Nested Named Entity Recognition via Second-best Sequence Learning and Decoding

Takashi Shibuya[†‡] **Eduard Hovy**[†]
† Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
‡ Sony Corporation, Tokyo 141-8610, Japan
`shibuyat@jp.sony.com` `hovy@cmu.edu`

Abstract

When an entity name contains other names within it, the identification of all combinations of names can become difficult and expensive. We propose a new method to recognize not only outermost named entities but also inner nested ones. We design an objective function for training a neural model that treats the tag sequence for nested entities as the second best path within the span of their parent entity. In addition, we provide the decoding method for inference that extracts entities iteratively from outermost ones to inner ones in an outside-to-inside way. Our method has no additional hyperparameters to the conditional random field based model widely used for flat named entity recognition tasks. Experiments demonstrate that our method outperforms existing methods capable of handling nested entities, achieving the F1-scores of 82.81%, 82.70%, and 77.25% on ACE-2004, ACE-2005, and GENIA datasets, respectively.

1 Introduction

Named entity recognition (NER) is the task of identifying text spans associated with proper names and classifying them according to their semantic class such as person or organization. NER, or in general the task of recognizing entity mentions, is one of the first stages in deep language understanding and its importance has been well recognized in the NLP community (Nadeau and Sekine, 2007).

One popular approach to the task of NER is to regard it as a sequence labeling problem. In this case, it is implicitly assumed that mentions are not nested in texts. However, names often contain entities nested within themselves, as illustrated in Fig. 1, which contains 3 mentions of the same type (PROTEIN) in the span "... *in Ca2+ -dependent PKC isoforms in* ...", taken from the GENIA dataset (Kim et al., 2003). Name nesting is common, especially in technical domains (Alex et al., 2007; Byrne, 2007; Wang, 2009). The assumption of no nesting leads to loss of potentially important information and may negatively impact subsequent downstream tasks. For instance, a downstream entity linking system that relies on NER may fail to link the correct entity if the entity mention is nested.

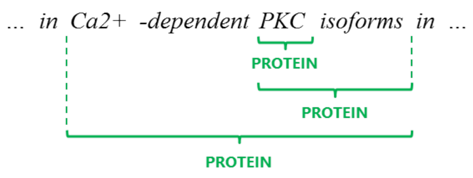


Figure 1: Example of nested entities.

Various approaches to recognizing nested entities have been proposed. Many of them rely on producing and rating all possible (sub)spans, which can be computationally expensive. Wang and Lu (2018) provided a hypergraph-based approach to consider all possible spans. Sohrab and Miwa (2018) proposed a neural exhaustive model that enumerates and classifies all possible spans. These methods, however, achieve high performance at the cost of time complexity. To reduce the running time, they set a threshold to discard longer entity mentions. If the hyperparameter is set low, running time is reduced but longer mentions are missed. In contrast, Muis and Lu (2017) proposed a sequence labeling approach that assigns tags to gaps between words, which efficiently handles sequences using Viterbi decoding. However, this approach suffers from structural ambiguity issues during inference as explained by Wang and Lu (2018). Katiyar and Cardie (2018) proposed another hypergraph-based approach that learns the structure in a greedy manner. However, their method uses an additional hyperparameter as the threshold for selecting multiple mention candidates. This hyperparameter affects the trade-off between recall and precision.

In this paper, we propose new learning and decoding methods to extract nested entities without any additional hyperparameters. We summarize our contributions as:

- We describe a decoding method that iteratively recognizes entities from outermost ones to inner ones without structural ambiguity. It recursively searches a span of each extracted entity for inner nested entities using the Viterbi algorithm. This algorithm does not require hyperparameters for the maximal length or number of mentions considered.

- We also provide a novel learning method that ensures the aforementioned decoding. Models are optimized based on an objective function designed according to the decoding procedure.

- Empirically, we demonstrate that our method outperforms the current state-of-the-art methods with 82.81%, 82.70%, and 77.25% in F1-score on three standard datasets: ACE-2004[1], ACE-2005[2], and GENIA.

2 Related Work

The success of neural networks has boosted the performance of NER (Strubell et al., 2017; Akbik et al., 2018). However, few attempts have explored nested entity recognition.

Alex et al. (2007) proposed several ways to combine multiple CRFs for such tasks. They separately built inside-out and outside-in layered CRFs that used the current guesses as the input for the next layer. They also cascaded separate CRFs of each entity type by using the output from the previous CRF as the input features of the current CRF, yielding best performance in their work. However, their method could not handle nested entities of the same entity type. In contrast, Ju et al. (2018) dynamically stacked multiple layers that recognize entities sequentially from innermost ones to outermost ones. Their method can deal with nested entities of the same entity type.

[1]https://catalog.ldc.upenn.edu/LDC2005T09

[2]https://catalog.ldc.upenn.edu/LDC2006T06

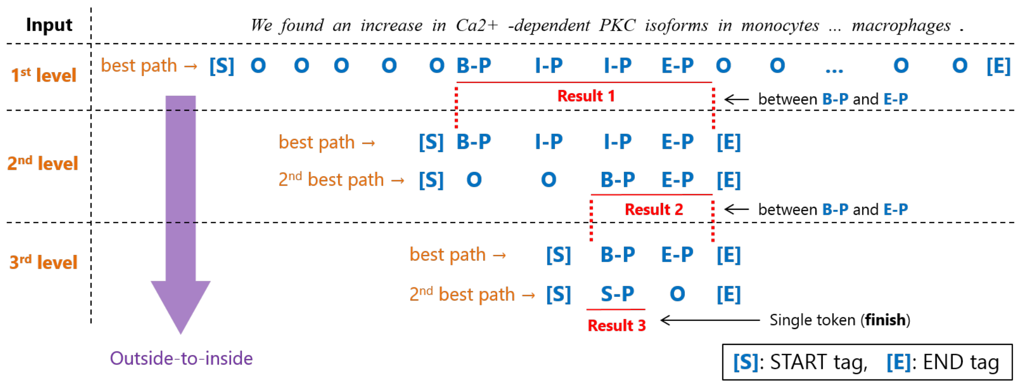
Finkel and Manning (2009) proposed a CRF-based constituency parser for this task such that each named entity is a node in the parse tree. Their model achieved state-of-the-art performance on the GENIA dataset of the time. However, its time complexity is the cube of the length of a given sentence, making it not scalable to large datasets involving long sentences. Later on, Wang et al. (2018) proposed a scalable transition-based approach, a constituency forest (a collection of constituency trees). Its time complexity is linear in the sentence length.

