3.32)\\
&= A_{y_{t-1}^{l}, y_{t}^{l}} + \sum_{\hat{\boldsymbol{h}} \in \mathcal{H}^{l}} \left(\frac{\exp \hat{\boldsymbol{h}}^{\top} \boldsymbol{v}_{y_{t}^{l}}}{\sum_{\boldsymbol{h} \in \mathcal{H}^{l}} \exp \boldsymbol{h}^{\top} \boldsymbol{v}_{y_{t}^{l}}}\right) \cdot \left(\hat{\boldsymbol{h}}^{\top} \boldsymbol{v}_{y_{t}^{l}}\right) && (3.33)\\
&= A_{y_{t-1}^{l}, y_{t}^{l}} + \frac{\sum_{\boldsymbol{h} \in \mathcal{H}^{l}} \left(\exp \boldsymbol{h}^{\top} \boldsymbol{v}_{y_{t}^{l}}\right) \cdot \left(\boldsymbol{h}^{\top} \boldsymbol{v}_{y_{t}^{l}}\right)}{\sum_{\boldsymbol{h} \in \mathcal{H}^{l}} \exp \boldsymbol{h}^{\top} \boldsymbol{v}_{y_{t}^{l}}} && (3.34)
\end{aligned}$$

This is, in fact, another differentiable approximation of the *max* operator.

$$\max\left(x_{1}, \ldots, x_{n}\right) \approx \frac{\sum_{i=1}^{n} x_{i} \cdot \exp x_{i}}{\sum_{i=1}^{n} \exp x_{i}} \tag{3.35}$$

Since this potential function utilizes the softmax function, we refer to it as *softmax-x*.

We conducted experiments on the ACE2004 dataset, and the experimental results are presented in Table 3.8. It is important to note that the performance of *softmax-x* is slightly better than *naive*, but the $\mathrm{F}_1$ score is still much lower than that of *max* and *logsumexp*. To simplify and illustrate the issue, we plotted the curves of $\max(x, 0)$, $\log(\exp x + \exp 0)$, and $\frac{x \cdot \exp x + 0 \cdot \exp 0}{\exp x + \exp 0}$, which represent the *max* operator and its two differentiable approximations, in Figure 3.3. The left part of the figure displays these three curves, while their gradient curves are shown on the right side. By examining these curves, we can observe that

| $\phi$ | Precision | Recall | $\mathrm{F}_1$ |
|---|---|---|---|
| Naive | 81.12 | 77.71 | 79.38 (0.31) |
| Max | 81.90 | 78.05 | <u>79.92</u> (0.10) |
| LogSumExp | 81.24 | 78.96 | **80.08** (0.22) |
| SoftmaxX | 80.93 | 78.15 | 79.51 (0.40) |

Table 3.8: Ablation study on various attention mechanisms.

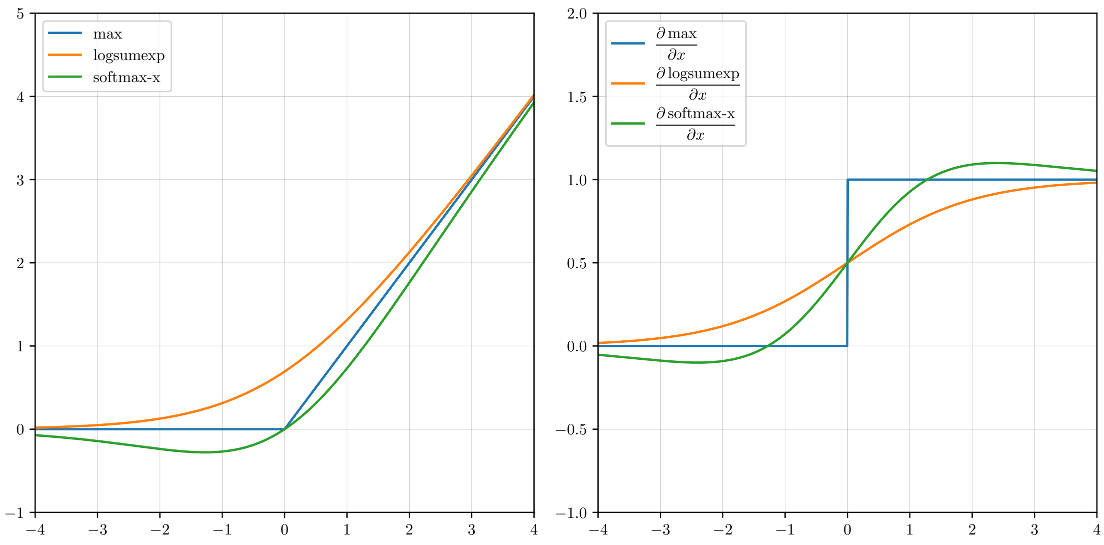


Figure 3.3: Curves of *max*, *logsumexp*, and *softmax-x* functions along with their gradients.

the gradient of *softmax-x* is not always monotonically increasing, which leads to training instability.

### 3.4.5 Case Studies

In this section, we present some prediction examples in Table 3.9 to compare the behaviors of different models. In the first example, although the model of [78] is designed to detect entities from outer to inner, it fails to recognize the entity *griffen bell*, which is nested within the outer entity *prominent democrats including griffen bell*. In contrast, all three of our potential functions successfully detected it. Similarly, their model mistakenly predicts *gore* as a single-word entity, whereas none of our methods make this mistake. For the longest entity, *gore who … in the state*, their model correctly detects the opening bracket but prematurely closes the bracket after *this year*. Surprisingly, we found that our *naive* potential function makes the same mistake, while our *max* and *logsumexp* potential functions correctly close the bracket at the end of the sentence. This observation supports our claim that our dynamic selection strategies are capable of leveraging information among all remaining chunks.

| Method | Example |
|---|---|
| Target | [bush]$_{\text{PER}}$ in [tennessee]$_{\text{GPE}}$ today [where]$_{\text{GPE}}$ the race on [the home turf]$_{\text{GPE}}$ is surprisingly tight , announcing the endorsement of [prominent democrats including [griffen bell]$_{\text{PER}}$]$_{\text{PER}}$ , and taunting [gore [who]$_{\text{PER}}$ earlier this year was accused by [a tenant]$_{\text{PER}}$ of failing to properly maintain [rental property [he]$_{\text{PER}}$ owns in [the state]$_{\text{GPE}}$]$_{\text{FAC}}$]$_{\text{PER}}$ . |
| Shibuya and Hovy [78] | [bush]$_{\text{PER}}$ in [tennessee]$_{\text{GPE}}$ today [where]$_{\text{GPE}}$ the race on the home turf is surprisingly tight , announcing the endorsement of [prominent democrats including griffen bell]$_{\text{PER}}$ , and taunting [[gore]$_{\text{PER}}$ [who]$_{\text{PER}}$ earlier this year]$_{\text{PER}}$ was accused by [a tenant of failing to properly maintain rental property [he]$_{\text{PER}}$ owns in [the state]$_{\text{GPE}}$]$_{\text{PER}}$ . |
| Naive | [bush]$_{\text{PER}}$ in [tennessee]$_{\text{GPE}}$ today where the race on the home turf is surprisingly tight , announcing the endorsement of [prominent democrats including [griffen bell]$_{\text{PER}}$]$_{\text{PER}}$ , and taunting [gore [who]$_{\text{PER}}$ earlier this year]$_{\text{PER}}$ was accused by a tenant of failing to properly maintain [rental property [he]$_{\text{PER}}$ owns in [the state]$_{\text{GPE}}$]$_{\text{FAC}}$ . |
| Max | [bush]$_{\text{PER}}$ in [tennessee]$_{\text{GPE}}$ today [where]$_{\text{GPE}}$ the race on the home turf is surprisingly tight , announcing the endorsement of [prominent democrats including [griffen bell]$_{\text{PER}}$]$_{\text{PER}}$ , and taunting [gore [who]$_{\text{PER}}$ earlier this year was accused by a tenant of failing to properly maintain rental property [he]$_{\text{PER}}$ owns in [the state]$_{\text{GPE}}$]$_{\text{PER}}$ . |
| LogSumExp | [bush]$_{\text{PER}}$ in [tennessee]$_{\text{GPE}}$ today [where]$_{\text{GPE}}$ the race on the home turf is surprisingly tight , announcing the endorsement of [prominent democrats including [griffen bell]$_{\text{PER}}$]$_{\text{PER}}$ , and taunting [gore [who]$_{\text{PER}}$ earlier this year was accused by a tenant of failing to properly maintain rental property [he]$_{\text{PER}}$ owns in [the state]$_{\text{GPE}}$]$_{\text{PER}}$ . |

Table 3.9: Case study. Brackets mark the boundary of entities, and the corresponding categories are annotated in the lower right corner. Additionally, `PER`, `GPE`, and `FAC` indicate that the entity types are person, geopolitical entity, and artifact, respectively.

### 3.4.6 Discussions

#### Chunk Distribution

We analyze the chunk distribution on the test split of the ACE2005 dataset by plotting heat maps in Figure 3.4. In these heat maps, the numbers indicate the percentages of each chunk being selected by a particular level or label. For example, the number 35 in the upper-right corner indicates that, when using the *logsumexp* potential function, 35% of predictions at the first level are made by choosing the sixth chunk. Similarly, the number 78 in the lower-left corner indicates that 78% of `WEA` entities are related to the first chunk when leveraging the *naive* potential function. To facilitate comparison with the *naive* potential function, we arranged the chunk orders of *max* and *logsumexp* in a way that does not lose generality and makes the level-chunk distribution primarily concentrate on the diagonal.

The *naive* potential function simply selects the $l$-th chunk at the $l$-th level, resulting in a diagonal heat map. In addition to the first chunk, the *logsumexp* potential function also tends to select the sixth and fourth chunks at the first level. We hypothesize that this is due to most of the `B-` and `S-` labels being located on the first level. This can be confirmed by analyzing the syntactic-chunk heat map of the *logsumexp* potential function, where 78% of `B-` and 70% of `S-` labels are associated with the sixth and fourth chunks. Similarly, the *max* potential function also exhibits a high probability of selecting the second chunk.

Generally, the chunk distribution of the *logsumexp* potential function is smoother than that of *max*. Additionally, we observe that the `O` label almost uniformly selects chunks in both the syntactic and semantic heat maps, while other meaningful labels exhibit distinct preferences.

Syntactic labels `S-` and `B-` mainly indicate the beginning of an entity, while `I-` and `E-` denote the continuation and ending of an entity, respectively. In the syntactic-chunk heat map of the *naive* potential function, these labels are uniformly distributed to the first chunk because most of the entities are located on the first level. However, the *max* and *logsumexp* potential functions use different chunks to represent these distinct syntactic categories.

Likewise, the semantic label `GPE`, when utilizing the *logsumexp* potential func-

