FIXED SIZE ORDINALLY-FORGETTING ENCODING AND ITS APPLICATIONS

MINGBIN XU

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
2017

Abstract

In this thesis, we propose the new Fixed-size Ordinally-Forgetting Encoding (FOFE) method, which can almost uniquely encode any variable-length sequence of words into a fixed-size representation. FOFE can model the word order in a sequence using a simple ordinally-forgetting mechanism according to the positions of words. We address two fundamental problems in natural language processing, namely, Language Modeling (LM) and Named Entity Recognition (NER).

- We have applied FOFE to FeedForward Neural Network Language Models (FFNN-LMs). Experimental results have shown that without using any recurrent feedbacks, FOFE-FFNN-LMs significantly outperform not only the standard fixed-input FFNN-LMs but also some popular Recurrent Neural Network Language Models (RNN-LMs).

- Instead of treating NER as a sequence labeling problem, we propose a new local detection approach, which relies on FOFE to fully encode each sentence

fragment and its left/right contexts into a fixed-size representation. This local detection approach has shown many advantages over the traditional sequence labeling methods. Our method has yielded pretty strong performance in all tasks we have examined.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Hui Jiang, for his valuable cultivation throughout my study at York University. He is always patient and helps me tackle myriads of academic problems. His enlightenment and support are highly appreciated. I am also sincerely grateful to my committee member, Prof. Aijun An, for reviewing my thesis and providing me with her advice.

Numerous individuals supported my research in various ways.

Special thanks to Jing Huang, the Principal Research Scientist of Visa Research, who granted me the opportunity to learn from and collaborate with the smartest brains of Apple's Siri team. Special thanks to Xiaochuan Niu, the Principal Research Scientist of Apple Siri, who provided me with endless guidance in engineering and optimization.

I would also like to thank my best friends Yiyun Huang and Chengjie Wang. I am gratefully indebted to their comments and suggestions on my work. Thank them for giving different perspectives of my work.

I would also like to acknowledge my senior lab mates Shiliang Zhang and Quan Liu for sharing their experience and insight with me in the early stage of my research.

Financial support from York University is also heartily acknowledged.

Last but not least, I must express my profound gratitude to my parents and to my partner for providing me with continuous encouragement throughout my years of study at York. This accomplishment would not have been possible without them. Thank you!

Table of Contents

Abstract **ii**

Acknowledgements **iv**

Table of Contents **vi**

List of Tables **x**

List of Figures **xi**

Abbreviations **xii**

1 Introduction **1**

1.1 Motivation 1

1.2 Contribution and Outline of the Thesis 2

2 Literature Review **5**

2.1 Deep Learning 5

2.1.1 Feed-Forward Neural Network 5
2.1.2 Recurrent Neural Network 7
2.2 Vector Representation of Words 8
2.2.1 Distributed Word Embedding 8
2.2.2 Character Word Embedding 9
2.3 Two Fundamental Problems in NLP 11
2.3.1 Language Modeling 11
2.3.2 Named Entity Recognition and Mention Detection 14

3 Fixed-size Ordinally Forgetting Encoding 17

3.1 One-Hot Encoding 17
3.2 Bag of Words 18
3.3 Term Frequency - Inverse Document Frequency 19
3.4 Fixed-size Ordinally Forgetting Encoding 19
3.4.1 Definition 20
3.4.2 Uniqueness 22

4 FOFE in Language Modeling 25

4.1 FOFE Language Model 25
4.2 Experiment Results 29
4.2.1 Penn TreeBank 30

4.2.2 Large Text Compression Benchmark 35

4.2.3 Google 1 Billion Benchmark 36

5 FOFE in Entity Discovery **39**

5.1 Local Detection 39

5.2 Feature Extraction 40

5.2.1 Word-level Features 41

5.2.2 Character-level Features 42

5.3 Training and Decoding Algorithm 43

5.4 Second-Pass Augmentation 47

5.5 Experiment Results 54

5.5.1 CoNLL 2003 NER task 55

5.5.2 KBP2015 EDL Task 58

5.5.3 KBP2016 EDL task 63

6 Conclusion **66**

6.1 Conclusion 66

6.2 Future Works 67

7 Accomplishment during MSc Study **68**

7.1 Publication 68

7.2 Patents 70

7.3 Awards 70

Bibliography **71**

List of Tables

3.1 Vocab of Size 7 21

3.2 Partial Encoding of $w_6, w_4, w_5, w_0, w_5, w_4$ 21

4.1 PPL on PTB for various LMs. 32

4.2 PPL on LTCB for various LMs 34

4.3 PPL on GBW on various LMs 38

5.1 Effect of various FOFE feature combinations on the CoNLL2003 . . 56

5.2 Data distribution of CoNLL2003 57

5.3 Comparison between neuro-NER on CoNLL2003 59

5.4 Number of Documents in KBP2015 60

5.5 Evaluation of EDL in KBP2015 61

5.6 Official Evaluation of EDL in KBP2016 62

5.7 Number of Documents in KBP2016 63

5.8 English-only Official Performance by Different Dataset Combinations 63

List of Figures

4.1 Diagram of 1st-order FOFE-FFNN-LM. 26

4.2 Diagram of 2nd-order FOFE-FFNN-LM. 27

4.3 Effectiveness of Forgetting Factor 31

5.1 Illustration of the local detection approach 40

5.2 Illustration of the 2nd-pass Augmentation 54

Abbreviations

AI	Artificial Intelligence
B-LSTM	Bidirectional Long Short-Term Memory
BP	Back-Propagation
BPTT	Back-Propagation Through Time
CNN	Convolutional Neural Network
CRF	Conditional Random Field
FFNN	FeedForward Neural Network
FFNN-LM	FeedForward Neural Network Language Model
FOFE	Fixed-size Oridinally Forgetting Encoding
FOFE-FFNN-LM	FOFE-based FFNN-LM
GBW	Google 1 Billion Word benchmark

LM	Language Modeling
LSTM	Long Short-Term Memory
LTCB	Large Text Compression Benchmark
MD	Mention Detection
NER	Named Entity Recognition
NCE	Noise Contrastive Estimation
NLP	Natural Language Processing
NN	Neural Network
PPL	PerPLexity
PTB	Penn TreeBank
RNN	Recurrent Neural Network
RNN-LM	Recurrent Neural Network Language Model
SGD	Stochastic Gradient Descent
SGNS	SkipGram with Negative Sampling

1 Introduction

1.1 Motivation

Artificial Intelligence (AI) was born in the 1950s. AI system in the early stage only involved hard-coded rules crafted by experts, which did not qualify as a learning process. However, a good set of rules is intractable as the problem grows. Later, machine learning arose as a sub-field of AI. Machine learning methods excel at problem where a good set of rules is hard to define. A machine learning system is trained rather than programmed. It is first presented with many examples, and then searches for an appropriate statistical structure which fits the presented examples and generalizes to unseen examples. Machine learning is a general field encompassing deep learning. Specifically, deep learning emphasizes on learning layers of increasingly meaningful representations. These layered representations are learned via a family of models called Neural Network (NN). Due to the advance in hardware, deep learning quickly becomes the most popular and most successful field in AI.