Lu and Roth (2015) introduced a mention hypergraph representation for capturing nested entities as well as crossing entities (two entities overlap but neither is contained in the other). One issue in their approach is the spurious structures of the representation. To address the spurious structures issue, Muis and Lu (2017) incorporated mention separators to yield better performance on nested entities. On the other hand, it still suffers from the structural ambiguity issue. Wang and Lu (2018) proposed a hypergraph representation free of structural ambiguity, and their method achieved state-of-the-art performance on the ACE-2004 and ACE-2005 datasets. However, they introduced a hyperparameter, the maximal length of an entity, to reduce the time complexity. Setting the hyperparameter to a small number results in speeding up but ignoring longer entity segments.

Katiyar and Cardie (2018) proposed another hypergraph-based approach that learns the structure using an LSTM network in a greedy manner. However, their method has a hyperparameter that sets a threshold for selecting multiple candidate mentions. It must be carefully tuned for adjusting the trade-off between recall and precision.

Sohrab and Miwa (2018) proposed a neural exhaustive model that enumerates all possible spans as potential entity mentions and classifies them. Their model achieved state-of-the-art performance on the GENIA dataset in terms of F1-score. However, they also use the maximal-length hyperparameter to reduce time complexity.

We show that our model outperforms all these methods on three datasets: ACE-2004, ACE-2005, and GENIA. In addition, our model has no additional hyperparameters to the conventional CRF-based model commonly used for flat NER. Also, our decoding algorithm is fast in practice as shown in Subsection 5.4.



Input
We found an increase in Ca2+ -dependent PKC isoforms in monocytes ... macrophages .
1st level
best path → [S] O O O O O B-P I-P I-P E-P O O ... O O [E]
Result 1
between B-P and E-P
2nd level
best path → [S] B-P I-P I-P E-P [E]
2nd best path → [S] O O B-P E-P [E]
Result 2
between B-P and E-P
3rd level
best path → [S] B-P E-P [E]
2nd best path → [S] S-P O [E]
Result 3
Single token (finish)
Outside-to-inside
[S]: START tag, [E]: END tag

Figure 2: Overview of our second-best path decoding algorithm to iteratively find nested entities.

3 Method

We first explain our usage of CRF, which is the base of our decoding and training methods. Then, we introduce our decoding and training methods. Our decoding and training methods focus on the output layer of neural architectures and therefore can be combined with any neural model.

3.1 Usage of CRF

We first explain two key points about our usage of CRF. The first key point is that we prepare a separate CRF for each named entity type. This enables our method to handle the situation where the same mention span is assigned multiple entity types. The second one is that each element of the transition matrix of each CRF has a fixed value according to whether it corresponds to a legal transition (*e.g.*, B-XXX to I-XXX, where XXX is the name of entity type) or an illegal one (*e.g.*, O to I-XXX). This is helpful for keeping the scores for tag sequences including outer entities higher than those of tag sequences including inner entities.

Formally, we use $\boldsymbol{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ to represent a sequence output from the last hidden layer of a neural model, where $\boldsymbol{z}_i$ is the vector for the i-th word, and n is the number of tokens. $\boldsymbol{y}^{(k)} = \{y_1^{(k)}, \ldots, y_n^{(k)}\}$ represents a sequence of IOBES tags of entity type k for $\boldsymbol{Z}$. Here, we define the score function to be

$$\phi_k\left(y_{i-1}^{(k)}, y_i^{(k)}, \boldsymbol{z}_i\right) = \boldsymbol{P}_{y_i^{(k)},i}^{(k)} + \boldsymbol{A}_{y_{i-1}^{(k)},y_i^{(k)}}^{(k)}, \quad (1)$$

where $\boldsymbol{P}_{y_i^{(k)},i}^{(k)} = \boldsymbol{W}_{y_i^{(k)}}^{(k)} \cdot \boldsymbol{z}_i + \boldsymbol{b}_{y_i^{(k)}}^{(k)}$,

$$\boldsymbol{A}_{y_{i-1}^{(k)},y_i^{(k)}}^{(k)} = \begin{cases} -\infty, & \text{if } y_{i-1}^{(k)} \rightarrow y_i^{(k)} \text{ is illegal,} \\ 0, & \text{otherwise.} \end{cases}$$

$\boldsymbol{W}_{y_i^{(k)}}^{(k)}$ and $\boldsymbol{b}_{y_i^{(k)}}^{(k)}$ denote the weight matrix and the bias vector corresponding to $y_i^{(k)}$, respectively. $\boldsymbol{A}^{(k)}$ stands for the transition matrix from the previous token to the current token, and $\boldsymbol{A}_{y_{i-1}^{(k)},y_i^{(k)}}^{(k)}$ is the transition scores from $y_{i-1}^{(k)}$ to $y_i^{(k)}$.

3.2 Decoding

We employ three strategies for decoding. First, we consider each entity type separately using multiple CRFs in decoding, which makes it possible to handle the situation that the same mention span is assigned multiple entity types. Second, our decoder searches nested entities in an outside-to-inside way[3], which realizes efficient processing by eliminating the spans of non-entity at an early stage. More specifically, our method recursively narrows down the spans to Viterbi-decode. The spans to Viterbi-decode are dynamically decided according to the preceding Viterbi-decoding result. Only the spans that have just been recognized as entity mentions are Viterbi-decoded again. Third, we use the same scores $\phi_k\left(y_{i-1}^{(k)}, y_i^{(k)}, \boldsymbol{z}_i\right)$ of Eq. 1 to extract outermost entities and even inner entities without re-encoding, which makes inference more efficient and faster. These three strategies are deployed and completed only in the output layer of neural architectures.