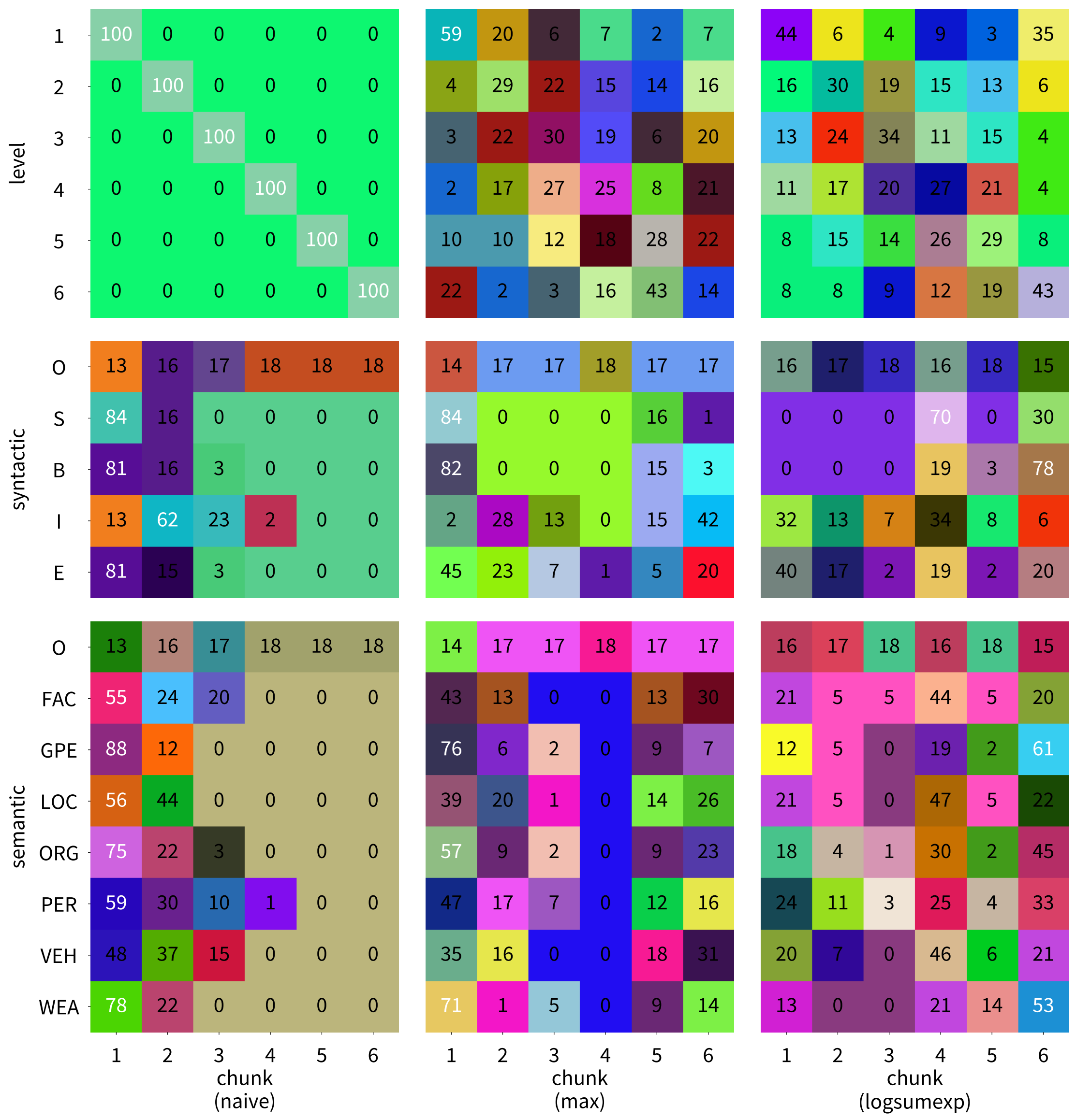


Figure 3.4: Chunk distributions of the *naive*, *max*, and *logsumexp* potential functions, respectively. Each row displays the chunk selection preferences with respect to levels, syntactic labels, and semantic labels, respectively.

tion, also has a 61% probability of selecting the sixth chunks, rather than concentrating solely on the first chunk as in the *naive* potential function. These

| Method | Batch | Training | Decoding |
|---|---|---|---|
| Wang et al. [98] | 16 | 1,937.16 | 3,626.53 |
| | 32 | 3,632.64 | 4,652.05 |
| | 64 | 6,298.85 | 5,113.85 |
| Our Method | 16 | 4,106.03 | 3,761.03 |
| | 32 | 7,219.57 | 6,893.03 |
| | 64 | 10,584.80 | 11,652.92 |

Table 3.10: Speed comparison on the dataset ACE2005. The numbers indicate the average processing speed in terms of words per second.

observations further demonstrate that our dynamic chunk selection strategies are capable of learning more meaningful representations.

### Time Complexity

The time complexity of the encoder is $\mathcal{O}(n)$. Additionally, due to our utilization of the same tree reduction acceleration technique as Rush [77], the time complexity of the CRF is reduced from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$. Consequently, the overall time complexity is $\mathcal{O}(n + m \cdot \log n)$. Given that $m$ is a constant, this time complexity can be further simplified to $\mathcal{O}(n)$.

However, even with the help of the acceleration trick, the time complexity of Wang et al. [98], i.e., $\mathcal{O}(n)$, remains lower than ours. This is because they utilize softmax to directly predict the types of each span, without relying on the CRF.

Even in this scenario, the training and inference speed of our model is significantly faster than theirs, as indicated in Table 3.10. Furthermore, when we increase the batch size to 64, the decoding speed becomes more than twice as fast as their model's. The primary reason for this improvement is that we do not require stacking LSTM to 16 layers, and since sentence lengths are generally short, the $\log n$ term in our model's time complexity does not become a performance bottleneck.

## 3.5 Conclusion

In this chapter, we introduced a straightforward and effective approach for nested NER by explicitly excluding the influence of the best path. We achieve this by selecting and removing chunks at each level to construct different potential functions. Additionally, we proposed three distinct selection strategies to select the most salient chunk from the remains. Furthermore, we observed that the innermost-first encoding scheme outperforms the conventional outermost-first encoding scheme. Our extensive experimental results validate the effectiveness and efficiency of our approach.

## 3.6 Limitations and Future Work

One limitation of our method is that the number of chunks, i.e., the maximal depth of entity nesting, needs to be set in advance as a hyper-parameter. However, in actual industrial applications, the depth of entity nesting is very variant and difficult to determine in advance. Therefore, directly applying our method presents certain challenges. In the future work, we aim to extend our method to handle arbitrary depths of entity nesting.

# Chapter 4

# Learning Bit-wise Representation

## 4.1 Introduction

Pre-trained language models [18, 49, 45, 70, 26] have already become the de-facto infrastructure of modern natural language processing. They have significantly improved performance on various tasks, and at the same time have profoundly and permanently changed the research paradigm. However, lacking interpretability still keeps them a black box to humans, the inability to explain their decision-making mechanisms hinders researchers from further improving them. Fortunately, two recently published papers, which focus on compressing and interpreting continuous representation as discrete tags from pre-trained language models, have shed some light on this issue.

On the one hand, Li and Eisner [46] propose to compress the contextual representation from pre-trained language models into discrete tags. They utilize the variational information bottleneck [87, 3] to non-linearly interpret high-dimensional continuous vectors into discrete tags, retaining only the information that aids the downstream parsing task. These obtained tags form an alternative tag set and contain necessary syntactic properties. Moreover, the mechanism of the variational information bottleneck, on which their method relies, is to maximize the mutual information between latent discrete tags and targets, while simultaneously minimizing the mutual information between inputs and latent discrete tags. In this way, only the task-relevant information remains in these tags.

On the other hand, Kitaev et al. [40] similarly collapse vectors into discrete

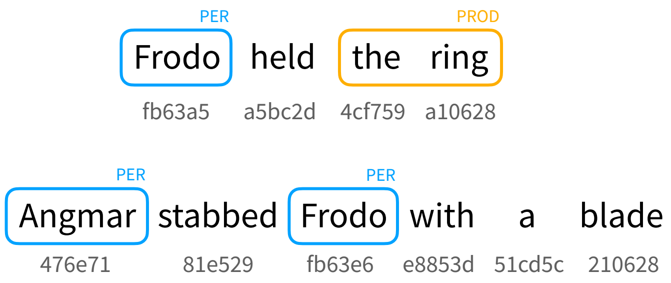


Figure 4.1: Examples of our method on the named entity recognition task. We assign each word a binary code, i.e., these hexadecimal numbers, and use them as the sole input to recognize entities. `PER` and `PROD` are the entity labels for *person* and *product*, respectively.

tags by employing a narrow bottleneck that limits the size of the discrete token vocabulary. Their approach consists of two stages. In the first stage, the contextual vectors of tokens are mapped to discrete tags via the vector quantization method [88]. At the second stage, tags are fed into a downstream model, referred to as the *read-out network* in the original paper, for downstream constituency parsing. Importantly, this read-out network has no access to the continuous vectors but only to these discrete tags, therefore, these tags are forced to encode all the needed syntactic information. Experiments show that their model achieves comparable performance with only a few bits for each word.

Different from the two methods above, we provide a novel contrastive hashing method to obtain binary codes from high-dimensional hidden states of pre-trained language models. We push the compression limit by further narrowing the information bottleneck to 24 bits. Following Kitaev et al. [40], we also introduce a stage to verify whether the information is properly preserved in these binary codes. Additionally, we train an extremely lightweight model using these binary codes as the sole inputs. Experiments show that it successfully reproduces comparable or even slightly better performance than the original full-size model.

Moreover, our method hashes vectors into bit-level binary codes, rather than using token-level tags as in the two previous works. Therefore, the compressed codes are much more interpretable and compact. More specifically, our hashing results not only indicate whether the syntactic properties of two given tokens are different, but also distinguish exactly which bits they differ in.

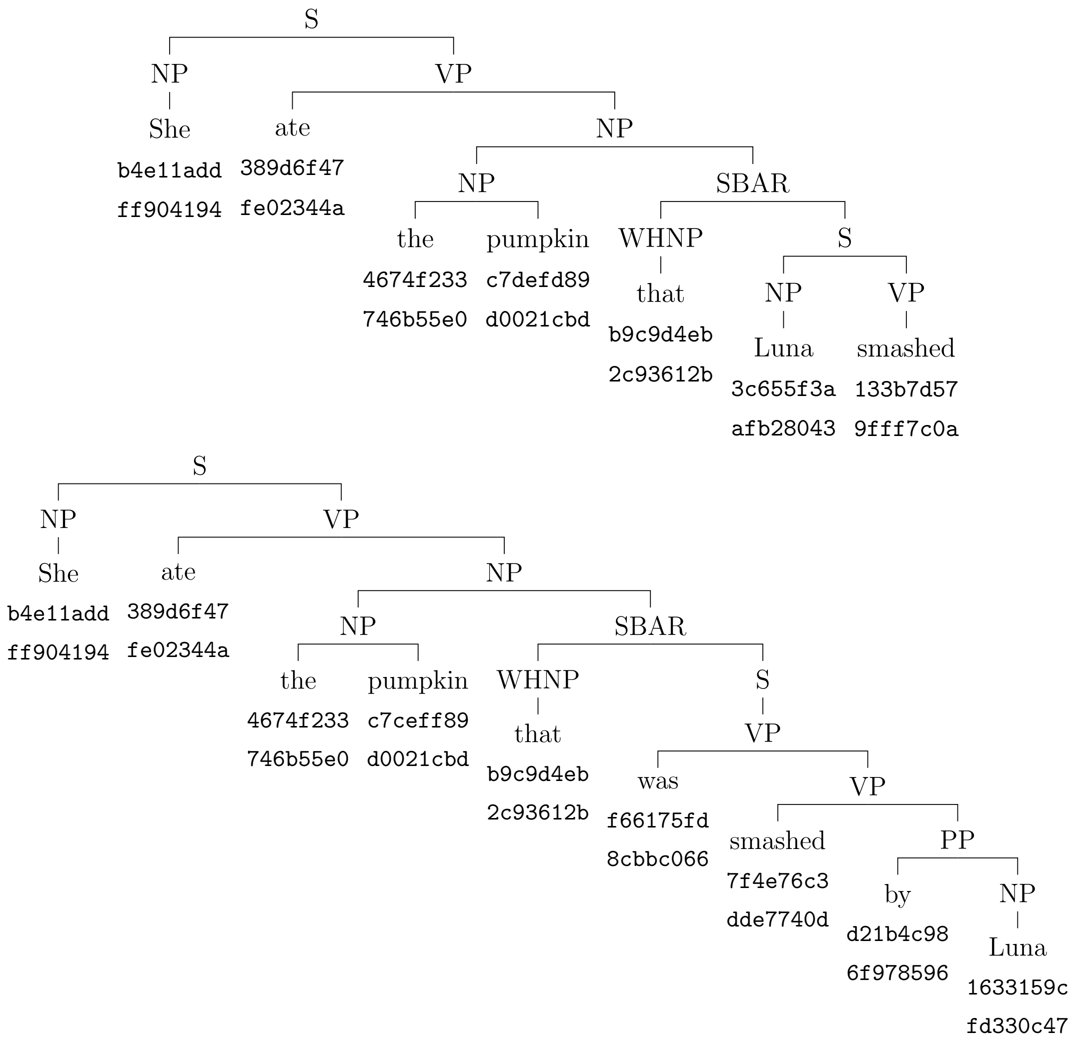


Figure 4.2: Derivation of the sentence *She ate the pumpkin that Luna smashed*, and the sentence *She ate the pumpkin that was smashed by Luna.* These hexadecimal numbers below each token are the hashing results of bidirectional and incremental parsing, respectively.

Our method builds upon contrastive hashing. We introduce a recently proposed Hamming similarity approximation [28] to combine contrastive learning with deep hashing methods. In addition, we introduce an instance selection strategy aimed at mitigating issues related to contextual false positives and false negatives. Moreover, we design a novel transformer-based hash layer, in which each attention head corresponds to a single bit. The entire model is trained to learn to hash by using both the downstream task objective and the contrastive hashing objective simultaneously. These two objectives share a portion of the attention matrix from the hash layer, ensuring that the learned binary codes are likely to properly preserve task-relevant information.