Natural Language Processing (NLP) is a field that studies how to understand and produce natural language using computers. Natural languages are hierarchical in the sense that a word consists of multiple characters, a phrase consists of multiple words, a sentence consists of multiple phrases, and ultimately sentences convey ideas. Natural languages, albeit compositional, are not intended to be fit into a finite set mathematically. The way in which characters and words are combined to form meaningful sentence is infinite and thus is impossible to be enumerated by rules. Deep learning is the most promising statistic approach bypass the rigid rules of natural languages. Once we are able to extract structured numerical data from natural language, we can take advantage of deep learning and rely on statistical relationships between characters and words.

The central problem in NLP and deep learning is that of learning useful representations of the input data. Such representations should get us closer to the expected output. In this research, we propose a novel representation of natural language called Fixed-size Ordinally-Forgetting Encoding (FOFE) and are interested in solving problems in NLP.

1.2 Contribution and Outline of the Thesis

In this thesis, we explore an alternative approach of sequence modeling. We propose to use Fixed-size Ordinally Forgetting Encoding (FOFE). We address two fundamental problems in NLP, Language Modeling (LM) and Named Entity Recognition (NER) & Mention Detection (MD), by applying FOFE. Our contributions are summarized as follows:

- We propose a novel sequence modeling method, FOFE. FOFE is able to encode any sequence of **variable length** into a **fixed-size** vector. We additionally prove that FOFE is a lossless representation, which leads to theoretical guarantees of its modeling power.

- We apply FOFE on LM. Extensive experiments are conducted and the time performance and scalability are carefully evaluated. It achieves strong performance with great parallelism. To the best of our knowledge, this is the first piece of work of sequence modeling without any recurrent feedback in deep learning.

- In light of human perception of NER & MD, we also propose a local detection algorithm for NER & MD. Unlike previous work whose context is limited, our local detection treats the entire sentence as context. Contextual information is losslessly encoded by FOFE. It ranked 2nd place and is the best single-model system in the EDL track of KBP2016 [25] [1].

The rest of the thesis is organized as the following: Chapter 2 reviews the related

[1]The contest is detailed in Section 5.5.3.

work . Chapter 3 presents our novel approach of sequence representation, namely, Fixed-size Ordinally-Forgetting Encoding (FOFE). Chapter 4 and Chapter 5 demonstrates how FOFE collaborates with deep learning to achieve performance on par with state of the art in LM and NER & MD. Chapter 6 concludes the thesis.

2 Literature Review

2.1 Deep Learning

2.1.1 Feed-Forward Neural Network

It is well known that Neural Network (NN) is a universal approximator under certain conditions [23]. A Feed-Forward Neural Network (FFNN) is a weighted graph with a layered architecture. Each layer is composed of several nodes. Successive layers are fully connected. Each node applies a function on the weighted sum of the lower layer. The values of the first layer is user-input. Formally, let

- N_n be the number of nodes in the n-th layer,
- $x_n \in \mathbb{R}^{N_n}$, where $x_{n,j}$ $(1 \leq j \leq N_n)$ denotes the value of the j-th node in the n-th layer,
- $W^n \in \mathbb{R}^{N_n \times N_{n+1}}$, where $W^n_{i,j}$ $(1 \leq i \leq N_n,\ 1 \leq j \leq N_{n+1})$ denotes the weight of the connection from $x_{n,i}$ to $x_{n+1,j}$, and

- $b^n \in \mathbb{R}^{n+1}$ be the bias that shifts the activation function.

Then

$$z_{n+1,j} = \sum_i W^n_{i,j} x_{n,i} + b^n_j \tag{2.1}$$

$$x_{n+1,j} = \sigma\left(z_{n+1,j}\right) \tag{2.2}$$

where σ is the activation function, usually chosen to be sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

or Rectified Linear Unit (ReLU) [15]:

$$\sigma(x) = \max(0, x). \tag{2.4}$$

For classification tasks, the outputs are normalized into a probability distribution by the so-called *softmax* function, where the i-th node is computed as follow:

$$\sigma(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}. \tag{2.5}$$

An NN can learn by adjusting its weights in a process called Back-Propagation (BP). Suppose that we have already calculated the outputs given by an NN for any input. Let $E(y,t)$ be an error metric that measures how incorrect the output y is with respect to the expected target output t. For each weight in NN, we may calculate:

$$\begin{aligned} \frac{\partial E}{\partial W^n_{i,j}} &= \frac{\partial E}{\partial \sigma} \frac{\partial \sigma}{\partial z_{n+1,j}} \frac{\partial z_{n+1,j}}{\partial W^n_{i,j}} \\ &= \frac{\partial E}{\partial \sigma} \frac{\partial \sigma}{\partial z_{n+1,j}} x_{n,i}. \end{aligned} \tag{2.6}$$

Each weight may be adjusted to slowly reduce this error for each training example, and hence the NN learns to fit the input and the output. This is accomplished by the following update rule, where α is called the learning rate:

$$W_{i,j}^n := W_{i,j}^n - \alpha \frac{\partial E}{\partial W_{i,j}^n} \tag{2.7}$$

The learned NN may be used to generalize and extrapolate to new inputs that have not been seen during training. All the equations above can be equivalently expressed in matrix operations for better efficiency:

$$\begin{aligned} z_{n+1} &= W^n x_n + b^n \\ x_{n+1} &= \sigma(z_{n+1}) \\ \frac{\partial E}{\partial W^n} &= \frac{\partial E}{\partial z_{n+1}} \frac{\partial z_{n+1}}{\partial W^n} x_n \\ W^n &:= W^n - \alpha \frac{\partial E}{\partial W^n}. \end{aligned} \tag{2.8}$$

2.1.2 Recurrent Neural Network

The major deficiency of FFNN is its incapability of modeling input of varying size. For example, sentences in natural language are of arbitrary lengths. Recurrent Neural Network (RNN) addresses this issue by recurrent connections. Let's assume the same notation in Section 2.1.1. The n-th layer, if recurrent, is additionally equipped with another parameter matrix W_r^n, and its input and output are attached with an timestep, denoted as x_n^t and x_{n+1}^t for timestep t. The output of such layers is redefined as:

$$x_{n+1}^{t} = \sigma(x_{n}^{t} W^{n} + x_{n+1}^{t-1} W_{r}^{n} + b^{n}) \tag{2.9}$$

RNNs are learned by an algorithm called BACK-PROPAGATION THROUGH TIME (BPTT) [52] due to the internal recurrent feedback cycles. BPTT works by unfolding through time. The unfolded network is acyclic and updated similar to FFNN.