We describe the pseudo-code of our decoding method in Algorithm 1. Also, we depict the overview of our decoding method with an example in Fig. 2 . We use the term *level* in the sense of the depth of entity nesting. [S] and [E] in Fig. 2 stand for the START and END tags respectively. We always attach these tags to both ends of every

[3]Our usage of *inside/outside* is different from the inside-outside algorithm in dynamic programming.

Algorithm 1: Nested NER via 2nd-best sequence decoding

```
$K$ = the set of the entity types;
Function main($\boldsymbol{z}_i$)
    $M = \{\}$; # the set of detected mentions. Each element of $M$ is a tuple ($s$, $e$, $k$) regarding a mention.
        # $s$, $e$, and $k$ are the start position, the end position, and the entity type of the mention, respectively.
    foreach $k \in K$ do
        calculate CRF scores $\boldsymbol{\Phi}$ for entity type $k$ with the score function $\phi_k\left(y_{i-1}^{(k)}, y_i^{(k)}, \boldsymbol{z}_i\right)$;
        find the best path of the span from position 1 to position $n$ based on the scores $\boldsymbol{\Phi}$;
        $\tilde{M}$ = the set of the mentions detected in the best path;
        $M = M \cup \tilde{M}$;
        foreach $m \in \tilde{M}$ do
            detectNestedMentions($\boldsymbol{\Phi}$, $m.s$, $m.e$, $k$, $M$);
    return $M$;
Function detectNestedMentions($\boldsymbol{\Phi}$, $s$, $e$, $k$, $M$)
    if $e - s > 1$ then
        find the 2nd best path of the span from position $s$ to position $e$ based on the scores $\boldsymbol{\Phi}$;
        $\tilde{M}$ = the set of the mentions detected in the 2nd best path;
        $M = M \cup \tilde{M}$;
        foreach $m \in \tilde{M}$ do
            detectNestedMentions($\boldsymbol{\Phi}$, $m.s$, $m.e$, $k$, $M$);
    return;
```

sequence of IOBES tags in Viterbi-decoding.

We explain the decoding procedure and mechanism in detail below. We consider each entity type separately and iterate the same decoding process regarding distinct entity types as described in Algorithm 1. In the decoding process for each entity type k, we first calculate the CRF scores $\phi_k\left(y_{i-1}^{(k)}, y_i^{(k)}, \boldsymbol{z}_i\right)$ over the entire sentence. Next, we decode a sequence with the standard 1-best Viterbi decoding as with the conventional linear-chain CRF. "*Ca2+ -dependent PKC isoforms*" is extracted at the 1st level with regard to the example of Fig. 2.

Then, we start our recursive decoding to extract nested entities within previously extracted entity spans by finding the 2nd best path. In Fig. 2, the span "*Ca2+ -dependent PKC isoforms*" is processed at the 2nd level. Here, if we search for the best path within each span, the same tag sequence will be obtained, even though the processed span is different. This is because we continue using the same scores $\phi_k\left(y_{i-1}^{(k)}, y_i^{(k)}, \boldsymbol{z}_i\right)$ and because all the values of $\boldsymbol{A}^{(k)}$ corresponding to legal transitions are equal to 0. Regarding the example of Fig. 2, the score of the transition from `[S]` to `B-P` at the 2nd level is equal to the score of the transition from `O` to `B-P` at the 1st level. This is true for the transition from `E-P` to `[E]` at the 2nd level and the one from `E-P` to `O` at the 1st level. The best path between the `[S]` and `[E]` tags is identical to the best path between the two `O` tags under our restriction about the transition matrix of CRF. Therefore, we search for the 2nd best path within the span by utilizing the N-best Viterbi A* algorithm (Seshadri and Sundberg, 1994; Huang et al., 2012).[4] Note that our situation is different from normal situations where N-best decoding is needed. We already know the best path within the span and want to find only the 2nd best path. Thus, we can extract nested entities by finding the 2nd best path within each extracted entity. Regarding the example of Fig. 2, "*PKC isoforms*" is extracted from the span "*Ca2+ -dependent PKC isoforms*" at the 2nd level.

We continue this recursive decoding until no entities are predicted or when only single-token entities are detected within a span. In Fig. 2, the span "*PKC isoforms*" is processed at the 3rd level. At the 3rd or deeper levels, the tag sequence of its grandparent level is no longer either the best path or the 2nd best path because the start or end po-

[4]Without our restriction about the transition matrix of CRF, we would have to watch both the best path and the 2nd best path.

Algorithm 2: LogSumExp of the scores of all possible paths

```
$C = \{\texttt{B-XXX}, \texttt{I-XXX}, \texttt{E-XXX}, \texttt{S-XXX}, \texttt{O}\}$;
$s = 1$; # the start position
$e = n$; # the end position
foreach $c \in C$ do
    $\boldsymbol{\alpha}(c) = \boldsymbol{P}^{(k)}_{c,s} + \boldsymbol{A}^{(k)}_{[\texttt{S}],c}$;
for $i = s + 1; i \leq e; i{+}{+}$ do
    foreach $c \in C$ do
        foreach $c' \in C$ do
            $\boldsymbol{\alpha}_c(c') = \boldsymbol{\alpha}(c') + \boldsymbol{P}^{(k)}_{c,i} + \boldsymbol{A}^{(k)}_{c',c}$;
    foreach $c \in C$ do
        $\boldsymbol{\alpha}(c) = \text{LogSumExp}(\boldsymbol{\alpha}_c)$;
foreach $c \in C$ do
    $\boldsymbol{\alpha}(c) += \boldsymbol{A}^{(k)}_{c,[\texttt{E}]}$;
return LogSumExp $(\boldsymbol{\alpha})$;
```

sition of the current span is in the middle of the entity mention at the grandparent level. As for the example shown in Fig. 2, "*PKC*" is tagged `I-P` at the 1st level, and the transition from `[S]` to `I-P` is illegal. The scores of the paths that includes illegal transitions cannot be larger than those of the paths that consist of only legal transitions because the elements of the transition matrix $\boldsymbol{A}^{(k)}$ corresponding to illegal transitions are set to $-\infty$. That is why at all levels below the 1st level we only need to find the 2nd best path.