## 4.2 Proposed Methods

For many tasks, the standard approach of modern language processing is first feeding the input sentence, i.e, $w_1, \ldots, w_n$, into a pre-trained language model to assign each token a continuous vector, i.e., $\boldsymbol{x}_i \in \mathbb{R}^d$, and leveraging them in the downstream task. In this work, we aim to interpret these continuous vectors as discrete binary codes, i.e., $\boldsymbol{c}_i \in \{-1, +1\}^K$ where $K \ll d$, which contains task-relevant information as well. In this way, our method converts continuous vectors to an interpretable format, thereby making the internal mechanism more transparent and comprehensible.

Our framework consists of two stages. In the first stage, i.e., *hashing stage*, we learn to hash the continuous vectors as discrete tokens. We append a transformer-based hash layer (§4.2.3) to the end of a pre-trained language model and train the entire model to learn to hash by fine-tuning it on the downstream task. Novelly, we employ the contrastive hashing method (§4.2.1) and carefully exclude potentially false positive and negative instances with a selection strategy (§4.2.2). After training, we utilize the hash layer to re-annotate the entire dataset by assigning each token a binary code.

In the second stage, i.e., the *validation stage*, we evaluate whether these binary codes preserve task-relevant information or simply contain meaningless bits. Using these binary codes as the sole inputs, we train a much more lightweight model from scratch. Experiments show that even with such limited capability, our model still achieves comparable or even slightly better performance than the original full-size model. Therefore, we claim that our method properly preserves task-relevant information in these binary codes. The pseudo-code can be found in Algorithm 3.

### 4.2.1 Contrastive Hashing

Contrastive learning [15, 59, 13, 106, 25] has already been shown to be an effective representation learning method. Its fundamental concept involves employing an encoder network to map instances into a continuous representation, i.e., $\boldsymbol{x} \in \mathbb{R}^d$. It then pulls together the positive pairs and pushes apart the negative pairs by applying the following objective function[1].

$$\begin{aligned} \mathcal{L}_{\text{self}} &= -\log \frac{\exp s(\boldsymbol{x}, \boldsymbol{x}^+)}{\sum_{\boldsymbol{x}' \in \mathcal{X}} \exp s(\boldsymbol{x}, \boldsymbol{x}')} \\ &= \log \sum_{\boldsymbol{x}' \in \mathcal{X}} \exp s(\boldsymbol{x}, \boldsymbol{x}') \quad - s(\boldsymbol{x}, \boldsymbol{x}^+) \end{aligned}$$

where $\mathcal{X}$ is the instance batch, and $s(\boldsymbol{x}, \boldsymbol{y})$ returns the similarity between the two given instances. Contrastive learning generally expects instances uniformly distributed on a unit hyper-sphere. Therefore, the most commonly used similarity function is the cosine function,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} \tag{4.1}$$

On the other hand, deep hashing methods [12, 82, 28] also aim at mapping instances into informative representation but in discrete space, i.e., $\boldsymbol{c} \in \{-1, +1\}^K$. They first utilize an encoder network to map instances to continuous score vectors, i.e., $\boldsymbol{s} \in \mathbb{R}^K$, and then obtain binary codes by taking signs, i.e., $\boldsymbol{c} = \text{sign}\,(\boldsymbol{s})$. Besides, deep hashing methods also pull together the positive pairs by encouraging all their bits to become the same and at the same time making negatives pairs have as many as possible different bits. Commonly, this is implemented as Hamming similarity. To be more specific, for two given score vectors, $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^K$, the similarity is defined as,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{K} \text{sign}\,(\boldsymbol{x}_i) \cdot \text{sign}\,(\boldsymbol{y}_i) \tag{4.2}$$

We notice that deep hashing shares the common fundamental concept with contrastive learning, except it represents instances in a $K$-dimensional Hamming

[1]We omit the temperature $\tau$ for clarity.

space, i.e., $\{-1,+1\}^K$, instead of a unit hyper-sphere, i.e., $\mathbb{R}^{d-1}$. Therefore, we propose introducing Hamming similarity to extend the contrastive learning to learn to hash.

However, the Hamming similarity above is not differentiable, directly introducing it to the contrastive learning framework is intractable. Recently, Hoe et al. [28] proposed a novel similarity function that takes the sign of one of its inputs before computing their cosine similarity. They demonstrate that maximizing this similarity preserves semantic information as well. Therefore, we instead introduce this approach to our contrastive learning framework to learn to hash.

$$s(\boldsymbol{x},\boldsymbol{y}) = \cos(\boldsymbol{x}, \operatorname{sign}(\boldsymbol{y})) \tag{4.3}$$

### 4.2.2 Instance Selection

One of the most appealing properties of contrastive learning is that it successfully converts tasks from *wh-questions* to *yes-no questions*. Conventional classification requires specifying target labels for all instances, but contrastive learning only demands knowing whether two instances are identical or not.

Due to this benefit, effective representation learning becomes possible even in unsupervised settings. Gao et al. [24] pass instances into a neural network twice to obtain two semantically identical but slightly augmented representations, i.e., $\boldsymbol{x}$ and $\boldsymbol{x}^+$, relying on the independently sampled dropout masks [79]. They employ the objective $\mathcal{L}_{\text{self}}$ to perform representation learning, treat these two views as positive to each other, and consider all existing instances in the batch as negatives. This simple method surprisingly works well and results in expressive representation.

Furthermore, in supervised settings, Khosla et al. [34] proposed leveraging label information by introducing an objective function capable of handling cases with multiple positive instances.

$$\begin{aligned}\mathcal{L}_{\text{sup}} =& \frac{-1}{|\mathcal{X}^+|}\sum_{\boldsymbol{x}^+\in\mathcal{X}^+}\log\frac{\exp s(\boldsymbol{x},\boldsymbol{x}^+)}{\sum_{\boldsymbol{x}'\in\mathcal{X}}\exp s(\boldsymbol{x},\boldsymbol{x}')}\\ =& \log\sum_{\boldsymbol{x}'\in\mathcal{X}}\exp s(\boldsymbol{x},\boldsymbol{x}')\\ & -\frac{1}{|\mathcal{X}^+|}\sum_{\boldsymbol{x}^+\in\mathcal{X}^+} s(\boldsymbol{x},\boldsymbol{x}^+)\end{aligned}$$

where the $\mathcal{X}^+$ is the set of positive instances. Obviously, the first term of $\mathcal{L}_{\text{sup}}$ and $\mathcal{L}_{\text{self}}$ are identical. The difference between them is, $\mathcal{L}_{\text{self}}$ pulls together only one positive while $\mathcal{L}_{\text{sup}}$ leverages all of these positive instances, i.e., the second term .

However, we observe that tokens are assigned different information in varying contexts, making it challenging to determine whether two identical tokens truly form a positive pair. For example, in Figure 4.3, the token *Frodo* appears in both sentences. It serves as the subject in the first sentence and as the object in the second, resulting in dissimilar parses. Therefore, identical tokens may contain distinct task-relevant information and, in such cases, deserve different binary codes.

Since it is difficult to determine whether two identical tokens contain identical task-relevant information in practice, we choose to not include them in either the positive or the negative set. For the numerator part of the objective function, we remove all identical token pairs and retain only the augmented version of each token as the sole positive instance, thereby reverting to the single positive instance scenario. For the denominator part, we also remove all identical tokens from $\mathcal{X}$ to exclude potential false negatives.

$$\mathcal{L}_{\text{hash}} = -\log\frac{\exp s(\boldsymbol{x},\boldsymbol{x}^+)}{\sum_{\boldsymbol{x}'\in\{\boldsymbol{x}^+\}\cup\mathcal{X}^-}\exp s(\boldsymbol{x},\boldsymbol{x}')} \tag{4.4}$$

Where $\mathcal{X}^-$ only contains tokens that are different from $\boldsymbol{x}$. More specifically, as shown in Figure 4.3, we consider the second *Frodo* as neither a positive nor a negative instance to the first *Frodo*, and remove it from both the numerator and the denominator.

**Algorithm 3:** PyTorch-like style pseudocode.

```
def compute_hash_loss(x, y, tokens):
    # Equation 4.3
    score = cos(x[:, None], y[None, :].sign(), dim=-1)
    score = score / tau # [tok, tok]

    # excludes potentially false positives and negatives
    mask = tokens[:, None] == tokens[None, :] # [tok, tok]
    score[mask ^ eye] = -float('inf')

    # Equation 4.4
    return (score.logsumexp(dim=-1) - score.diag()).mean()

def hashing_step(plm, task_model, inputs, targets):
    h1, s1 = plm(inputs) # [bsz, snt, dim], [bsz, snt, K]
    h2, s2 = plm(inputs) # [bsz, snt, dim], [bsz, snt, K]

    task_loss1 = compute_task_loss(task_model(h1), targets)
    task_loss2 = compute_task_loss(task_model(h2), targets)
    task_loss = task_loss1 + task_loss2

    s1 = flatten(s1) # [tok, K]
    s2 = flatten(s2) # [tok, K]
    tokens = flatten(inputs) # [tok]

    hash_loss1 = compute_hash_loss(s1, s2, tokens)
    hash_loss2 = compute_hash_loss(s2, s1, tokens)
    hash_loss = hash_loss1 + hash_loss2

    # Equation 4.10
    return task_loss + beta * hash_loss

def reannotate(plm, dataset):
    new_dataset = []

    for inputs in dataset:
        _, s = plm(inputs) # [bsz, snt, k]
        codes = s.sign() # [bsz, snt, k]
        new_dataset.extend(codes)

    return new_dataset

def validation_step(lite_task_model, codes, targets):
    logits = lite_task_model(codes)
    task_loss = compute_task_loss(logits, targets)

    return task_loss
```

The pseudo-code of this objective function can be found in the `compute_hash_loss` of Algorithm 3.

Before introducing our transformer-based hashing layer, we briefly review the mechanism of multi-head attention [90]. The attention layer first projects the input vectors into queries, keys, and values. It then constructs output vectors by aggregating desired information from these key-value pairs.

$$s_{i,j}^h = \frac{(\mathbf{W}_q^h \boldsymbol{x}_i)^\top (\mathbf{W}_k^h \boldsymbol{x}_j)}{\sqrt{d_h}} \tag{4.5}$$

$$a_{i,j}^h = \text{softmax}_j\left(s_{i,j}^h\right) \tag{4.6}$$

$$\boldsymbol{z}_i^h = \sum_j a_{i,j}^h (\mathbf{W}_v^h \boldsymbol{x}_j) \tag{4.7}$$

$$\boldsymbol{o}_i = \mathbf{W}_o \left[\boldsymbol{z}_i^1, \ldots, \boldsymbol{z}_i^H\right] \tag{4.8}$$

where $\mathbf{W}_q^h, \mathbf{W}_k^h, \mathbf{W}_v^h \in \mathbb{R}^{d_h \times d}$ are the projection weights of query, key, and value of the $h$-th head, respectively. The $\mathbf{W}_o \in \mathbb{R}^{d \times (H \times d_h)}$ is the output weight, $d, d_h, H$ are the input dimension, head dimension, and the number of heads, respectively. $[\cdot, \ldots, \cdot]$ indicate concatenation and bias terms are omitted for clarity. These hidden states $\boldsymbol{o}_i$ are then fed into a feed-forward network to obtain the output vectors $\boldsymbol{h}_i = \text{FFN}(\boldsymbol{o}_i) \in \mathbb{R}^d$ for downstream tasks. Conventionally, the head size $d_h$ is simply bounded to $d$ and $H$, but we let the $d_h$ become an independent hyper-parameter, therefore, $d$ does not have to equal to $d_h \times H$ in our implementation.

### 4.2.3 Transformer Hash Layer

Intuitively speaking, the mechanism of attention is to selectively aggregate information from tokens. The attention score $s_{i,j} \in \mathbb{R}$ estimates the amount of desired information that token $i$ may obtain from token $j$. Specifically, $s_{i,i}$ estimates how much desired information is retained in token $i$ itself. Furthermore, by increasing the number of heads to $K$, the vector $\boldsymbol{s}_{i,i} \in \mathbb{R}^K$ represents the desired information amount of token $i$ from $K$ different aspects, and can naturally produce $K$ bits by taking their signs.