BPTT significantly increases the computational complexity of the learning algorithms and it may cause many problems in learning, such as gradient vanishing and exploding [4]. More recently, some new architectures have been proposed to solve these problems. For example, the LONG SHORT-TERM MEMORY (LSTM) [22] is an enhanced architecture to implement the recurrent feedbacks using various learnable gates, and it has obtained promising results on handwriting recognition [19] and sequence modeling [18].

2.2 Vector Representation of Words

2.2.1 Distributed Word Embedding

Curse of dimensionality and lack of semantics demand a compact representation. The construction of low-dimensional word vectors is inspired by the linguistic concept of distributional hypothesis, which claims that words appear in the similar context share similar meanings [21]. The most well-known approach of generating such word vectors is introduced by [41], which uses the SKIP-GRAM model trained with STOCHASTIC GRADIENT DESCENT (SGD) and NEGATIVE SAMPLING (NS), named as SGNS.

SGNS maintains two matrices, $W_{word} \in \mathbb{R}^{|V| \times n}$ and $W_{context} \in \mathbb{R}^{|V| \times n}$, where n is the desired number of dimension of the compact vectors. The i-th row of W_{word} and the i-th row of $W_{context}$ correspond to the i-th word in V. Given a sentence with N words $w_1, w_2, ..., w_N$, and a central word w_i, $1 \leq i \leq N$, the rest words in $C = \{w_1, w_2, ..., w_{i-1}\} \cup \{w_{i+1}, w_{i+2}, ..., w_N\}$ are treated as the context of w_i. SGNS tries to maximize the dot product of $W_{word}[w_i]$ and $W_{context}[c]$ if $c \in C$ and minimize the dot product of $W_{word}[w_i]$ and $W_{context}[c]$ if $c \in V - C$. SGNS iterates all the observed pairs in the corpus and learns W_{word} and $W_{context}$ by SGD. W_{word} contains the compact word vectors.

2.2.2 Character Word Embedding

CONVOLUTIONAL NEURAL NETWORKS (CNN) is known to be effective in NLP, and have been widely used as character-level models [30]. The goal of character word embedding is on one hand to alleviate information loss from OUT-OF-VOCABULARY (OOV) issue, and on the other hand to take into account internal structure of words. The latter is especially useful or morphologically rich languages.

Let C denote the set of possible characters, and D denote the dimensionality of character embeddings. A matrix $\boldsymbol{M} \in \mathbb{R}^{|C| \times D}$ is randomly initialized, where the i-th row denotes the vector representation of the i-th character in C. Given a word or phrase whose spelling is $[c_1, c_2, c_3, ..., c_L]$, a matrix $\boldsymbol{C} \in \mathbb{R}^{L \times D}$ is constructed, where the j-th row is a copy of the row in $\boldsymbol{M}$ corresponding to c_j. $\boldsymbol{C}$ can be viewed as a single-channel image. Let $\boldsymbol{F} \in \mathbb{R}^{h \times D}$ be a feature map to be learned, where h denotes the height of feature maps. D sometimes is called the width of the feature map. An intermediate vector $\boldsymbol{v}$ of $l - h + 1$ elements is generated after $\boldsymbol{F}$ sweeps $\boldsymbol{C}$. Each component v_k in $\boldsymbol{v}$, is computed as:

$$v_k = \sigma(Trace(\boldsymbol{F}\boldsymbol{C}[k : k + h])) \tag{2.10}$$

where σ is either sigmoid (Eq. 2.3) or ReLU (Eq. 2.4). The output y of this feature map is given by:

$$y = \max(v_1, v_2, ..., v_{l-h+1}) \tag{2.11}$$

If there are N groups of feature maps, each of which has n_1, n_2, n_3, ... , $n_{|N|}$ feature maps respectively, following Eqs. (2.10) and (2.11), the final representation from the character CNN for this word or fragment is a vector of length $\sum_{i=1}^{|N|} n_i$.

2.3 Two Fundamental Problems in NLP

2.3.1 Language Modeling

Language Modeling (LM) plays an important role in many applications like speech recognition, machine translation, information retrieval and nature language understanding. The goal of LMs is to compute a probability for a sequence of tokens. Syntactically and semantically sound sentences are assigned high scores while invalid and silly sentences are given low scores, for example,

$$P(\text{``}\textit{The bottle is small}\text{''''}) > P(\text{``}\textit{The bottle is mall}\text{''})$$

The probability of a sentence with N words $w_1, w_2, ..., w_N$ can be broken apart as:

$$P(w_1, w_2, ..., w_N) = \prod_{i=1}^{N} P(w_i | w_1, w_2, ..., w_{i-1}) \tag{2.12}$$

where each term is conditioned on all previous words. Estimating $P(w_i | w_1, w_2, ..., w_{i-1})$ is difficult. A certain degree of independence is assumed and Markov property [37] is adopted. That is, each term is conditioned on a window of n previous words instead:

$$P(w_1, w_2, ..., w_N) = \prod_{i=1}^{N} P(w_i | w_{i-n}, w_{i-n+1}, ..., w_{i-1}). \tag{2.13}$$

A great deal of effort has been devoted to the estimation of $P(w_i | w_1, w_2, ..., w_{i-1})$ and $P(w_i | w_{i-n}, w_{i-n+1}, ..., w_{i-1})$. Traditionally, the back-off n-gram models [29, 31] are the standard approach to LM. Recently, NNs have been successfully applied to

LM, yielding the state-of-the-art performance in many tasks. In Neural Network Language Models (NNLM), FFNN and RNN [14] are two popular architectures. The basic idea of NNLMs is to use word vectors to project discrete words into a continuous space and estimate word conditional probabilities in this space, which may be smoother to better generalize to unseen contexts. Feed-Forward Neural Network Language Models (FFNN-LM) [2, 3] usually use a limited history within a fixed-size context window to predict the next word. Recurrent Neural Network Language Models (RNN-LM) [39, 42] adopt a time-delayed recursive architecture for the hidden layers to memorize the long-term dependency in language. Therefore, it is widely reported that RNN-LMs outperform FFNN-LMs in LM.

The most commonly used metric to evaluate the performance of LM is PerPLexity (PPL) over unseen sentences. Give a corpus c of n words $w_1, w_2, ..., w_n$, a language model m assign a probability to each word in the corpus. PPL of m is defined as

$$PPL(m, c) = 2^{-\frac{1}{n}\sum_{i=1}^{n} log_2 m(w_i)} \tag{2.14}$$

A lower PPL in general indicates a better fit. A good language model assigns high probabilities to the patterns in the corpus, reflecting the language usage of the corpus.