This recursive processing is stopped when no entities are predicted or when only single-token entities are detected within a span. In Fig. 2, the span "*PKC*" is not processed any more because it is a single-token entity. The aforementioned processing is executed on all entity types, and all detected entities are returned as an output result.

3.3 Training

To extract entities from outside to inside successfully, a model has to be trained in a way that the scores for the paths including outer entities will be higher than those for the paths including inner entities. We propose a new objective function to achieve this requirement.

We maximize the log-likelihood of the correct tag sequence as with the conventional CRF-based model. Considering that our model has a separate CRF for each entity type, the log-likelihood for one training data, $\mathcal{L}(\boldsymbol{\theta})$, is as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_k \log p\left(\boldsymbol{Y}^{(k)} | \boldsymbol{Z}; \boldsymbol{\theta}\right), \tag{2}$$

where $\boldsymbol{\theta}$ is the set of parameters of a neural model, and $\boldsymbol{Y}^{(k)}$ denotes the collection of the gold IOBES tags for all levels regarding the entity type k.

Let $s^{(k)}_{l,j}$ and $e^{(k)}_{l,j}$ denote the start and end positions of the j-th span at the l-th level. With regard to the 1st level, $s^{(k)}_{1,1} = 1$ and $e^{(k)}_{1,1} = n$ because we consider the whole span of a sentence. The spans considered at each deeper level, $l > 1$, are determined according to the spans of multi-token entities at its immediate parent level. As for the example of Fig. 2, only the span of "*Ca2+ -dependent PKC isoforms*" is considered at the 2nd level. Here, the log-likelihood for each entity type can be expressed as follows:

$$\log p\left(\boldsymbol{Y}^{(k)} | \boldsymbol{Z}; \boldsymbol{\theta}\right) = L_{\text{1st}}\left(y^{(k)}_{1,1}, \ldots, y^{(k)}_{1,n} | \boldsymbol{Z}; \boldsymbol{\theta}\right) + \sum_{l>1} \sum_j L_{\text{2nd}}\left(y^{(k)}_{l,s^{(k)}_{l,j}}, \ldots, y^{(k)}_{l,e^{(k)}_{l,j}} | \boldsymbol{Z}; \boldsymbol{\theta}\right), \tag{3}$$

where $L_{\text{1st}}(\ldots)$ and $L_{\text{2nd}}(\ldots)$ are the log-likelihoods of the (1st) best and 2nd best paths for each span, respectively. $y^{(k)}_{l,i}$ denotes the correct IOBES tag of the position i of the l-th level of the entity type k.

Best path. $L_{\text{1st}}(\ldots)$ can be calculated in the same manner as the conventional linear-chain CRF:

$$L_{\text{1st}}\left(y^{(k)}_{1,1}, \ldots, y^{(k)}_{1,n} | \boldsymbol{Z}; \boldsymbol{\theta}\right) = \psi^{(k)}_{1:n}\left(\boldsymbol{y}^{(k)}_1, \boldsymbol{Z}\right) - \log \sum_{\boldsymbol{y}' \in \mathcal{Y}^{(k)}_{1:n}} \exp \psi^{(k)}_{1:n}(\boldsymbol{y}', \boldsymbol{Z}), \tag{4}$$

$$\text{where} \quad \psi^{(k)}_{s:e}(\boldsymbol{y}, \boldsymbol{Z}) = \sum_{i=s}^{e} \phi_k(y_{i-1}, y_i, \boldsymbol{z}_i) + \boldsymbol{A}^{(k)}_{y_e, y_{e+1}}, \quad y_{s-1} = [\texttt{S}],\ y_{e+1} = [\texttt{E}].$$

$\mathcal{Y}^{(k)}_{s:e}$ denotes the set of all possible tag sequences from position s to position e of the entity type k. It is well known that the second term of Eq. 4 can be efficiently calculated by the algorithm shown in Algorithm 2.

2nd best path. $L_{\text{2nd}}(\ldots)$ given the best path can be calculated by excluding the best path from all

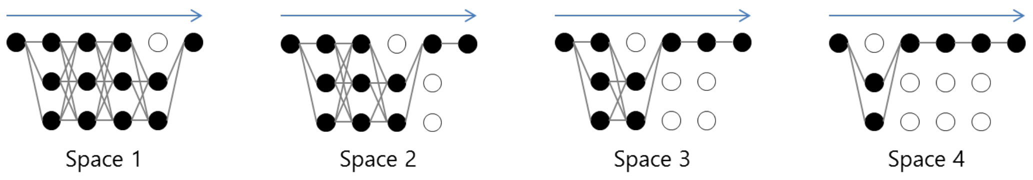


Figure 3: Divided search spaces.