Therefore, inspired by the multi-head mechanism, we propose to leverage $K$ heads to produce $K$ bits. We add an additional transformer layer with its number of heads increased to $K$, and use the diagonal entries $\boldsymbol{s}_{i,i}$ of its attention matrix

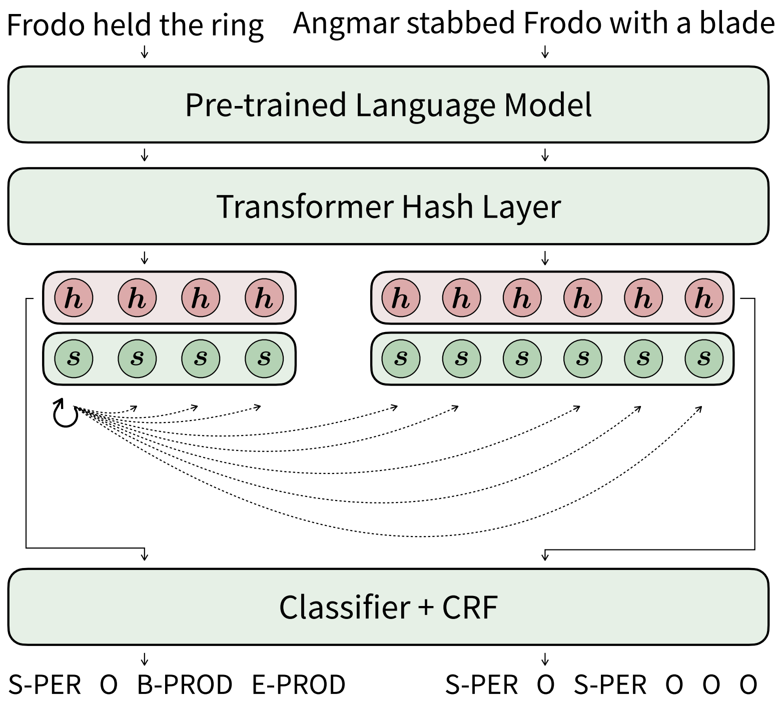


Figure 4.3: The architecture of the hashing stage model for named entity recognition. The transformer hash layer (§4.2.3) produces both contextual representation $\boldsymbol{h}$ and ego-attention scores $\boldsymbol{s}$ (§4.2.3) for the task-specific fine-tuning and contrastive hashing (§4.2.1), respectively. Solid lines indicate the positive instance, while dotted lines show negatives. Note that the token *Frodo* appears twice in different sentences, thus, to avoid including false positives and false negatives (§4.2.2), there is no arrow pointing from the first *Frodo* to the second one.

as the hashing scores to learning to hash, and take their signs to generate binary codes as the hashing results after training, i.e., $\boldsymbol{c}_i = \operatorname{sign}(\boldsymbol{s}_{i,i})$. Since $\boldsymbol{s}_{i,i}$ represents a form of attention that one attends to oneself, to distinguish it from the commonly known term *self-attention*, We use the term *ego-attention* to refer to it throughout the remainder of this dissertation.

In summary, the full attention matrix $\boldsymbol{s}_{i,j}$ is utilized in a dual manner: it not only serves the purpose in the conventional Transformer architecture for computing the output vectors, but also lends its diagonal entries $\boldsymbol{s}_{i,i}$ to learn to hash. Given that a portion of the attention matrix is shared between these two objectives, the learned binary codes are inclined to preserve task-relevant information. This hypothesis is demonstrated by our experimental results in the validation stage.

### 4.2.4 Hashing Stage Architecture

The architecture of the hashing stage model, as shown in Figure 4.3, consists of one pre-trained language model, one transformer-based hash layer, and the task-specific layers. We initialize RoBERTa [49] with the checkpoint `roberta-base` as the pre-trained language model.

**Part-of-speech Tagging** We employ a two-layered classifier and a Conditional Random Field [42] (CRF) to compute the log-likelihood and utilize the Viterbi algorithm [23] for inference.

**Named Entity Recognition** We transform the sequence of vectors from the sub-token level back to the token level by taking the average of the sub-token vectors of each individual token. We use the same task-specific layers as part-of-speech tagging.

**Constituency Parsing** Similarly, we generate the token-level representation by averaging the vectors of sub-tokens. In addition, following Zhang et al. [107], we use a biaffine span classifier along with a tree-structured CRF. We identify the most probable tree from all valid trees using the Cocke–Kasami–Younger Algorithm [33] (CKY). Following Kitaev et al. [40], we also incorporate GPT-2 [70] using the `gpt2-medium` checkpoint for incremental parsing.

### 4.2.5 Validation Stage Architecture

As mentioned above, this stage is only to validate if the task-relevant information has been properly preserved in these binary codes, and is not to distill knowledge into a lightweight model. In this stage, we introduce an extremely lightweight model to ensure that the model lacks the capacity to learn the tasks from scratch.

As such, any performance gains are solely owed to the information already preserved within the binary inputs. The architecture for this validation stage comprises a binary code embedding layer, a conventional transformer layer, and the same task-specific layers used during the hashing stage.

The binary code embedding layer produces code embeddings through constructing instead of looking up. For a given binary code, $\boldsymbol{c} \in \{-1, +1\}^K$, the binary code embedding layer simply flips the direction of each bit embedding $\boldsymbol{b}_i$, and returns the concatenation of these flipped vectors, where $\boldsymbol{b}_i \in \mathbb{R}^{d/K}$ is the embedding of the $i$-th bit.

$$\boldsymbol{w} = [c_1\boldsymbol{b}_1, \ldots, c_K\boldsymbol{b}_K] \in \mathbb{R}^d \tag{4.9}$$

Compared with the learned discrete tags of Kitaev et al. [40], our binary codes literally encode information at the bit level, while their tags remain at the token level. Thus, although Kitaev et al. [40] emphasize that their model requires only $K$ bits per word, in practice, their model demands an embedding matrix with shape $2^K \times d$, while our real bit-level embedding needs only $K \times \dfrac{d}{K}$.

### 4.2.6 Training and Inference

In the hashing stage, we balance the task-specific loss $\mathcal{L}_{\text{task}}$ and the hashing loss $\mathcal{L}_{\text{hash}}$, as the `hashing_step` function in Algorithm 3. Besides, our training procedure is also simpler than Kitaev et al. [40], since we don't need to employ the $k$-mean algorithm [1] to initialize the centroids in the first two epochs.

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \beta \cdot \mathcal{L}_{\text{hash}} \tag{4.10}$$

In the validation stage, we re-annotate the entire dataset first and then use the task-specific loss $\mathcal{L}_{\text{task}}$ only to train the lightweight model with only these binary codes as inputs. The procedures for `reannotate` and `validation_step` are described in Algorithm 3, respectively.

| Dataset | Train | Dev | Test | Label |
|---|---|---|---|---|
| POS | 39,832 | 1,700 | 2,416 | 45 |
| NER | 59,924 | 8,528 | 8,262 | 73 |
| Parsing | 39,832 | 1,700 | 2,416 | 142 |

Table 4.1: Statistics on the WSJ and OntoNotes datasets for the three tasks.

## 4.3 Experiments

### 4.3.1 Datasets

**Part-of-speech Tagging** We conduct experiments on the English Penn Treebank [55] dataset. The task involves assigning a syntactic label to each token in a given sentence. We report the accuracy scores on the test split.

**Named Entity Recognition** The OntoNotes English dataset [67] is used for evaluation. We transform span annotations into the BIOES encoding scheme [71], and report the $F_1$ scores on the test split.

**Constituency Parsing** We evaluate on the English Penn Treebank [55]. Following Zhang et al. [107] and Kitaev et al. [40], we transform the original tree into those of Chomsky normal form and adopt left binarization with `NLTK` [9]. We report the $F_1$ scores on the WSJ test split.

### 4.3.2 Settings

We implement our models with the deep learning framework `PyTorch` [64] and fetch weights of pre-trained language model from `huggingface/tramsformers` [103].

For each batch, we keep collating sentences until the total number of tokens reaches 1024. The reason that we don't use the number of sentences as batch size is to stabilize contrastive learning, since it is performed at token-level, not sentence-level. We employ AdamW [37, 50] with 50,000 training steps and 6%

| Hyper-param | POS | NER | Parsing |
|---|---|---|---|
| $\beta$ | 0.05 | 0.005 | 0.001 |
| $\tau$ | 0.1 | 0.1 | 0.1 |
| Dropout | 0.1 | 0.1 | 0.5 |
| Learning Rate | 5e-5 | 7e-5 | 5e-5 |
| Dropout | 0.1 | 0.2 | 0.3 |
| Learning Rate | 5e-4 | 3e-3 | 1e-3 |

Table 4.2: Hyper-parameters on all tasks. The first block shows the hyper-parameters on hashing stage, while the second one shows the validation stage.

warm-up steps. In the hashing stage, we evaluate the performance with different number of bits, specifically $K \in \{16, 24, 32\}$.

We run experiments on a single NVIDIA Tesla V100 graphics card. The hashing stage takes approximately 2 hours, while the validation stage requires only around 30 minutes. We run the experiments four times with different random seeds. The reported numbers in the following tables are their averages.

For comparison, we also conduct baseline experiments for each task without using the contrastive hashing loss, i.e., $\beta = 0$. These baseline experiments utilize only the task-specific loss. Comparing these results with the hashing-stage performance illustrates the impact of contrastive hashing on training, allowing us to determine whether the contrastive hashing itself negatively affects model performance. Additionally, comparing them with the validation stage performance reveals how well task-relevant information is preserved.

### 4.3.3 Main Results

As presented in Table 4.3 and Table 4.4, experiments on the part-of-speech tagging show that 32 bits achieve slightly better results than 16 bits and 24 bits on both stages. Besides, we notice that results in the validation stage are constantly superior to hashing stage results, no matter how many bits are used.

For named entity recognition, we achieve 90.39 in $F_1$ score with 24 bits, which is

| Model | POS RoBERTa Acc | POS RoBERTa $\lvert\theta\rvert$ | NER RoBERTa $F_1$ | NER RoBERTa $\lvert\theta\rvert$ |
|---|---|---|---|---|
| Baseline | 98.27 | 134.2M | 90.24 | 134.2M |
| 16 bits | 98.37 | 132.6M | 90.21 | 132.6M |
| | 98.38 | 0.6M | 90.28 | 0.6M |
| 24 bits | 98.39 | 134.2M | 90.27 | 134.2M |
| | 98.38 | 0.6M | **90.39** | 0.6M |
| 32 bits | 98.40 | 135.7M | 90.12 | 135.7M |
| | **98.41** | 0.6M | 90.31 | 0.6M |

Table 4.3: The main results on part-of-speech and named entity recognition experiments. The results of our methods are displayed in two rows, which indicate the performance in hashing and validation stages, respectively. $|\theta|$ columns show the number of parameters, and the bold numbers indicate **the best validation performance** in the settings.

even slightly higher than its hashing stage performance, i.e., 90.27. For 16 bits and 32 bits, the validation stage performance also consistently surpasses their hashing stage performance. We hypothesize that this is because hashing the ego-attention scores may implicitly exclude some unconfident attention scores that might lead to wrong predictions. For example, consider a token that barely contains the desired information of a query, it should be ignored by getting a small attention score. However, if the network unconfidently assigns it an attention score that is only slightly less than 0, then its information still occupies a certain proportion in the final output. On the contrary, our method truncates the attention scores to be $-1$ or $+1$, and eases the issue in some degree.

For constituency parsing, our method outperforms Kitaev et al. [40] with 32, 40, and 48 bits in the bidirectional parsing setting, even they introduce much more tags, i.e., 256 in total. Besides, our 16 bits and 24 bits settings also achieve remarkable performance and are only slightly inferior to theirs. In this task, all experiments in the validation stage show worse results than the corresponding

| Model | Parsing RoBERTa | | Parsing GPT2 | |
|---|---|---|---|---|
| | $F_1$ | $\|\theta\|$ | $F_1$ | $\|\theta\|$ |
| Kitaev et al. [38] | 95.59 | 342.8M | 93.95 | 362.5M |
| Kitaev et al. [40] | 95.55 | 361.4M | 94.97 | 381.1M |
| Baseline | 95.92 | 136.0M | 95.04 | 422.5M |
| 16 bits | 96.00 | 134.4M | 95.02 | 420.4M |
| | 95.24 | 2.9M | 93.76 | 5.3M |
| 24 bits | 95.92 | 136.0M | 95.14 | 422.5M |
| | 95.51 | 2.9M | 93.82 | 5.3M |
| 32 bits | 95.97 | 137.6M | 95.15 | 424.6M |
| | 95.65 | 2.9M | **94.02** | 5.3M |
| 40 bits | 96.09 | 139.2M | 95.14 | 426.7M |
| | **95.75** | 2.9M | 93.98 | 5.3M |
| 48 bits | 96.11 | 140.8M | 95.07 | 428.8M |
| | 95.63 | 2.9M | 94.01 | 5.3M |

Table 4.4: The main results on bidirectional and incremental parsing experiments. The results of our methods are displayed in two rows, which indicate the performance in hashing and validation stages, respectively. $|\theta|$ columns show the number of parameters, and the bold numbers indicate **the best validation performance** in the settings.

| $s(\boldsymbol{x}, \boldsymbol{y})$ | $\mathcal{L}_{\text{contrastive}}$ | NER |
|---|---|---|
| $\cos(\boldsymbol{x}, \boldsymbol{y})$ | $\mathcal{L}_{\text{self}}$ | 90.12 $\rightarrow$ 88.74 |
| | $\mathcal{L}_{\text{sup}}$ | 90.07 $\rightarrow$ 86.91 |
| | $\mathcal{L}_{\text{hash}}$ | 90.19 $\rightarrow$ 88.94 |
| $\cos(\boldsymbol{x}, \operatorname{sign}(\boldsymbol{y}))$ | $\mathcal{L}_{\text{self}}$ | 90.15 $\rightarrow$ 90.21 |
| | $\mathcal{L}_{\text{sup}}$ | 90.19 $\rightarrow$ 90.04 |
| | $\mathcal{L}_{\text{hash}}$ | 90.27 $\rightarrow$ **90.39** |

Table 4.5: Ablation study on the different similarity functions and objective functions on the OntoNotes dataset. The numbers on the left and right sides of $\rightarrow$ represent the hashing and validation performance, respectively.

hashing stage results. We hypothesize that this is because constituency parsing is a span-level classification task, token-level hashing is unable to capture the span information completely. This may also be the reason that our method works best on named entity recognition since it is just at the token level.