NNLM can be unbearably slow because of *softmax*. Noise Contrastive Es-

TIMATION (NCE) [20] is a self-normalized approach that approximates *softmax* over large vocabulary efficiently. NCE reduces probability estimation to binary classification. Let's assume the following notations:

- $p_\theta(w,c)$: probability of word w given the context c, modeled by parameter set θ, and
- $p_{uni}(w)$: unigram probability of word w, i.e., $\dfrac{count(w)}{\#words}$.

If a word w is sampled from the corpus with auxiliary label $D = 1$, and k other words are sampled from p_{uni} with auxiliary label $D = 0$. The probability of D is a mixture of $p_\theta(w,c)$ and $p_{uni}(w)$:

$$\begin{aligned} p(D=0|c,w) &= \frac{k \times p_{uni}(w)}{p_\theta(w,c) + k \times p_{uni}(w)} \\ p(D=1|c,w) &= \frac{p_\theta(w,c)}{p_\theta(w,c) + k \times p_{uni}(w)} \end{aligned} \tag{2.15}$$

NCE further assumes that when θ is large enough, the summation term of *softmax* can be estimated by a scalar constant Z. The choice of Z varies on different corpus. The binary classification problem shares the same parameters θ with the LM that we desired. It is trained to maximize the log-likelihood of D with k negative noises.

$$\begin{aligned} L_{NCE_k} &= \sum_{(w,c)\in corpus} (log\ p(D=0|c,w) + k\mathbb{E}_{\overline{w}} log\ p(D=0|c,\overline{w})) \\ &\approx \sum_{(w,c)\in corpus} (log\ p(D=0|c,w) + \sum^{k} log\ p(D=0|c,\overline{w})) \end{aligned} \tag{2.16}$$

The expectation term in Equation 2.16 is replaced by its Monte Carlo approximation.

2.3.2 Named Entity Recognition and Mention Detection

Named Entity Recognition (NER) and Mention Detection (MD) are very challenging tasks in NLP, laying the foundation of almost every NLP application. NER and MD are tasks of identifying entities (named and/or nominal) from raw text, and classifying the detected entities into one of the pre-defined categories such as person (PER), organization (ORG), location (LOC), etc. Some tasks focus on named entities only, for example,

> $[S.E.C.]_{ORG}$ chief $[Mary\ Shapiro]_{PER}$ left $[Washington]_{LOC}$ in December .

while the others also detect nominal mentions. which are important for other NLP tasks such as co-reference resolution.

> $[Mark]_{PER}$ and his closest $[friend]_{PER\_N}$ $[Scarlet]_{PER}$, a cello $[player]_{PER\_N}$, joined the same music $[company]_{ORG\_N}$.

Moreover, nested mentions may need to be extracted too. For example,

> He used to study in $[University\ of\ [Toronto]_{LOC}]_{ORG}$.

where *Toronto* is a LOC entity, embedded in another longer ORG entity *University of Toronto*.

Similar to many other NLP problems, NER and MD are formulated as a sequence labeling problem, where a tag is sequentially assigned to each word in the input sentence. It has been extensively studied in the NLP community [5]. The core problem is to model the conditional probability of an output sequence given an arbitrary input sequence. Many hand-crafted features are combined with statistical models, such as CONDITIONAL RANDOM FIELDS (CRFs) [43], to compute conditional probabilities. More recently, some popular NNs, including CNNs and RNNs, are proposed to solve sequence labeling problems. In the inference stage, the learned models compute the conditional probabilities and the output sequence is generated by the Viterbi decoding algorithm [51].

It has been a long history of research involving NN. The success of word embedding [41, 35] encourages researchers to focus on machine-learned representation instead of heavy feature engineering in NLP. Using word vectors as the typical feature representation for words, NNs become competitive to traditional approaches in NER. Many NLP tasks, such as NER, chunking and part-of-speech (POS) tagging can be formulated as sequence labeling tasks. In [9], deep CNN and CRF are used to infer NER labels at a sentence level, where they still use many hand-crafted features to improve performance, such as capitalization features explicitly defined based on first-letter capital, non-initial capital and so on.

Recently, RNNs have demonstrated the ability in modeling sequences [17]. Huang [24] built on the previous CNN-CRF approach by replacing CNNs with BIDIRECTIONAL LONG SHORT-TERM MEMORY (B-LSTM). Though they have reported improved performance, they employ heavy feature engineering in that work, most of which is language-specific. There is a similar attempt in [46] with full-rank

CRF. CNNs are used to extract character-level features automatically in [13].

Gazetteer is a list of names grouped by the pre-defined categories. Gazetteer is shown to be one of the most effective external knowledge sources to improve NER performance [47]. Thus, gazetteer is widely used in many NER systems. In [8], state-of-the-art performance on a popular NER task, i.e., CoNLL2003 [2], is achieved by incorporating a large gazetteer. Different from previous ways to use a set of bits to indicate whether a word is in gazetteer or not, they have encoded a match in BIOES (Begin, Inside, Outside, End, Single) annotation, which captures positional information.

The quality of a NER system is measured by F1 score, the balanced harmonic mean of precision and recall:

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}. \tag{2.17}$$

Precision is the ratio of the number of system-predicted entities that match the ground truth to the number of system prediction. Recall is similarly the number of system-predicted entities that match the ground truth to the number of entities from ground truth. Only exact span constitutes matches. Wrong span with correct predicted type and correct span with wrong predicted type are not given partial credits, and thus don't contribute to F_1.

[2]See Section 5.5.1 for more details

3 Fixed-size Ordinally Forgetting Encoding

In order to plug in statistical tools, characters and words must be encoded into a vector space. Let's exemplify this at word level in English. It could be easily generalized to character and phrase. Suppose that we are interested in the most representative subset V for words in English, usually chosen according to frequency.

3.1 One-Hot Encoding

Words build sentences. A representation of words lays the foundation of representing sentences. One-hot vector is the most straightforward representation. A fixed integer id is assigned to each word occurring in the corpus. Each word is an $\mathbb{R}^{1\times|V|}$ vector with all 0s and one 1 at the index of the word in V. Word vectors in this type of encoding could appear as following:

$$\text{“}the\text{”} = [1, 0, 0, 0, ..., 0, 0]$$

$$\text{“,”} = [0, 1, 0, 0, ..., 0, 0]$$

$$\text{“.”} = [0, 0, 1, 0, ..., 0, 0]$$

$$\text{“}to\text{”} = [0, 0, 0, 1, ..., 0, 0]$$

$$...$$

$$\text{“}rereleased\text{”} = [0, 0, 0, 0, ..., 1, 0]$$

$$\text{“}unearths\text{”} = [0, 0, 0, 0, ..., 0, 1]$$

They are sparse vectors suffering from the curse of dimensionality [1]. Each additional dimension doubles the computational power required. Each word is treated as an independent entity. As a result, the semantics behind words is lost. This representation does not capture the similarity between words.