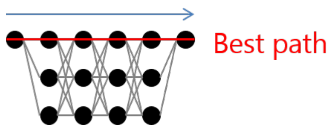


Figure 4: Lattice and best path.

possible paths. This concept is also adopted by ListNet (Cao et al., 2007), which is used for ranking tasks such as document retrieval or recommendation. $L_{\text{2nd}}(\dots)$ can be expressed by the following equation:

$$L_{\text{2nd}}\left(y^{(k)}_{l,s^{(k)}_{l,j}}, \dots, y^{(k)}_{l,e^{(k)}_{l,j}} | \boldsymbol{Z}; \boldsymbol{\theta}\right) = \psi^{(k)}_{s^{(k)}_{l,j}:e^{(k)}_{l,j}}\left(\boldsymbol{y}^{(k)}_{d}, \boldsymbol{Z}\right) - \log \sum_{\boldsymbol{y}' \in \tilde{\mathcal{Y}}^{(k)}_{s^{(k)}_{l,j}:e^{(k)}_{l,j}}} \exp \psi^{(k)}_{s^{(k)}_{l,j}:e^{(k)}_{l,j}}\left(\boldsymbol{y}', \boldsymbol{Z}\right), \tag{5}$$

where $\tilde{\mathcal{Y}}^{(k)}_{s:e}$ denotes the set of all possible tag sequences except the best path within the span from position s to position e of the entity type k.

However, to the best of our knowledge, the way of efficiently computing the second term of Eq. 5 has not been proposed yet in the literature. Simply subtracting the exponential score of the best path from the summation of the exponential scores of all possible paths causes underflow, overflow, or loss of significant digits. We introduce the way of accurately computing it with the same time complexity as Eq. 4. For explanation, we use the simplified example of the lattice depicted in Fig. 4, in which the span length is 4 and the number of states is 3. The special nodes for start and end states are attached to the both ends of the span. There are $81(=3^4)$ paths in this lattice. We assume that the path that consists of top nodes of all time steps are the best path as shown in Fig. 4. No generality is lost by making this assumption. To calculate the second term of Eq. 5, we have to consider the exponential scores for all the possible paths except the best path, $80(=81-1)$ paths.

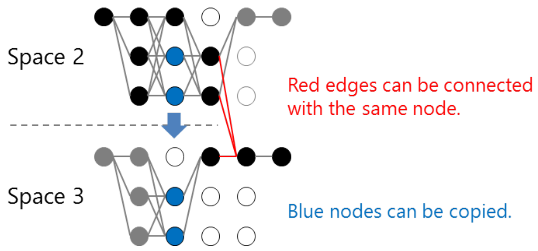

Space 2
Red edges can be connected with the same node.
Space 3
Blue nodes can be copied.

Figure 5: Merge of search spaces.

We first give a way of thinking, which is not our algorithm itself but helpful to understand it. In the example, we can further group these 80 paths according to the steps where the best path is not taken. In this way, we have 4 spaces in total as illustrated in Fig. 3. In Space 1, the top node of time step 4 is excluded from consideration. $54(=3^3 \times 2)$ paths are taken into account here. Since this space covers all paths that do not go through the top node of time step 4, we only have to consider the paths that go through this node in other spaces. In Space 2, this node is always passed through, and instead the top node of time step 3 is excluded. $18(=3^2 \times 2)$ paths are considered in this space. Similarly, $6(=3^1 \times 2)$ paths and $2(=3^0 \times 2)$ paths are taken into consideration in Space 3 and Space 4, respectively. Thus, we can consider all the possible paths except the best path, $80(=54+18+6+2)$ paths. However, this is not our algorithm itself as we mentioned.

We introduce two tricks for making the calculation more efficient. We explain them with Fig. 5, in which Spaces 2 and 3 are picked up. The first trick is that the separated two spaces can be merged at time step 4 because the paths later than time step 3 are identical. When we reach time step 4 in the forward iteration in each of the two spaces, we can merge them using the calculation results at time step 3, as shown with the red edges in Fig. 5. The second trick is that the blue nodes in Fig. 5 can be copied from Space 2 to Space 3 at time step 2 since the considered paths until that time

Algorithm 3: LogSumExp of the scores of all possible paths except the best path

```
$C = \{\texttt{B-XXX}, \texttt{I-XXX}, \texttt{E-XXX}, \texttt{S-XXX}, \texttt{O}\}$;
$s = s^{(k)}_{l,j}$; # the start position
$e = e^{(k)}_{l,j}$; # the end position
$\boldsymbol{c}_1(s) = \texttt{B-XXX}$; # the best path
for $i = s + 1; i \leq e - 1; i++$ do
    $\boldsymbol{c}_1(i) = \texttt{I-XXX}$;
$\boldsymbol{c}_1(e) = \texttt{E-XXX}$;
foreach $c \in C$ do
    $\boldsymbol{\alpha}(c) = \boldsymbol{P}^{(k)}_{c,s} + \boldsymbol{A}^{(k)}_{[\texttt{S}],c}$;
$\beta = -\infty$;
for $i = s + 1; i \leq e; i++$ do
    foreach $c \in C$ do
        foreach $c' \in C$ do
            $\boldsymbol{\alpha}_c(c') = \boldsymbol{\alpha}(c') + \boldsymbol{P}^{(k)}_{c,i} + \boldsymbol{A}^{(k)}_{c',c}$;
        if $c == \boldsymbol{c}_1(i)$ then
            foreach $c' \in C \backslash \{\boldsymbol{c}_1(i-1)\}$ do
                $\boldsymbol{\beta}_c(c') = \boldsymbol{\alpha}_c(c')$;
            $\boldsymbol{\beta}_c(\boldsymbol{c}_1(i-1)) =$
              $\beta + \boldsymbol{P}^{(k)}_{c,i} + \boldsymbol{A}^{(k)}_{\boldsymbol{c}_1(i-1),c}$;
    foreach $c \in C$ do
        $\boldsymbol{\alpha}(c) = \mathrm{LogSumExp}(\boldsymbol{\alpha}_c)$;
    $\beta = \mathrm{LogSumExp}(\boldsymbol{\beta}_c)$;
foreach $c \in C \backslash \{\boldsymbol{c}_1(e)\}$ do
    $\boldsymbol{\alpha}(c) += \boldsymbol{A}^{(k)}_{c,[\texttt{E}]}$;
$\boldsymbol{\alpha}(\boldsymbol{c}_1(e)) = \beta + \boldsymbol{A}^{(k)}_{\texttt{E-XXX},[\texttt{E}]}$;
return $\mathrm{LogSumExp}(\boldsymbol{\alpha})$;
```

step are also the same. These two tricks can be applied to other pairs of two adjacent spaces, which relieves the need to separately calculate the summation of the exponential scores for each space. Therefore, the second term of Eq. 5 can be calculated as shown in Algorithm 3.