We notice that hashing stages always yield results similar to the corresponding baselines, therefore we claim that contrastive hashing loss does not have a negative impact on the training itself. Introducing the contrastive hashing loss does not significantly change the model performance, but only produces compact and interpretable binary codes. Besides, validation stages also achieve comparable performance to baselines, this fact further shows our method has not affect on the model inherent capabilities.

For all these tasks, our codes still reproduce comparable or even slightly better performance than the original full-size model, even with such a lightweight model in validation stages. We claim that these results demonstrate that our learned binary codes have properly preserved task-relevant information.

| $\beta$ | 0 | 0.001 | 0.005 | 0.01 | 0.05 |
|---|---|---|---|---|---|
| NER | 90.24 | 90.25 | 90.27 | 90.10 | 90.02 |
| | 79.60 | 90.29 | **90.39** | 90.24 | 90.23 |

Table 4.6: Ablation study on the named entity recognition experiments, along with the $\beta$ coefficient. The two rows display hashing and validation performance, respectively.

### 4.3.4 Ablation Studies

#### Influence of the Similarity and Objective Functions

Table 4.5 shows that the similarity and objective functions are essential to our method. Using the cosine similarity, the model shows relatively high performance in the hashing stage, however, the naive cosine similarity can not preserve information properly, as its performance dramatically drops in validation stage. Furthermore, the fact that $\mathcal{L}_{\text{hash}}$ consistently outperforms both $\mathcal{L}_{\text{sup}}$ and $\mathcal{L}_{\text{self}}$ demonstrates our hypothesis that false positives and false negatives are harmful.

#### Influence of the $\beta$ Coefficient

Additionally, as indicated in Equation 4.10, the coefficient $\beta$ serves to balance the two terms. According to Table 4.6, even though the contrastive hashing loss requires only a minor proportion of the overall loss, demonstrated by the optimal performance of a small $\beta = 0.005$, it is also critical for preserving information. Experiments reveal that removing the contrastive hashing loss, i.e., $\beta = 0$, results in a dramatic performance drop.

#### Influence of the Temperature $\tau$

Table 4.7 shows that the temperature also has an influence on the hashing performance. Generally, it controls the strength of penalties on hard negative instances. We empirically found the sweet spot is around 0.1.

| $\tau$ | 0.01 | 0.02 | 0.05 | 0.1 | 0.2 |
|---|---|---|---|---|---|
| NER | 90.19 | 90.08 | 90.02 | 90.27 | 90.13 |
| | 89.13 | 89.12 | 89.81 | **90.39** | 90.16 |

Table 4.7: Ablation study on the OntoNotes experiments, along with the temperature $\tau$, which controls the strength of penalties on hard negative instances [97].

### 4.3.5 Case Studies

We present the hashing and constituency parsing results to demonstrate the interpretability of our learned binary codes. For comparison with Kitaev et al. [40], we use the exact same examples as in their paper.

We begin by discussing bidirectional parsing. In our transformer-based hash layer, each head corresponds to a single bit, and these heads operate independently of one another. This design allows each bit to capture distinct and orthogonal syntactic and semantic properties. Notably, we observe that the generated binary codes cluster based on the part-of-speech properties. For example, the past tense verbs *brought* and *approved* receive similar codes even when they appear in different sentences, differing by only four bits. Similarly, the common nouns *groceries* and *proposal* share 28 bits, highlighting their shared noun properties.

Moreover, since both of these nouns finalize a similar noun phrase, the preposition *the* before them is assigned the same code. However, the preposition *the* before the *council* retains quite different bits. We hypothesize these bits indicate the varied phrase structures. Besides, for the left side two sentences, the final attachments *him* and *himself* determine the attachment location of the *for* phrase. We observe that only 2 bits differ between them, however, finalizing these phrases impacts the structures of previous phrases, leading to corresponding changes in attachment locations. Thus, the subjective *Lucas* and the predicate verb *brought* also flip one bit, respectively, to reflect the different phrase structures. Similarly, for the right side sentences, *Monday* and *taxes* differ in 5 bits, and the attachment locations of all the phrases that depend on this phrase are influenced, thus, *approved*, *the*, and *proposal* alters their bits as well.

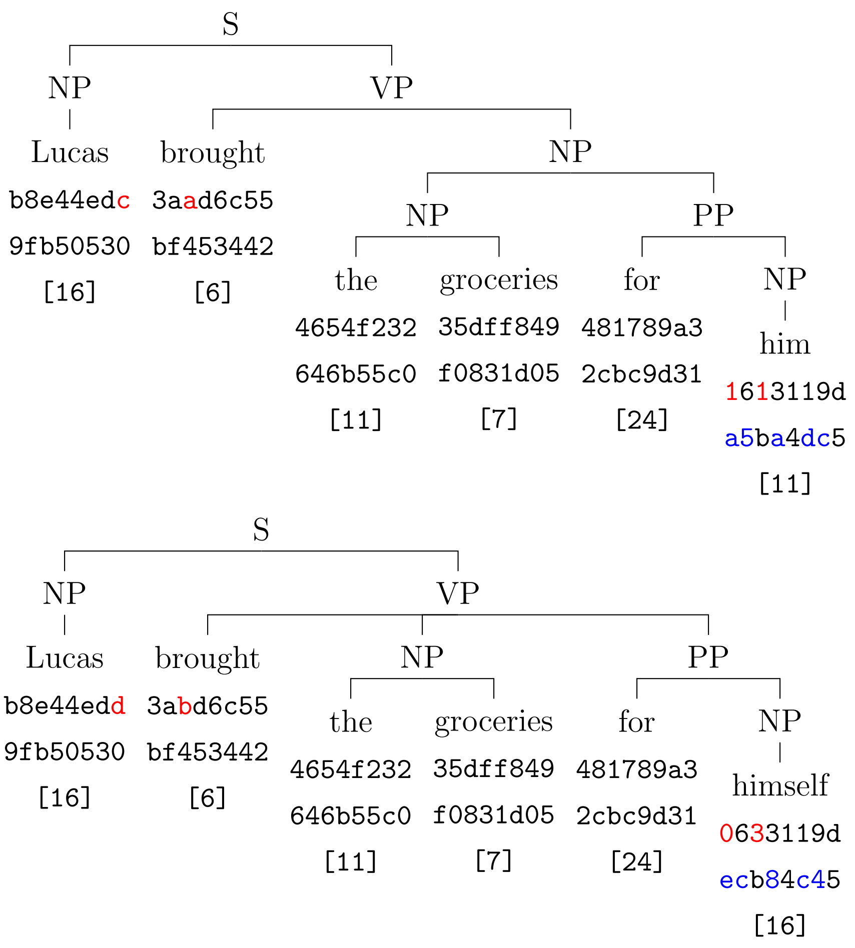


Figure 4.4: Examples of the hashing and constituency parsing results. There are three numbers below each token, the first two are represented in hexadecimal (32 bits), and indicate the hashing results of the bidirectional (RoBERTa) and unidirectional (GPT2) pre-trained language models, respectively. The third number is taken from Kitaev et al. [40] for comparison and is represented in decimal. The red and blue parts indicate the exact different bits.

Apart from that, incremental parsing disallows the information from future tokens, and the future tokens potentially contain syntactic properties that is needed for committing parsing decisions. Therefore, compressed codes should not only retain the already revealed information but also be open to all possible

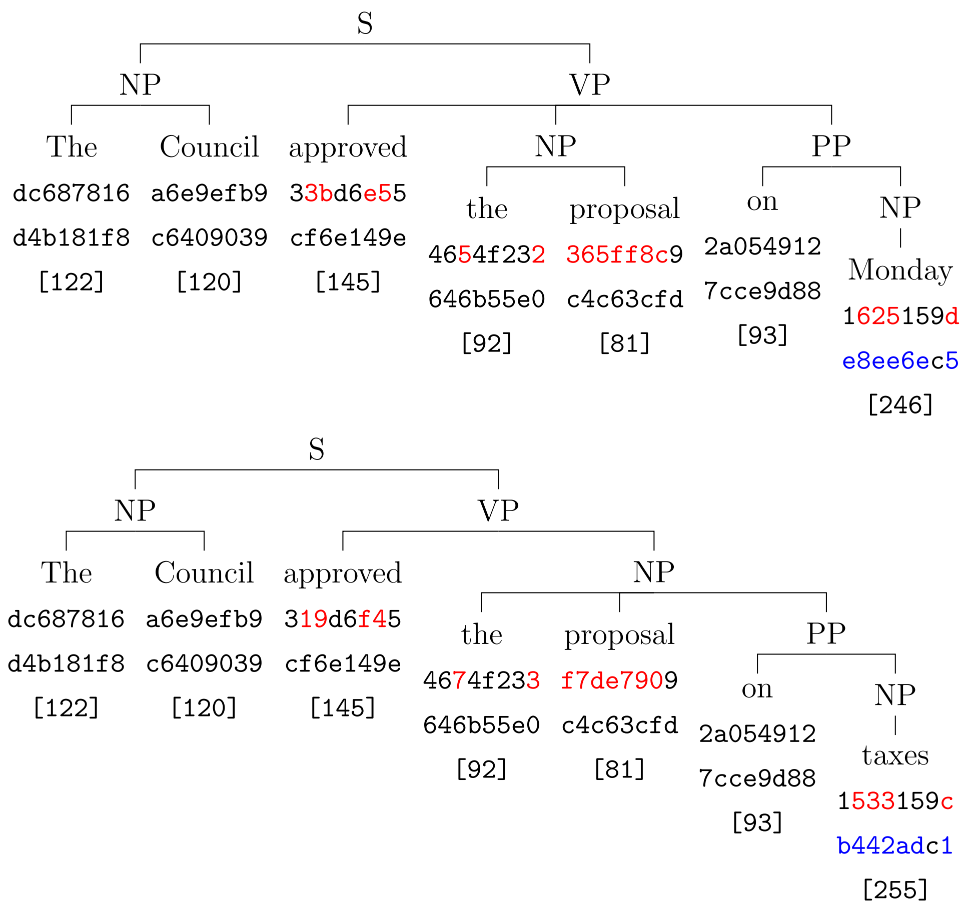


Figure 4.5: Examples of the hashing and constituency parsing results. There are three numbers below each token, the first two are represented in hexadecimal (32 bits), and indicate the hashing results of the bidirectional (RoBERTa) and unidirectional (GPT2) pre-trained language models, respectively. The third number is taken from Kitaev et al. [40] for comparison and is represented in decimal. The red and blue parts indicate the exact different bits.

upcoming tokens, as called *speculation free* in Kitaev et al. [40]. Therefore, needed information is mostly concentrated in the last tokens, and thus they are likely to obtain varied codes reflecting varied phrases. For example, on the left side, the last noun tokens *him* and *himself* obtain quite different codes, 5 bits different in total, more than the 2 bits in the bidirectional parsing case above. Besides,

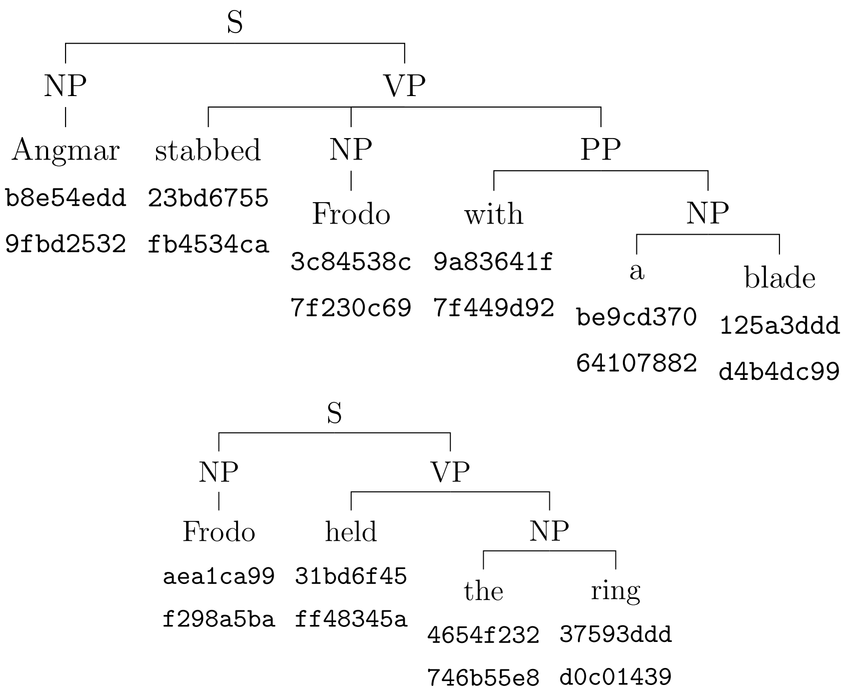


Figure 4.6: Derivation of the sentence *Angmar stabbed Frodo with a blade*, and the sentence *Frodo held the ring.*

incremental parsing model also commits similar bits for the prepositions in *the groceries* and *the proposal*, i.e., only 1 different bit, but assigns a much different code to *the council*, which has 15 nonidentical bits. By comparison, even Kitaev et al. [40] also assign them distinct tags, e.g., 11, 92, and 122, but it is hard for them to tell how different they are and where the differences lie exactly. Thus, we claim that our binary codes are much more informative and interpretable.