3.2 Bag of Words

The most intuitive way of building the representation of a sentence or a word sequence is to add up the one-hot encoding of the words of which it composed. Effectively, it counts the number of times each word appears. This approach is called Bag of Words (BoW). BoW simplifies the problem at the cost of ignoring the context of words. It may fail badly in specific cases. For example, consider the sentences “I love apple but not banana” and “I love banana but not apple”. They express the opposite preferences to fruits while sharing the same BoW representation.

3.3 Term Frequency - Inverse Document Frequency

In natural language, some words are significantly more present than the others, (e.g. “the”, “a” in English), and thus not very informative. If the direct count from BoW is used, those frequent terms will shadow the importance of rare but more interesting terms. Term Frequency - Inverse Document Frequency (TF-IDF) is an algorithm that re-weights importance. TF-IDF weights each word by it frequency in the sentence and the logarithm of the reciprocal of its frequency in the corpus. Similar to BoW, it assumes word counts provide independent importance and therefore does not respect the semantics between words.

3.4 Fixed-size Ordinally Forgetting Encoding

FFNN is a powerful computation model. However, it requires fixed-size inputs and lacks the ability of capturing long-term dependency. Because most NLP problems involves variable-length sequences of words, RNNs/LSTMs are more popular than FFNNs in dealing with these problems. The Fixed-size Ordinally-

Forgetting Encoding (FOFE), originally proposed in [55, 56], nicely overcomes the limitations of FFNNs. The intuition behind this idea is that the closer words are more related to local decisions. FOFE adopts this concept and re-weights each word in the history from new to old in a exponentially decaying fashion. More importantly, it can uniquely and losslessly encode a variable-length sequence of words into a fixed-size representation.

3.4.1 Definition

Give a vocabulary V, each word can be represented by a one-hot vector. FOFE mimics bag-of-words (BOW) but incorporates a forgetting factor to capture positional information. It encodes any sequence of variable length composed by words in V. Let $S = w_1, w_2, w_3, ..., w_T$ denote a sequence of T words from V, and $\boldsymbol{e_t}$ be the one-hot vector of the t-th word in S, where $1 \leq t \leq T$. The FOFE of each partial sequence $\boldsymbol{z_t}$ from the first word to the t-th word is recursively defined as:

$$\boldsymbol{z_t} = \begin{cases} \mathbf{0}, & \text{if } t = 0 \\ \alpha \cdot \boldsymbol{z_{t-1}} + \boldsymbol{e_t}, & \text{otherwise} \end{cases} \tag{3.1}$$

where the constant α is called forgetting factor, and it is picked between 0 and 1 exclusively. Obviously, the size of $\boldsymbol{z_t}$ is $|V|$, and it is irrelevant to the length of original sequence T. An example is included in Table 3.1 and Table 3.2 of how to encode the sequence $[w_6, w_4, w_5, w_0, w_5, w_4]$ with the vocabulary $\{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$.

WORD	1-HOT
w_0	1000000
w_1	0100000
w_2	0010000
w_3	0001000
w_4	0000100
w_5	0000010
w_6	0000001

Table 3.1: Vocab of Size 7

PARTIAL SEQUENCE	FOFE
w_6	$0, 0, 0, 0, 0, 0, 1$
w_6, w_4	$0, 0, 0, 0, 1, 0, \alpha$
w_6, w_4, w_5	$0, 0, 0, 0, \alpha, 1, \alpha^2$
w_6, w_4, w_5, w_0	$1, 0, 0, 0, \alpha^2, \alpha, \alpha^3$
w_6, w_4, w_5, w_0, w_5	$\alpha, 0, 0, 0, \alpha^3, 1 + \alpha^2, \alpha^4$
$w_6, w_4, w_5, w_0, w_5, w_4$	$\alpha^2, 0, 0, 0, 1 + \alpha^4, \alpha + \alpha^3, \alpha^5$

Table 3.2: Partial Encoding of $w_6, w_4, w_5, w_0, w_5, w_4$

3.4.2 Uniqueness

The word sequences can be unequivocally recovered from their FOFE representations [55, 56]. The uniqueness of FOFE representation is theoretically guaranteed by the following lemma and theorems:

Lemma 1. *If the forgetting factor* α *satisfies* $0 < \alpha \leq 0.5$*, there exists exactly one element in the vector s.t. its value is no less than* 1*, and that element correspond to the last symbol in the sequence.*

Proof. The symbol at position t is raised to α^{T-t}, where $t \in \{\mathbb{Z} | 1 \leq x \leq n\}$. For any symbol w in the sequence s.t. $w \neq w_T$, its value must follow

$$value(w) \leq \sum_{n=1}^{T-1} \alpha^n < 1$$

$\square$

Theorem 2. *For* $0 < \alpha \leq 0.5$*, FOFE is unique for any countable vocabulary* V *and any finite value* T*.*

Proof. Assume that there are two different sequences $S_1 = [w_1^1, w_2^1, ..., w_{T1}^1]$ and $S_2 = [w_1^2, w_2^2, ..., w_{T2}^2]$. Denote the encoding at each time step as e_{t1}^1 and e_{t2}^2 respectively, where $1 \leq t1 \leq T1$ and $1 \leq t2 \leq T2$. To prove it by contradiction, we further assume that $e_{T1}^1 = e_{T2}^2$.

Case 1 $T1 = T2$**.** By Lemma 1, $w^1_{T1} = w^2_{T2}$. By definition, $e^1_{T1} = \alpha \times e^1_{T1-1} + oneHot(w^1_{T1})$. Because both multiplication and addition are bijective, $e^1_{T1-1} = (e^1_{T1} - oneHot(w^1_{T1}))/\alpha$, and thus $e^1_{T1-1} = e^2_{T2-1}$. By induction, both sequences share the same word at the same position, which contradicts our assumption.

Case 2 $T1 > T2$**.** Applying the induction in Case 1, the last $T2$ symbols of $S1$ are identical to $S2$, and the first $T2 - T1$ symbols of $S1$ has an encoding of $e^1_{T1-T2} = e^2_{T2-T2} = \mathbf{0}$. However, the first $T2 - T1$ symbols of $S1$ is a non-empty sequence whose encoding cannot be $\mathbf{0}$, which is a contradiction.

Case 3 $T1 < T2$**.** By symmetry, it leads to the same contradiction as in Case 2.