Thus, we can train a model using the objective function of Eqs. 2, 3, 4, and 5.

3.4 Characteristics

We explain some theoretical characteristics of our method here. Regarding the time complexity of decoder, the worst case for our method is when our decoder narrows down the spans one by one, from n tokens (a whole sentence) to 2 tokens. The time complexity for the worst case is therefore $\mathcal{O}(n + \cdots + 2) = \mathcal{O}(n^2)$ for each entity type, $\mathcal{O}(mn^2)$ in total, where m denotes the number of entity types. However, this rarely happens. The average processing time in the case where our decoding method narrows down spans successfully according to gold labels is $\mathcal{O}(dmn)$, where d is the average number of gold IOBES tags of each entity type assigned to a word. The average numbers calculated from the gold labels of ACE-2004, ACE-2005, and GENIA are 1.06, 1.06, and 1.05, respectively. For these datasets, the expected processing time is nearly equal to $\mathcal{O}(mn)$.

Further, our method focuses on the output layer of neural architectures; therefore our method can be combined with any neural model. While some existing methods have hyperparameters beyond those of the conventional CRF-based model used for flat NER tasks, our method does not. Wang et al. (2018)'s method and Sohrab and Miwa (2018)'s method use a hyperparameter for the maximal length of considered entities. Katiyar and Cardie (2018)'s method uses a threshold, which affects the number of detected entities. These hyperparameters must be tuned depending on datasets. Our method is easy to use from this viewpoint.

We verify the empirical performances of our methods in the successive sections.

4 Experimental Settings

4.1 Datasets

We perform nested entity extraction experiments on ACE-2005 (Doddington et al., 2004) and GENIA (Kim et al., 2003). For ACE-2005, we use the same splits of documents as Lu and Roth (2015), published on their website[5]. For GENIA, we use GENIAcorpus3.02p[6] in which sentences are already tokenized (Tateisi and Tsujii, 2004). Following previous works (Finkel and Manning, 2009; Lu and Roth, 2015), we first split the last 10% of sentences as the test set. Next, we use the first 81% and the subsequent 9% for training and development sets, respectively. We make the same modifications as described by Finkel and Manning (2009) by collapsing all DNA, RNA, and protein subtypes into DNA, RNA, and protein, keeping cell line and cell type, and removing other entity types, resulting in 5 entity types. The statistics of each dataset are shown in Table 1.

[5] http://www.statnlp.org/research/ie

[6] http://www.geniaproject.org/genia-corpus/pos-annotation

	ACE-2005						GENIA					
	Train	(%)	Dev	(%)	Test	(%)	Train	(%)	Dev	(%)	Test	(%)
# documents	370		43		51		-		-		-	
# sentences	(7,214)		(954)		(1,054)		15,022		1,669		1,855	
# mentions	24,818		3,234		3,038		47,027		4,469		5,600	
- 1st level	21,959	(88)	2,900	(90)	2,683	(88)	44,611	(95)	4,239	(95)	5,273	(94)
- 2nd level	2,634	(11)	316	(10)	323	(11)	2393	(5)	230	(5)	327	(6)
- 3rd level	214	(1)	18	(1)	30	(1)	23	(0)	0	(0)	0	(0)
- 4th level	9	(0)	0	(0)	2	(0)	0	(0)	0	(0)	0	(0)
# labels per token (d)	1.06		1.05		1.05		1.05		1.05		1.05	

Table 1: Statistics of the datasets used in the experiments. Note that in ACE-2005, sentences are not originally split. We report the numbers of sentences based on the preprocessing with the Stanford CoreNLP (Manning et al., 2014).

Hyperparameter	**Value**
word dropout rate	0.05
character embedding dimension	25
CNN window size	3
CNN filter number	50
batch size	32
LSTM hidden size	200
LSTM dropout rate	0.2
gradient clipping	5.0

Table 2: Hyperparameters in our experiments.

4.2 Model and Training

In this study, we adopt as baseline a BiLSTM-CRF model, which is widely used for NER tasks (Lample et al., 2016; Chiu and Nichols, 2016; Ma and Hovy, 2016). We apply our usage of CRF to this baseline. We prepare two types of models for our experiments. The first one is the model combined with conventional word embeddings,[7] which is prepared for fairly comparing our method with existing methods. We initialize word embeddings with the pretrained embeddings GloVe (Pennington et al., 2014) of dimension 100 in ACE-2005. For GENIA, we initialize each word with the pretrained embeddings trained on MEDLINE abstracts (Chiu et al., 2016). The initialized word embeddings are fixed during training. The words that do not appear in embedding vocabulary but appear in the training set are initialized with zero vectors and tuned during training. We also adopt a CNN-based character-level representation (Chiu and Nichols, 2016; Ma and Hovy, 2016). The vectors of the word embeddings and the character-level representation are concatenated and then input into the BiLSTM layer. The second model is the model combined with the pretrained BERT model (Devlin et al., 2018).[8] We use the cased version of BERT large model as a contextual word embeddings generator without fine-tuning and stack the BiLSTM layers on top of the BERT model. Both models have 2 BiLSTM hidden layers, and the dimensionality of each hidden unit is 200 in all our experiments. Table 2 lists the hyperparameters used for our experimental evaluations. We adopt AdaBound (Luo et al., 2019) as an optimizer. Early stopping is used based on the performance of development set. We repeat the experiment 5 times with different random seeds and report average and standard deviation of F1-scores on a test set as the final performance.