### 4.3.6 Discussions

#### Bit Distribution

To further demonstrate interpretability by analyzing the specific information preserved by each bit, we display the bit distribution in Figure 4.8.

The sub-figure above illustrates the distribution of bits related to different syntactic information, which serves to indicate the boundary of each entity. It

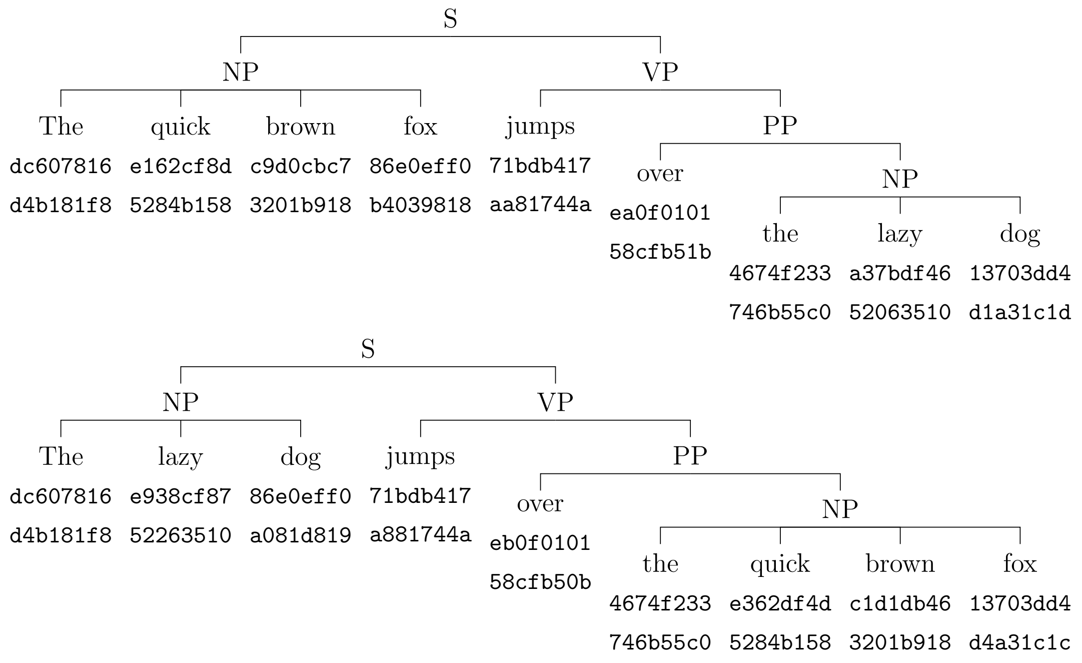
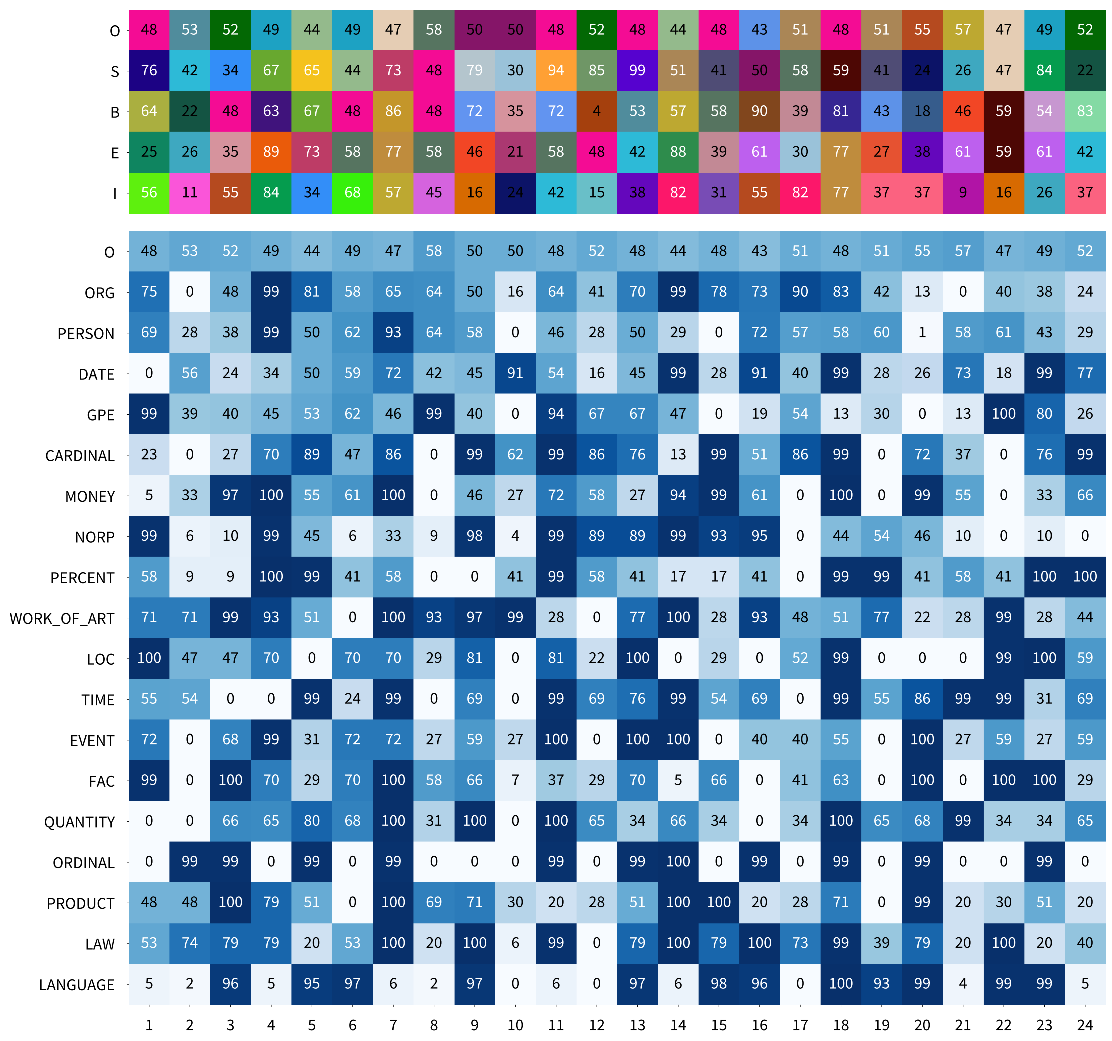


Figure 4.7: Derivation of the sentence *The quick brown fox jumps over the lazy dog*, and the sentence *The lazy dog jumps over the quick brown fox.*

is noteworthy that the bit distributions for the non-entity label `O` are uniform, such that in all these positions the probability of being assigned a 1 is roughly around 50%. In contrast, the distribution of bits for other labels exhibits a clear bias. For instance, examining the 9th bit, we hypothesize that the reason `S` and `B` have 80% and 73% probabilities of being assigned a 1, while the numbers are only 47% and 17% for `E` and `I`, is that both `S` and `B` can indicate the beginning of an entity, but such property is not shared by the other labels.

The sub-figure below reveals more distinct distributional features. Although the non-entity label `O` continues to display uniform distribution characteristics, labels `MONEY`, `NORP`, and `PERCENT` show that the probabilities at the 4th and 17th bits are skewed to 100% and 0%, respectively. Such a pronounced tendency, low entropy in other words, suggests that task-relevant information is clearly and deterministically preserved within these bits, each carrying a distinct meaning.

From the discussion above, we can see that the meaning of these induced



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O | 48 | 53 | 52 | 49 | 44 | 49 | 47 | 58 | 50 | 50 | 48 | 52 | 48 | 44 | 48 | 43 | 51 | 48 | 51 | 55 | 57 | 47 | 49 | 52 |
| S | 76 | 42 | 34 | 67 | 65 | 44 | 73 | 48 | 79 | 30 | 94 | 85 | 99 | 51 | 41 | 50 | 58 | 59 | 41 | 24 | 26 | 47 | 84 | 22 |
| B | 64 | 22 | 48 | 63 | 67 | 48 | 86 | 48 | 72 | 35 | 72 | 4 | 53 | 57 | 58 | 90 | 39 | 81 | 43 | 18 | 46 | 59 | 54 | 83 |
| E | 25 | 26 | 35 | 89 | 73 | 58 | 77 | 58 | 46 | 21 | 58 | 48 | 42 | 88 | 39 | 61 | 30 | 77 | 27 | 38 | 61 | 59 | 61 | 42 |
| I | 56 | 11 | 55 | 84 | 34 | 68 | 57 | 45 | 16 | 24 | 42 | 15 | 38 | 82 | 31 | 55 | 82 | 77 | 37 | 37 | 9 | 16 | 26 | 37 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O | 48 | 53 | 52 | 49 | 44 | 49 | 47 | 58 | 50 | 50 | 48 | 52 | 48 | 44 | 48 | 43 | 51 | 48 | 51 | 55 | 57 | 47 | 49 | 52 |
| ORG | 75 | 0 | 48 | 99 | 81 | 58 | 65 | 64 | 50 | 16 | 64 | 41 | 70 | 99 | 78 | 73 | 90 | 83 | 42 | 13 | 0 | 40 | 38 | 24 |
| PERSON | 69 | 28 | 38 | 99 | 50 | 62 | 93 | 64 | 58 | 0 | 46 | 28 | 50 | 29 | 0 | 72 | 57 | 58 | 60 | 1 | 58 | 61 | 43 | 29 |
| DATE | 0 | 56 | 24 | 34 | 50 | 59 | 72 | 42 | 45 | 91 | 54 | 16 | 45 | 99 | 28 | 91 | 40 | 99 | 28 | 26 | 73 | 18 | 99 | 77 |
| GPE | 99 | 39 | 40 | 45 | 53 | 62 | 46 | 99 | 40 | 0 | 94 | 67 | 67 | 47 | 0 | 19 | 54 | 13 | 30 | 0 | 13 | 100 | 80 | 26 |
| CARDINAL | 23 | 0 | 27 | 70 | 89 | 47 | 86 | 0 | 99 | 62 | 99 | 86 | 76 | 13 | 99 | 51 | 86 | 99 | 0 | 72 | 37 | 0 | 76 | 99 |
| MONEY | 5 | 33 | 97 | 100 | 55 | 61 | 100 | 0 | 46 | 27 | 72 | 58 | 27 | 94 | 99 | 61 | 0 | 100 | 0 | 99 | 55 | 0 | 33 | 66 |
| NORP | 99 | 6 | 10 | 99 | 45 | 6 | 33 | 9 | 98 | 4 | 99 | 89 | 89 | 99 | 93 | 95 | 0 | 44 | 54 | 46 | 10 | 0 | 10 | 0 |
| PERCENT | 58 | 9 | 9 | 100 | 99 | 41 | 58 | 0 | 0 | 41 | 99 | 58 | 41 | 17 | 17 | 41 | 0 | 99 | 99 | 41 | 58 | 41 | 100 | 100 |
| WORK_OF_ART | 71 | 71 | 99 | 93 | 51 | 0 | 100 | 93 | 97 | 99 | 28 | 0 | 77 | 100 | 28 | 93 | 48 | 51 | 77 | 22 | 28 | 99 | 28 | 44 |
| LOC | 100 | 47 | 47 | 70 | 0 | 70 | 70 | 29 | 81 | 0 | 81 | 22 | 100 | 0 | 29 | 0 | 52 | 99 | 0 | 0 | 0 | 99 | 100 | 59 |
| TIME | 55 | 54 | 0 | 0 | 99 | 24 | 99 | 0 | 69 | 0 | 99 | 69 | 76 | 99 | 54 | 69 | 0 | 99 | 55 | 86 | 99 | 99 | 31 | 69 |
| EVENT | 72 | 0 | 68 | 99 | 31 | 72 | 72 | 27 | 59 | 27 | 100 | 0 | 100 | 100 | 0 | 40 | 40 | 55 | 0 | 100 | 27 | 59 | 27 | 59 |
| FAC | 99 | 0 | 100 | 70 | 29 | 70 | 100 | 58 | 66 | 7 | 37 | 29 | 70 | 5 | 66 | 0 | 41 | 63 | 0 | 100 | 0 | 100 | 100 | 29 |
| QUANTITY | 0 | 0 | 66 | 65 | 80 | 68 | 100 | 31 | 100 | 0 | 100 | 65 | 34 | 66 | 34 | 0 | 34 | 100 | 65 | 68 | 99 | 34 | 34 | 65 |
| ORDINAL | 0 | 99 | 99 | 0 | 99 | 0 | 99 | 0 | 0 | 0 | 99 | 0 | 99 | 100 | 0 | 99 | 0 | 99 | 0 | 99 | 0 | 0 | 99 | 0 |
| PRODUCT | 48 | 48 | 100 | 79 | 51 | 0 | 100 | 69 | 71 | 30 | 20 | 28 | 51 | 100 | 100 | 20 | 28 | 71 | 0 | 99 | 20 | 30 | 51 | 20 |
| LAW | 53 | 74 | 79 | 79 | 20 | 53 | 100 | 20 | 100 | 6 | 99 | 0 | 79 | 100 | 79 | 100 | 73 | 99 | 39 | 79 | 20 | 100 | 20 | 40 |
| LANGUAGE | 5 | 2 | 96 | 5 | 95 | 97 | 6 | 2 | 97 | 0 | 6 | 0 | 97 | 6 | 98 | 96 | 0 | 100 | 93 | 99 | 4 | 99 | 99 | 5 |