□

Theorem 3. *For $0.5 < \alpha < 1$, given any finite value T and any countable vocabulary V, FOFE is unique almost everywhere, except only a finite set of countable choices of α.*

Proof. Ambiguity happens when at least one symbol whose value can be composed by two disjoint sets of α^{t-1}, where $t \in \{\mathbb{Z} | 1 \le t \le n\}$. The choice of α must satisfy at least one of the follow polynomial equations in order to bring up ambiguity:

$$\sum_{t=1}^{T} \xi_t \cdot \alpha^{t-1} = 0. \tag{3.2}$$

The positive terms in Eq 3.2 represent one possible way of composition while the negative terms represent the other. Each equation is of order T, which implies at most $T-1$ real roots for α. There are at most 3^T equations in the same form as Eq 3.2. Therefore, it totals no more than $(T-1)\cdot 3^T$ choices of α that lead to ambiguity. [3] $\square$

Though in theory uniqueness is not guaranteed when α is chosen from 0.5 to 1, in practice the chance of hitting such scenarios is extremely slim, almost impossible due to quantization errors in the system. Furthermore, in natural languages, normally a word does not appear repeatedly within a near context. Simply put, FOFE is capable of uniquely encoding any sequence of arbitrary length, serving as a fixed-size but theoretically lossless representation for any sequence.

[3] The actual possible number of equations is less than 3^T because some of them overlap after simplification. The actual possible number of α is much smaller. Since one symbol's value composition affect other symbol's, this proof does not give a tight upper bound.

4 FOFE in Language Modeling

4.1 FOFE Language Model

The architecture of a FOFE-based neural network language model (FOFE-FFNN-LM) is shown in Figure 4.1. It is similar to regular bigram FFNN-LMs except that it feeds a FOFE into neural network LM at each time. Moreover, the FOFE can be easily scaled to higher orders like n-gram NNLMs. For example, Figure 4.2 is an illustration of a second order FOFE-FFNN-LM.

FOFE is a simple recursive encoding method but a direct sequential implementation may not be efficient for the parallel computation platform like GPUs. In this section, we will show that the FOFE computation can be efficiently implemented as sentence-by-sentence matrix multiplications, which are suitable for the mini-batch based SGD method running on GPUs.

Given a sentence, $S = \{w_1, w_2, \cdots, w_T\}$, where each word is represented by a 1-of-K code as $\mathbf{e}_t$ $(1 \leq t \leq T)$. The FOFE codes for all partial sequences in S can

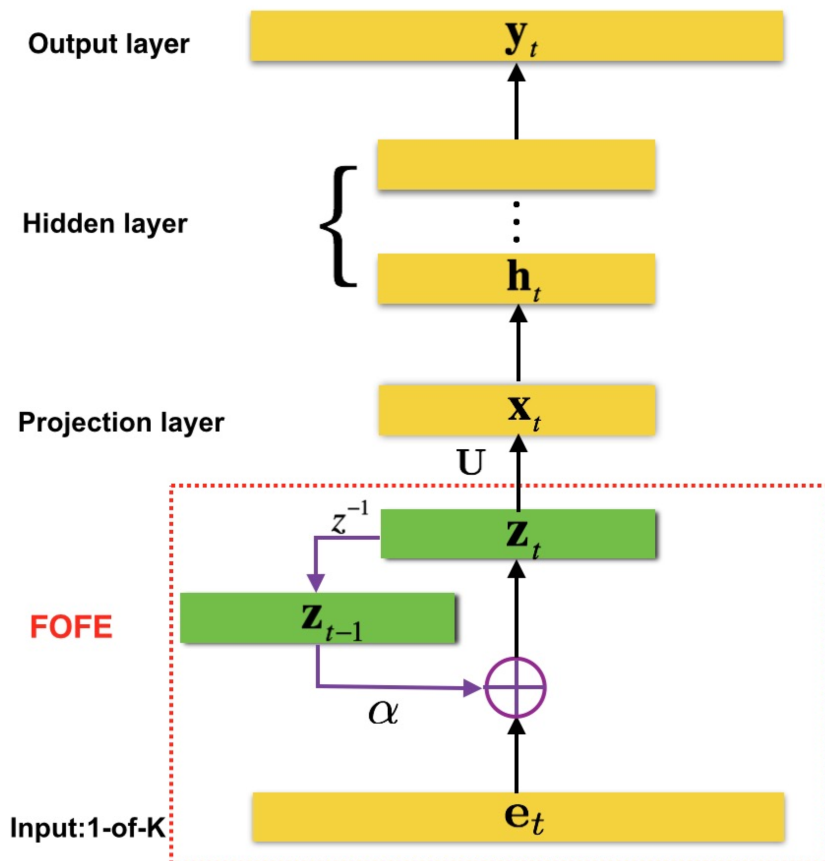

Figure 4.1: Diagram of 1st-order FOFE-FFNN-LM.

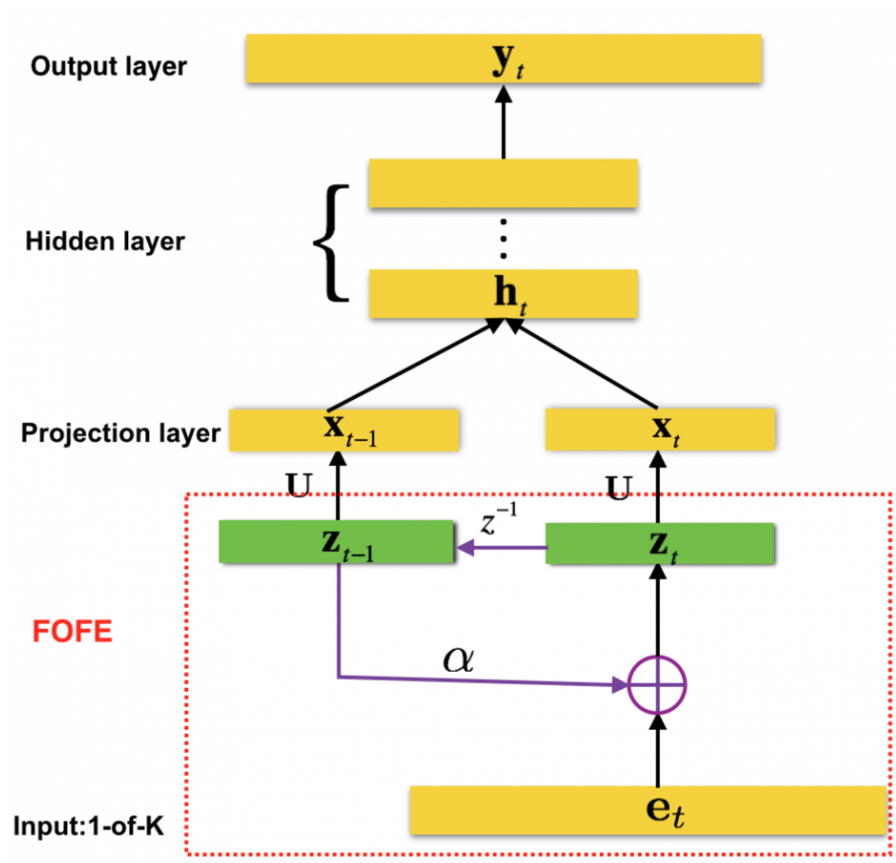

Figure 4.2: Diagram of 2nd-order FOFE-FFNN-LM.