5 Experimental Results

5.1 Comparison with Existing Methods

Table 3 presents comparisons of our model with existing methods. Note that some existing methods use embeddings of POS tags as an additional input feature whereas our method does not. Our method outperforms the existing methods with 75.92% and 77.25% in terms of F1-score when combined with conventional word embeddings. Especially, our method brings much higher recall values than the other methods. The recall scores are improved by 2.8% and 3.2% on ACE-2005 and GENIA datasets, respectively. These results demonstrate that our training and decoding algorithms are quite effective for extracting nested entities. Moreover, when we use BERT as contextual word embeddings, we achieve an F1-score of 82.70% on ACE-2005. This result shows that BERT is very powerful for this task, too. On the other hand, BERT does not perform well on GENIA. We assume that this is because the domain of

[7] `https://github.com/yahshibu/nested-ner-2019`

[8] `https://github.com/yahshibu/nested-ner-2019-bert`

	ACE-2005			GENIA		
Method	Precision (%)	Recall (%)	F1 (%)	Precision (%)	Recall (%)	F1 (%)
Finkel and Manning (2009)	-	-	-	75.4	65.9	70.3
Muis and Lu (2017)	75.5	51.7	61.3	75.4	66.8	70.8
Katiyar and Cardie (2018)	70.6	70.4	70.5	79.8	68.2	73.6
Ju et al. (2018)[9]	74.2	70.3	72.2	78.5	71.3	74.7
Wang and Lu (2018)	76.8	72.3	74.5	77.0	73.3	75.1
Wang et al. (2018)[10]	74.5	71.5	73.0	78.0	70.2	73.9
Sohrab and Miwa (2018)	-	-	-	**93.2**	64.0	77.1
This work	76.78 ± 0.52	75.08 ± 0.54	75.92 ± 0.23	78.07 ± 0.49	**76.45** ± 0.12	**77.25** ± 0.21
This work [BERT]	**82.98** ± 0.61	**82.42** ± 0.36	**82.70** ± 0.47	76.28 ± 0.90	74.67 ± 0.84	75.46 ± 0.29

Table 3: Main results. The last two lines show our proposed method.

	ACE-2005			GENIA		
	Precision (%)	Recall (%)	F1 (%)	Precision (%)	Recall (%)	F1 (%)
This work	76.78 ± 0.52	75.08 ± 0.54	75.92 ± 0.23	78.07 ± 0.49	76.45 ± 0.12	77.25 ± 0.21
– L	58.94 ± 0.63	75.41 ± 1.07	66.15 ± 0.12	69.95 ± 0.39	78.97 ± 0.38	74.18 ± 0.18
– L&D	76.08 ± 1.22	66.67 ± 0.46	71.06 ± 0.69	79.45 ± 0.48	73.84 ± 0.38	76.54 ± 0.26

Table 4: Results when ablating away the learning (L) and decoding (D) components of our proposed method.

GENIA is quite different from that of the corpus used for training the BERT model. Regardless, it is demonstrated that our method outperforms existing methods.

5.2 Ablation Study

We conduct an ablation study to verify the effectiveness of our learning and decoding methods. We first replace our objective function for training with the standard objective function of the liner-chain CRF. The methods for decoding N-best paths have been well studied because such algorithms have been required in many domains (Soong and Huang, 1990; Kaji et al., 2010; Huang et al., 2012). However, we hypothesize that our learning method, as well as our decoding method, helps to improve performance. That is why we first remove only our learning method. Then, we also replace our decoding algorithm with the standard decoding algorithm of the linear-chain CRF. It is equivalent to preparing the conventional CRF for each entity type separately.

The results are shown in Table 4. They demonstrate that introducing only our decoding algorithm surely brings high recall scores but hurts precision. This suggests that our learning method should be necessary for achieving high precision. Besides, removing the decoding algorithm results in lower recall. This is natural because it does not intend to find any nested entity after extracting outermost entities. Thus, it is demonstrated that both our learning and decoding algorithms contribute much to good performance across these datasets.

	ACE-2005		GENIA	
Level	Recall (%)	Num.	Rcall (%)	Num.
1st	75.79 ± 0.66	2,683	78.75 ± 0.12	5,273
2nd	70.71 ± 0.75	323	39.39 ± 0.94	327
3rd	60.00 ± 3.65	30	-	0
4th	50.00 ± 0.00	2	-	0

Table 5: Recall scores for gold annotations of each level.

	ACE-2005		GENIA	
Level	Precision (%)	Num.	Precision (%)	Num.
1st	78.43	2,582	79.84	5079
2nd	64.90	359	55.67	397
3rd	59.62	52	83.33	6
4th	77.78	9	-	0

Table 6: Precision scores for predictions of each level of one trial.

5.3 Analysis of Behavior

To further understand how our method handles nested entities, we investigate the performances for entities of each level. Table 5 shows the recall scores for gold entities of each level when using conventional word embeddings. Among all levels, our model results in the best performance at the 1st level that consists of only gold outermost entities. The deeper a level, the lower recall scores. On the other hard, Table 6 shows the precision scores

[9]Note that in ACE-2005, Ju et al. (2018) did their experiments with a different split from Lu and Roth (2015) that we follow.

[10]Wang et al. (2018) did not report precision and recall scores. Instead of Wang et al. (2018), Wang and Lu (2018) reported the scores for the model of Wang et al. (2018).

Method	# tokens per second
Wang and Lu (2018)	459
Wang et al. (2018)	2,059
This work	5,594

Table 7: Decoding speed on ACE-2005.