Figure 4.8: The heatmap of bits distribution. The sub-figure above shows the distribution of bits concerning different syntactic information, while the one below corresponds to semantic information. The number inside cell represents the probability of this label being assigned a 1 at the $n$-th bit position. For example, the 6 at the bottom left corner indicates that among all of the `LANGUAGE` labels, only 6% of them are assigned a 1 at the first bit position.

bits become human-understandable in some degree. For example, the 24th bit shows a strong tendency towards only a few labels such as `CARDINAL`, `PERCENT`, and `ORDINAL`, we hypothesize that this bit preserves information about quantity. However, at the same time, we also acknowledge that we are currently unable to provide a clear definition of their meanings.

**Bit Flipping Tests**

In this section, we study how individual and paired bits impact the final parsing results. Our first finding is that arbitrary code sequences not always yield valid parsing results. This is because during the parsing process, Chomsky normal form and adopt left binarization (§4.3.1) are firstly adopted on the original parsing tree, intermediate labels are introduced to mark these incomplete non-terminal nodes. Some code sequences leads to that all nodes are assigned with incomplete labels, therefore, produces invalid trees. Therefore, We study their sensitivity by reversing the bits of the code sequence obtained from the hashing stage.

For simplicity, we start with 1-bit flipping tests, as shown in Figure 4.3.6. We found that bit flips have little effect on the final parsing for most tokens and at most bit positions. The parser often produces identical trees when given code sequences with a single bit flipped as inputs. This suggests potential redundancy in our binary representation, where information might be duplicated in multiple locations, such as parent nodes. Among all these tokens, only the tokens *deeper* and *swords* are affected by flipping bits. The token *deeper* changes its own attachment location as well as its parent's, while the token *swords* changes its sibling node.

Subsequently, we conducted further experiments on 2-bit flipping tests, and most yielded different trees. As shown in Figure 4.3.6, we found that all instances of the token *deeper* that change the first code from `0b0010` to `0x0110` have the non-terminal label `ADVP` removed. Interestingly, we did not observe a similar phenomenon in the 1-bit flipping tests. We hypothesize that this is because removing the non-terminal label `ADVP` also requires synchronizing and coordinating with the structure of parent nodes. To accommodate this change, their parent nodes also need to flip corresponding bits, as parsing is generally a span-level task. Therefore, such attachment location changes are not observed in 1-bit flipping tests.

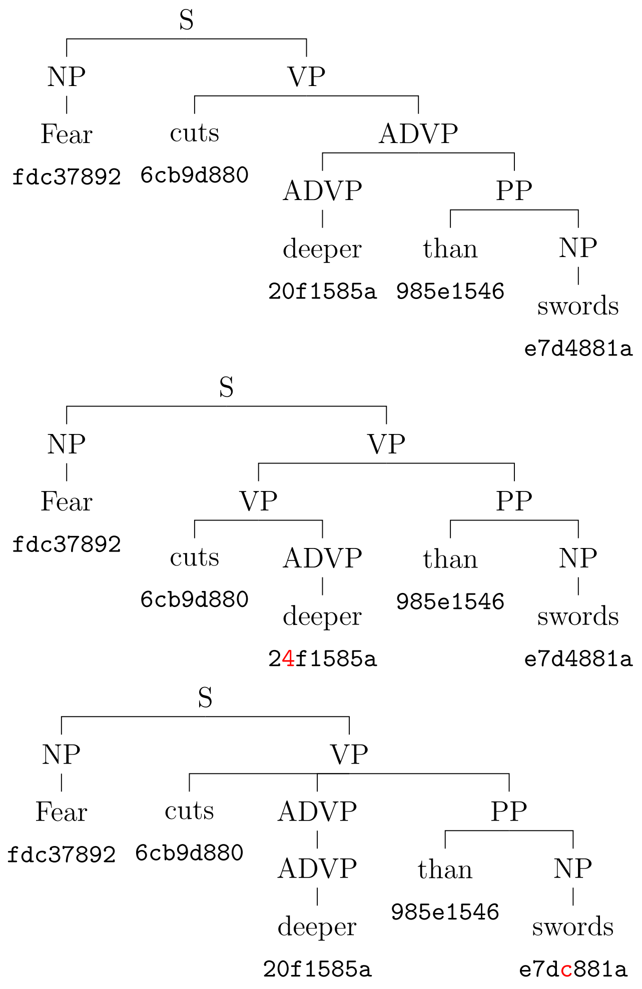


Figure 4.9: Derivation of the sentence *Fear cuts deeper than swords* with different code sequence as inputs. The red part indices the flipped bits.

From the above discussion, we can see that the internal decision-making mechanisms are somewhat revealed by these bits, but similar to the previous discussion (§4.3.6), the precise mechanisms remain unclear at this stage.

## 4.4 Conclusion

In this chapter, we have proposed a contrastive hashing method to generate interpretable binary codes from pre-trained language models. We designed a

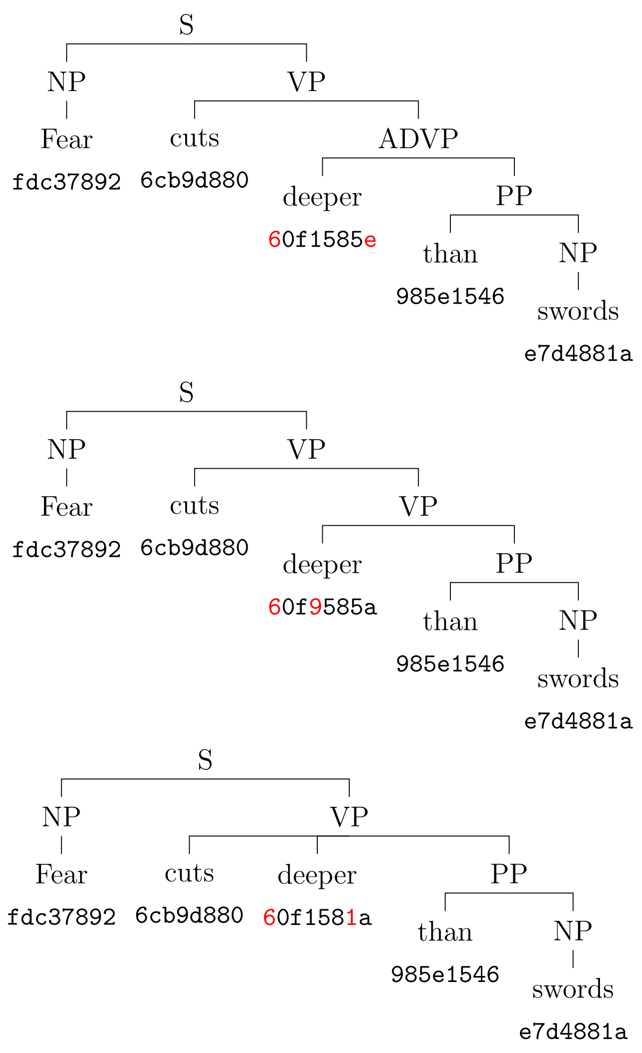


Figure 4.10: Derivation of the sentence *Fear cuts deeper than swords* with different code sequence as inputs. The red part indices the flipped bits.

transformer-based hash layer, incorporated it into the contrastive hashing framework, and introduced a novel instance selection strategy to minimize false positives and negatives. Experimental results indicate that our lightweight model achieves superior performance with fewer bits while effectively preserving task-relevant information. Further analyses show that the generated binary codes retain syntactic information in a relatively high interpretable and fine-grained manner. Although we primarily focus on these three tasks, as a novel interpretable and compact representation learning method, our approach can be easily adapted to other applications and may inspire future research on designing efficient architectures.

## 4.5 Limitations and Future Work

Our methods outperform previous work by achieving superior performance with fewer bits during the validation stage. However, there is still potential for improvement in non-token level tasks, such as constituency parsing.

Besides, from the interpretability aspect, our binary codes provide a discrete representation that is more interpretable than previous work. Our models tell not only if two tokens are different but also show in which bits they differ. However, at this stage, identifying which bits differ represents the limit of our model's interpretability. Questions like *what the specific meaning of each bit is* and *what is the internal decision making mechanism* are still unanswerable.

Moreover, even the limit has been pushed to 24 bits, which is much better than previous work, this is still not the theoretical limit. For example, the total number of labels of the named entity recognition task is 73, thus, the theoretical limit is $\lceil \log_2 73 \rceil = 7$ bits, which is still fewer than ours.

Future work might focus on further improving the information preservation capability on constituency parsing. For example, introducing span-level contrastive hashing instead of the current token-level one, due to the constituency parsing is inherently span-level classification task. Besides, the augmentation mechanism simply relies on randomly sampled dropout mask, future work might explore introducing more complex mechanism, such as lexical [105, 68] and tree substitution [16, 11] to produce harder negative instances.

# Chapter 5

# Conclusion

## 5.1 Summary

In this dissertation, we studied the benefits of incorporating structures into representation learning for different structured prediction tasks.

In Chapter 3, our method first discussed the harmful influence from the best path of previous level and analyzed how this problem arises. After that, we took the hierarchical structures of the nested named entity recognition task into account and designed a model to explicitly exclude the harmful influence from the best path. Moreover, we further applied chunk selection strategies to fully leverage the information between chunks and switched to the innermost-first encoding scheme. In this way, our model is capable of learning level-wise representation with the cross-level influence completely excluded. Experimental results indicate that our method achieves satisfactory performance.

Chapter 4, we focused on improving the interpretability of representation from pre-trained language models. By compressing the continuous vectors into discrete representations, our models induce compact and fine-grained binary codes with the task-relevant information completely preserved. Compared with the token-level tags of previous work, our binary codes not only indicate whether the syntactic properties of two given tokens are different, but also distinguish exactly which bits they differ in. Our method is built upon the contrastive hashing method. Additionally, we introduce an instance selection strategy and a transformer-based hash layer for learning to hash. The experiments demonstrate that during the validation stage, the lightweight model is capable of achieving performance comparable to that of the full-size model. Therefore, we claim that task-relevant information is fully preserved in these bitwise representations.