be computed based on the following matrix multiplication:

$$\mathbf{S} = \begin{bmatrix} 1 & & & & \\ \alpha & 1 & & & \\ \alpha^2 & \alpha & 1 & & \\ \vdots & & \ddots & 1 & \\ \alpha^{T-1} & \cdots & & \alpha & 1 \end{bmatrix} \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \\ \vdots \\ \mathbf{e}_T \end{bmatrix} = \mathbf{MV} \tag{4.1}$$

where $\mathbf{V}$ is a matrix arranging all 1-of-K codes of the words in the sentence row by row, and $\mathbf{M}$ is a T-th order lower triangular matrix. Each row vector of $\mathbf{S}$ represents a FOFE code of the partial sequence up to each position in the sentence.

This matrix formulation can be easily extended to a mini-batch consisting of several sentences. Assume that a mini-batch is composed of N sequences, $\mathcal{L} = \{S_1\ S_2 \cdots S_N\}$, we can compute the FOFE codes for all sentences in the mini-batch as follows:

$$\bar{\mathbf{S}} = \begin{bmatrix} \mathbf{M}_1 & & & \\ & \mathbf{M}_2 & & \\ & & \ddots & \\ & & & \mathbf{M}_N \end{bmatrix} \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_N \end{bmatrix} = \bar{\mathbf{M}}\bar{\mathbf{V}}. \tag{4.2}$$

When feeding the FOFE codes to FFNN as shown in Figure 4.1, we can compute all the histories in S projected by word embedding as follow:

$$(\mathbf{MV})\mathbf{U} = \mathbf{M}(\mathbf{VU}) \tag{4.3}$$

where **U** denotes the word embedding matrix that projects the word indices onto a continuous low-dimensional continuous space. As above, **VU** can be done efficiently by looking up the embedding matrix. Therefore, for the computational efficiency purpose, we may apply FOFE to the word embedding vectors instead of the original high-dimensional one-hot vectors. In the backward pass, we can calculate the gradients with the standard BP algorithm rather than BPTT. As a result, FOFE-FFNN-LMs are the same as the standard FFNN-LMs in terms of computational complexity in training, which is much more efficient than RNN-LMs.

4.2 Experiment Results

In order to evaluate the performance of FOFE-FFNN-LM, we have selected 3 popular data sets, namely Penn TreeBank, Large Text Compression Benchmark and Google Billion Word. Each line in these corpora is a single sentence, so history does not cross sentence boundary.

We will compared the proposed model with traditional back-off n-gram LMs and the state-of-ther-art performance obtained by NNLMs. We apply the following setting by default:

- End-of-sentence symbol $</s>$ is appended to the end of each sentence, and it is included during training and evaluation. Begin-of-sentence symbol $<s>$ is padded to the input when history is shorter than the order of the FFNN-LM.

- When NNLM is involved, model parameters are randomly initialized by a uniform distribution between $-\sqrt{\frac{6}{fanIn+fanOut}}$ and $\sqrt{\frac{6}{fanIn+fanOut}}$, where $fanIn$ and $fanOut$ are the input and the output dimensions respectively [15].

- The learning rate lr is frozen until the first non-improving epoch e. lr is halved from the e^{th} epoch on. We use the best learning rate chosen from $\{0.512, 0.256, 0.128, 0.064, 0.032\}$ by validation set.

4.2.1 Penn TreeBank

The Penn TreeBank (PTB) protion of the WSJ corpus has been used extensively in LM community. In our experiment, we follow the same training/validation/test split as in other researches [40]. PTB is a small corpus. There are 930k, 74k, 82k words in training, validation and test respectively. The vocabulary is restricted to the 10k most frequent words. The rest are mapped to a special token <*unk*>.

PTB is relatively small. Overfitting is very likely if the model is not carefully regularized. We train using momentum [45] of 0.9 and weight decay (L2 penalty) of 0.0004. We found that the additional time and space consumption from momentum and weight decay are negligible because of PTB's size. It also turns out that the same level of performance is not reachable with plain SGD.

In table 4.3, we study how the forgetting factor α affects performance. Through out this set of experiments, hyper-parameters except forgetting factor and order are

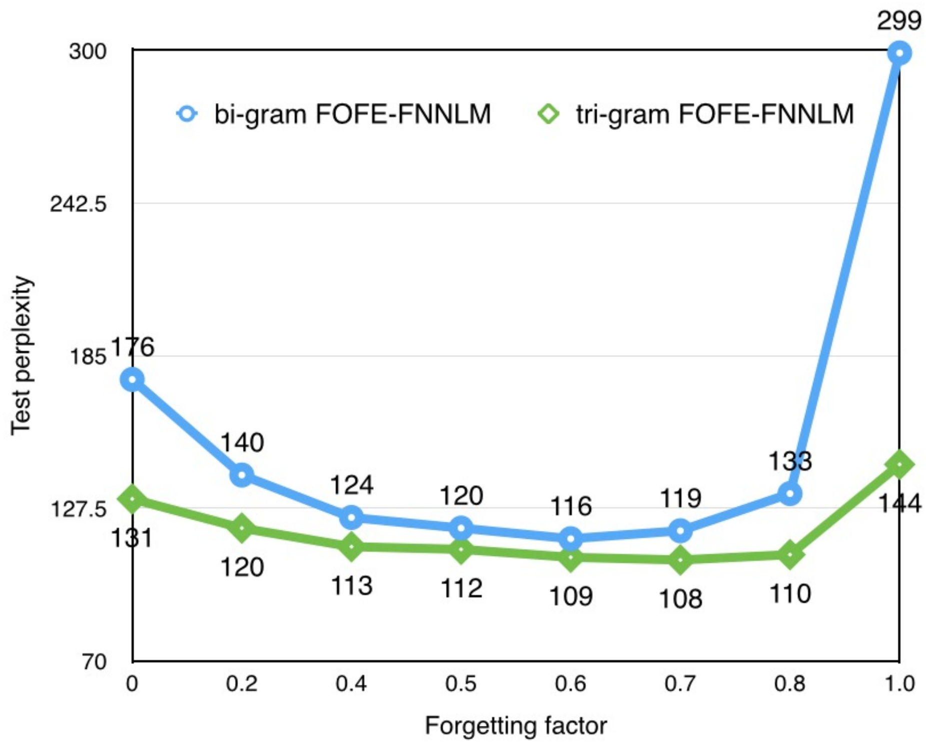

Figure 4.3: PPL of FOFE-FNNLM as a function of the forgetting factor.