Method	P (%)	R (%)	F1 (%)
Muis and Lu (2017)	72.7	58.0	64.5
Katiyar and Cardie (2018)	72.3	66.8	69.7
Wang and Lu (2018)	78.0	72.4	75.1
Wang et al. (2018)[11]	74.9	71.8	73.3
This work	79.27	75.04	77.10
This work [BERT]	**83.73**	**81.91**	**82.81**

Table 8: Comparison on ACE-2004. The last two lines show our proposed method.

for predicted entities in each level of one trial on each dataset. Because the number of levels in the predictions vary between trials, taking macro average of precision scores over multiple trials is not representative. Therefore, we show only the precision scores from one trial in Table 6. The precision score for the 4th level on ACE-2005 is as high as or higher than those of other levels. Precision scores are less dependent on level. This tendency is also shown in other trials.

5.4 Running Time

We examine the decoding speed in terms of the number of words processed per second. We compare our model with the models of Wang and Lu (2018) and Wang et al. (2018) using their implementations. The used platform is the same (PyTorch), and the machine on which we run them is also the same (CPU: Intel i7 2.7 GHz). Results on ACE-2005 are listed in Table 7. Our model turns out to be faster than theirs, showing its scalability.

5.5 Comparison on ACE-2004

We compare our method with existing methods also on the ACE-2004 dataset. We use the same splits as Lu and Roth (2015). The setups are the same as those of our experiment on ACE-2005. Table 8 shows the results. As shown, our method significantly outperforms existing methods. Note that most of them use embeddings of POS tags as an additional input feature whereas our method does not.

[11]Wang et al. (2018) did not report precision and recall scores. Instead of Wang et al. (2018), Wang and Lu (2018) reported the scores for the model of Wang et al. (2018).

5.6 Error Analysis

We manually scan the test set predictions on ACE-2005. We find out that many of the errors can be classified into two types.

The first type is partial prediction error. Given the following sentence: "*Let me set aside the hypocrisy of a man who became president because of a lawsuit trying to eliminate everybody else's lawsuits, but instead focus on his own experience*". Our model predicts "*a man who became president*" or "*a man who became president because of a lawsuit trying to eliminate everybody else's lawsuits*" as PER (Person) while the annotation marks "*a man who became president because of a lawsuit*". It is difficult for our model to extract the proper spans of clauses or independent sentences that contain numerous modifiers.

The second type is error derived from pronominal mention, as reported by Katiyar and Cardie (2018). Consider the following example: "*They roar, they screech.*". These "*They*"s refer to "*tanks*" in another sentence of the same document and are indeed annotated as VEH (Vehicle). Our model fails to detect these pronominal mentions or wrongly labels them as PER. Document context should be taken into consideration to solve this problem.

6 Conclusion

We propose learning and decoding methods for extracting nested entities. Our decoding method iteratively recognizes entities from outermost ones to inner ones in an outside-to-inside way. It recursively searches a span of each extracted entity for nested entities with second-best sequence decoding. We also design an objective function for training that ensures decoding. Our method has no hyperparameters beyond those of conventional CRF-based models. Our method achieve 82.81%, 82.70%, and 77.25% on ACE-2004, ACE-2005, and GENIA datasets, respectively, in terms of F1-score. We also demonstrate that our decoding method is faster than existing methods.

For future work, one interesting direction is joint modeling of NER with entity linking or coreference resolution. Durrett and Klein (2014), Luo et al. (2015), and Nguyen et al. (2016) demonstrated that leveraging mutual dependency of the NER, linking, and coreference tasks could boost each performance. We would like to address this issue while taking nested entities into account.

References

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.

Beatrice Alex, Barry Haddow, and Claire Grover. 2007. Recognising nested named entities in biomedical text. In *Biological, translational, and clinical language processing*, pages 65–72, Prague, Czech Republic. Association for Computational Linguistics.

Kate Byrne. 2007. Nested named entity recognition in historical archive text. In *International Conference on Semantic Computing (ICSC 2007)*, pages 589–596.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136.

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.

Jason P.C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *Computing Research Repository*, arXiv:1810.04805. Version 1.

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).

Greg Durrett and Dan Klein. 2014. A joint model for entity analysis: Coreference, typing, and linking. *Transactions of the Association for Computational Linguistics*, 2:477–490.

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.

Zhiheng Huang, Yi Chang, Bo Long, Jean-Francois Crespo, Anlei Dong, Sathiya Keerthi, and Su-Lin Wu. 2012. Iterative Viterbi A* algorithm for k-best sequential decoding. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 611–619, Jeju Island, Korea. Association for Computational Linguistics.

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.

Nobuhiro Kaji, Yasuhiro Fujiwara, Naoki Yoshinaga, and Masaru Kitsuregawa. 2010. Efficient staggered decoding for sequence labeling. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 485–494, Uppsala, Sweden. Association for Computational Linguistics.

Arzoo Katiyar and Claire Cardie. 2018. Nested named entity recognition revisited. 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 861–871, New Orleans, Louisiana. Association for Computational Linguistics.

J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. 2003. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl_1):i180–i182.

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.

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.

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.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. 2019. Adaptive gradient methods with dynamic bound of learning rate. *Computing Research Repository*, arXiv:1902.09843. Version 1.

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.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

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.

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticæ Investigationes*, 30(1):3–26.

Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. 2016. J-NERD: Joint named entity recognition and disambiguation with rich linguistic features. *Transactions of the Association for Computational Linguistics*, 4:215–229.

Jeffrey Pennington, Richard ocher, 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.

Nambirajan Seshadri and Carl-Erik W. Sundberg. 1994. List Viterbi decoding algorithms with applications. *IEEE Transactions on Communications*, 42(234):313–323.

Mohammad Golam Sohrab and Makoto Miwa. 2018. Deep exhaustive model for nested named entity recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2843–2849, Brussels, Belgium. Association for Computational Linguistics.

Frank K. Soong and Eng-Fong Huang. 1990. A Tree.Trellis based fast search for finding the n best sentence hypotheses in continuous speech recognition. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2670–2680, Copenhagen, Denmark. Association for Computational Linguistics.

Yuka Tateisi and Jun-ichi Tsujii. 2004. Part-of-speech annotation of biology research abstracts. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal. European Language Resources Association (ELRA).

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.

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.

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.