In these two chapters, we proposed two methods on learning level-wise and bit-wise representations, respectively. We demonstrate that leveraging structural information is beneficial to various tasks. Chapter 3 shows that explicitly introducing its hierarchical structures helps on improving performance on nested named entity recognition. Chapter 4 indicates that implicitly inducing structural information from target structures brings better interpretability and slight performance improvements. These findings might inspire future work on developing more efficient and interpretable models by introducing various structures in representation learning.

## 5.2 Limitations

Although we have improved performance and interpretability across various tasks and models, some issues still remain in current approaches.

For nested named entity recognition, the number of chunks is considered a hyper-parameter that must be determined before training. This limitation hinders its applicability in scenarios with more deeply nested entities, where this hyper-parameter may not be determinable in advance. Designing a method based on Principal Component Analysis (PCA) to directly decompose the level-wise representation could be a future research direction.

Regarding the bit-wise representation, performance on tasks that are not token-level still needs improvement, such as in constituency parsing. Even though the bottleneck has been reduced to 24 bits, which is significantly better than in previous work, it still isn't the theoretical limit. For instance, the total number of labels for named entity recognition is 73, so the theoretical limit is $\lceil \log_2 73 \rceil = 7$ bits, which is still much more compact than our codes. Besides, our model can only provid limited interpretability. Many questions, such as the internal decision making mechanisms and the exact meanings of each bit, remain unanswered. Introducing more complex structures may be a worthwhile direction for future research.

# List of Publications

## Journal Papers

1. Yiran Wang, Hiroyuki Shindo, Yuji Matsumoto, and Taro Watanabe. 2022. Nested named entity recognition via explicitly excluding the influence of the best path. *Journal of Natural Language Processing*, 29(1):23–52

## International Conference Papers

1. Yiran Wang, Taro Watanabe, Masao Utiyama, and Yuji Matsumoto. 2023. 24-bit languages. *In Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 408–419, Nusa Dua, Bali. Association for Computational Linguistics

2. Yiran Wang, Hiroyuki Shindo, Yuji Matsumoto, and Taro Watanabe. 2021. Nested named entity recognition via explicitly excluding the influence of the best path. *In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3547–3557, Online. Association for Computational Linguistics

3. Yiran Wang, Hiroyuki Shindo, Yuji Matsumoto, and Taro Watanabe. 2021. Structured refinement for sequential labeling. *In Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1873–1884, Online. Association for Computational Linguistics

4. Jun Liu, Fei Cheng, Yiran Wang, Hiroyuki Shindo, and Yuji Matsumoto. 2018. Automatic error correction on Japanese functional expressions using character-based neural machine translation. *In Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*, Hong Kong. Association for Computational Linguistics

# Bibliography

[1] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–1.

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

[3] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. 2017. Deep variational information bottleneck. In *International Conference on Learning Representations*.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

[5] Jiangang Bai, Yujing Wang, Yiren Chen, Yaming Yang, Jing Bai, Jing Yu, and Yunhai Tong. 2021. Syntax-BERT: Improving pre-trained transformers with syntax trees. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3011–3020, Online. Association for Computational Linguistics.

[6] J. K. Baker. 2005. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132.

[7] Sriram Balasubramanian, Naman Jain, Gaurav Jindal, Abhijeet Awasthi, and Sunita Sarawagi. 2020. What's in a name? are BERT named entity representations just as good for any other name? In *Proceedings of the 5th*

*Workshop on Representation Learning for NLP*, pages 205–214, Online. Association for Computational Linguistics.

[8] Leonard E Baum et al. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3(1):1–8.

[9] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

[10] Bernd Bohnet, Ryan McDonald, Gonçalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez. 2018. Morphosyntactic tagging with a meta-BiLSTM model over context sensitive token encodings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2642–2652, Melbourne, Australia. Association for Computational Linguistics.

[11] Steven Cao, Nikita Kitaev, and Dan Klein. 2020. Unsupervised parsing via constituency tests. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4798–4808, Online. Association for Computational Linguistics.

[12] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S. Yu. 2017. Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV).*

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

[14] Billy Chiu, Gamal Crichton, Anna Korhonen, and Sampo Pyysalo. 2016. How to train good word embeddings for biomedical NLP. In *Proceedings of the 15th Workshop on Biomedical Natural Language Processing*, pages 166–174, Berlin, Germany. Association for Computational Linguistics.

[15] Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

[16] Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11(102):3053–3096.

[17] James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[19] George Doddington, Alexis Mitchell, Mark Przybocki, Lance Ramshaw, Stephanie Strassel, and Ralph Weischedel. 2004. The automatic content extraction (ACE) program – tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal. European Language Resources Association (ELRA).

[20] Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17, Austin, TX. Association for Computational Linguistics.

[21] Jenny Rose Finkel and Christopher D. Manning. 2009. Nested named entity recognition. In *Proceedings of the 2009 Conference on Empirical Methods*

*in Natural Language Processing*, pages 141–150, Singapore. Association for Computational Linguistics.

[22] Joseph Fisher and Andreas Vlachos. 2019. Merge and label: A novel neural network architecture for nested NER. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5840–5850, Florence, Italy. Association for Computational Linguistics.

[23] G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

[24] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[25] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. 2020. Bootstrap your own latent - a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284. Curran Associates, Inc.

[26] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In *International Conference on Learning Representations.*

[27] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[28] Jiun Tian Hoe, Kam Woh Ng, Tianyu Zhang, Chee Seng Chan, Yi-Zhe Song, and Tao Xiang. 2021. One loss for all: Deep hashing with a single cosine similarity based learning objective. In *Advances in Neural Information Processing Systems.*

[29] Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R. Bowman. 2019. Do attention heads in bert track syntactic dependencies?

[30] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

[31] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

[32] Meizhi Ju, Makoto Miwa, and Sophia Ananiadou. 2018. A neural layered model for nested named entity recognition. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1446–1459, New Orleans, Louisiana. Association for Computational Linguistics.

[33] Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages.

[34] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 18661–18673. Curran Associates, Inc.

[35] J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. 2003. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19:i180–i182.

[36] Taeuk Kim, Jihun Choi, Daniel Edmiston, and Sang goo Lee. 2020. Are pre-trained language models aware of phrases? simple but strong baselines for grammar induction. In *International Conference on Learning Representations*.

[37] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

[38] Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th*

*Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

[39] Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

[40] Nikita Kitaev, Thomas Lu, and Dan Klein. 2022. Learned incremental representations for parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3086–3095, Dublin, Ireland. Association for Computational Linguistics.

[41] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519.

[42] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[43] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

[44] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *CoRR*, abs/1901.08746.

[45] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

[46] Xiang Lisa Li and Jason Eisner. 2019. Specializing word embeddings (for parsing) by information bottleneck. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2744–2754, Hong Kong, China. Association for Computational Linguistics.

[47] Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. 2019. Sequence-to-nuggets: Nested entity mention detection via anchor-region networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5182–5192, Florence, Italy. Association for Computational Linguistics.

[48] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. 2016. Large-margin softmax loss for convolutional neural networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 507–516, New York, New York, USA. PMLR.

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

[50] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

[51] Wei Lu and Dan Roth. 2015. Joint mention extraction and classification with mention hypergraphs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 857–867, Lisbon, Portugal. Association for Computational Linguistics.

[52] Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. 2015. Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 879–888, Lisbon, Portugal. Association for Computational Linguistics.

[53] Ying Luo and Hai Zhao. 2020. Bipartite flat-graph network for nested named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6408–6418, Online. Association for Computational Linguistics.

[54] Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany. Association for Computational Linguistics.

[55] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[56] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861.

[57] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

[58] Aldrian Obaja Muis and Wei Lu. 2017. Labeling gaps between words: Recognizing overlapping mentions with mention separators. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2608–2618, Copenhagen, Denmark. Association for Computational Linguistics.

[59] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding.

[60] Zijing Ou, Qinliang Su, Jianxing Yu, Ruihui Zhao, Yefeng Zheng, and Bang Liu. 2021. Refining BERT embeddings for document hashing via mutual information maximization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2360–2369, Punta Cana, Dominican Republic. Association for Computational Linguistics.

[61] Tommaso Pasini, Federico Scozzafava, and Bianca Scarlini. 2020. CluBERT: A cluster-based approach for learning sense distributions in multiple languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4008–4018, Online. Association for Computational Linguistics.

[62] Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon infused phrase embeddings for named entity resolution. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 78–86, Ann Arbor, Michigan. Association for Computational Linguistics.

[63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc.

[64] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[65] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[66] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

[67] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards robust linguistic analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152, Sofia, Bulgaria. Association for Computational Linguistics.

[68] Fanchao Qi, Yuan Yao, Sophia Xu, Zhiyuan Liu, and Maosong Sun. 2021. Turn the combination lock: Learnable textual backdoor attacks via word substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4873–4883, Online. Association for Computational Linguistics.

[69] Zexuan Qiu, Qinliang Su, Jianxing Yu, and Shijing Si. 2022. Efficient document retrieval by end-to-end refining and quantizing BERT embedding with contrastive product quantization. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 853–863, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[70] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

[71] Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

[72] Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado. Association for Computational Linguistics.

[73] Marek Rei, Gamal Crichton, and Sampo Pyysalo. 2016. Attending to characters in neural sequence labeling models. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 309–318, Osaka, Japan. The COLING 2016 Organizing Committee.

[74] Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B Viegas, Andy Coenen, Adam Pearce, and Been Kim. 2019. Visualizing and measuring the geometry of bert. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[75] Nicky Ringland, Xiang Dai, Ben Hachey, Sarvnaz Karimi, Cecile Paris, and James R. Curran. 2019. NNE: A dataset for nested named entity recognition in English newswire. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5176–5181, Florence, Italy. Association for Computational Linguistics.

[76] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[77] Alexander Rush. 2020. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, Online. Association for Computational Linguistics.

[78] Takashi Shibuya and Eduard Hovy. 2020. Nested named entity recognition via second-best sequence learning and decoding. *Transactions of the Association for Computational Linguistics*, 8:605–620.

[79] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural net-

works from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[80] Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

[81] Jana Straková, Milan Straka, and Jan Hajic. 2019. Neural architectures for nested NER through linearization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5326–5331, Florence, Italy. Association for Computational Linguistics.

[82] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. 2018. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[83] Sho Takase, Jun Suzuki, and Masaaki Nagata. 2018. Direct output connection for a high-rank language model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4599–4609, Brussels, Belgium. Association for Computational Linguistics.

[84] Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

[85] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.

[86] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE.

[87] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle.

[88] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[92] David Vilares, Michalina Strzyz, Anders Søgaard, and Carlos Gómez-Rodríguez. 2020. Parsing as pretraining. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9114–9121.

[93] Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 57:45.

[94] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5307–5315, Hong Kong, China. Association for Computational Linguistics.

[95] Bailin Wang and Wei Lu. 2018. Neural segmental hypergraphs for overlapping mention recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 204–214, Brussels, Belgium. Association for Computational Linguistics.

[96] Bailin Wang, Wei Lu, Yu Wang, and Hongxia Jin. 2018. A neural transition-based model for nested mention recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1011–1017, Brussels, Belgium. Association for Computational Linguistics.

[97] Feng Wang and Huaping Liu. 2021. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2495–2504.

[98] Jue Wang, Lidan Shou, Ke Chen, and Gang Chen. 2020. Pyramid: A layered model for nested named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5918–5928, Online. Association for Computational Linguistics.

[99] Yefeng Wang. 2009. Annotating and recognising named entities in clinical notes. In *Proceedings of the ACL-IJCNLP 2009 Student Research Workshop*, pages 18–26, Suntec, Singapore. Association for Computational Linguistics.

[100] Yiran Wang, Yuji Matsumoto, Masao Utiyama, and Taro Watanabe. 2023. 24-bit languages. In *The 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics.*

[101] Yiran Wang, Hiroyuki Shindo, Yuji Matsumoto, and Taro Watanabe. 2021. Nested named entity recognition via explicitly excluding the influence of the best path. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3547–3557, Online. Association for Computational Linguistics.

[102] Gregor Wiedemann, Steffen Remus, Avi Chawla, and Chris Biemann. 2019. Does bert make any sense? interpretable word sense disambiguation with contextualized embeddings.

[103] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. pages 38–45. Association for Computational Linguistics.

[104] Huiyin Xue and Nikolaos Aletras. 2022. HashFormers: Towards vocabulary-independent pre-trained transformers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7862–7874, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[105] Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online. Association for Computational Linguistics.

[106] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. 2021. Barlow twins: Self-supervised learning via redundancy reduction. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12310–12320. PMLR.

[107] Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020. Fast and accurate neural crf constituency parsing. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4046–4053. International Joint Conferences on Artificial Intelligence Organization. Main track.