Model	PPL
KN 5-gram [40]	141
FNNLM [42]	140
RNNLM [40]	123
LSTM [18]	117
bigram FFNN-LM	176
trigram FFNN-LM	131
4-gram FFNN-LM	118
5-gram FFNN-LM	114
6-gram FFNN-LM	113
1st-order FOFE-FNNLM	116
2nd-order FOFE-FNNLM	**108**

Table 4.1: PPL on PTB for various LMs.

frozen, i.e. all experiments are with 100-dimension word embedding and 2 hidden layers of 400 nodes. We have examine $\alpha \in \{0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1\}$. When $\alpha = 0$, it is equivalent to bigram-FFNN-LM. We observed that when α lies between 0.5 and 0.7, FOFE-FFNN-LM yields best performance. Therefore, we chose $\alpha = 0.7$ for experiments afterwards.

We have built several baseline n-gram FFNN-LMs. Meanwhile, we have compared our results with known solutions in the literature. In Table 4.1, we have summarized the PPL on the PTB test set for various models. The proposed FOFE-FNNLMs significantly outperform the baseline FFNN-LMs with the same architecture. Moreover, the FOFE-FNNLMs even overtake a well-trained RNNLM (400 hidden units) in [40] and an LSTM in [18], which indicates that FOFE-FFNN-LMs can effectively model the long-term dependency in language without using any recurrent feedback. At last, the 2nd-order FOFE-FFNN-LM improves further, yielding the PPL of 108 on PTB. It also outperforms all higher-order FFNN-LM counterparts (4-gram, 5-gram and 6-gram), which are bigger in terms of parameter number. To our best knowledge, this is one of the best reported results on PTB without dropout [50] and model combination.

Model	Architecture	Test PPL
KN 3-gram	-	156
KN 5-gram	-	132
FFNN-LM	[1*200]-400-400-80k	241
	[2*200]-400-400-80k	155
	[2*200]-600-600-80k	150
	[3*200]-400-400-80k	131
	[4*200]-400-400-80k	125
RNN-LM	[1*600]-600-80k	112
FOFE FFNN-LM	[1*200]-400-400-80k	120
	[1*200]-600-600-80k	115
	[2*200]-400-400-80k	112
	[2*200]-600-600-80k	**107**

Table 4.2: PPL on LTCB for various LMs. [M*N] denotes the sizes of the input context window and projection layer.

4.2.2 Large Text Compression Benchmark

We have further examined the modeling power of FOFE-FFNN-LMs on a larger text corpus, i.e. Large Text Compression Benchmark (LTCB) [36], which contains the first 10^9 bytes of the English version of Wikipedia dumped in March 3, 2006. The corpus is converted to lowercase. The most frequent 80k are kept and the rest are similarly mapped to $<unk>$. We have trained several baseline systems:

- two n-gram LMs (3-gram and 5-gram) using the modified Kneser-Ney smoothing without count cutoffs,
- several traditional FFNN-LMs with different model sizes and input context windows (bigram, trigram, 4-gram and 5-gram), and
- an RNN-LM with one hidden layer of 600 nodes using the toolkit in [39], in which we have further used a spliced sentence batch in [7] to speed up the training on GPUs.

Moreover, we have examined four FOFE-FFNN-LMs with various model sizes and input window sizes (two 1st-order FOFE models and two 2nd-order ones). For all NNLMs, we have used an output layer of the full vocabulary (80k words). In these experiments, we have used an initial learning rate of 0.1, and a bigger mini-batch of 500 for FFNN-LMMs and FOFE-FFNN-LMs, and of 256 sentences for the RNN-LMs. Since forgetting factor of 0.7 demonstrates best performance in

PTB, it is used throughout the experiments in LCTB. Overfitting seldom happens when the dataset is sufficiently large. We pick SGD as our optimizer for all NNLMs without regularization. Experimental results in Table 4.2 have shown that the FOFE-FFNN-LMs significantly outperform the baseline FFNN-LMs (including some larger higher-order models) and also slightly overtake the popular RNN-LMs, yielding the best result (perplexity of 107) on the test set.

4.2.3 Google 1 Billion Benchmark

The Google Billion Word dataset [6] is one of the largest benchmark for LM with almost one billion words. Its vocabulary size is over 800k. In order to fairly compare FOFE with known solutions, we follow the same preprocessing in [6], replacing words that occur less than 3 times with $<unk>$, and removing duplicated sentences.

Because computing *softmax* over such big vocabulary in a large scale is almost impossible, we estimate it by NCE. We have trained a medium-size and a large-size FOFE-FFNN-LMs (shown in Table 4.3). The former and the latter sample 2048 and 4096 noise examples respectively. We have tried several options of the normalization constant Z. e (base of the natural logarithm) shows the fastest convergence in the first 100 mini-batches, so we stick to this value throughout all experiments.

General speaking, the rule of thumb is that the larger the model, the better performance. Because of a huge vocabulary size, word embedding and NCE account for 95% of the model size. However, only a tiny portion has non-zero gradients within a mini-batch. Optimizers such as momentum and Adam [12] significantly increase time complexity and space complexity in the sense that their auxiliary data structure doubles or triples memory usage and renders the sparse parameter update of word embedding and NCE impossible. It in turn prevents larger model. Therefore, we pick SGD instead of other fancy optimizers. We employ a word embedding of 256 dimensions, 3 hidden layers of $\{2048, 4096\}$ neurons, and a compression layer of $\{640, 720\}$ neurons, totaling $\{0.73B, 0.78B\}$ parameters, which is the biggest model that fits in a GeForce GTX TITAN X (Maxwell) of 12 GB memory.

The results are presented in Table 4.3 along with the performance reported in the community. FOFE-FFNN-LM achieves similar performance with greater efficiency and less computational resources. FOFE-FFNN-LM is competitive to [28] with fewer number of parameters. However, [28] imposes demanding constraint on hardware. [12] and [28] sidestep the parameter number disaster introduced by word embedding and Softmax output by working at character level rather than word level. However, character CNN is known to be computationally expensive.

Model	Test PPL	#param	Hardware	Time
Sigmoid-RNN-2048 [27]	68.3[4]	4.1B	1 CPU	175 hours
Interpolated KN 5-gram & 1.1B n-grams [6]	67.6	1.8B	100 CPUs	3 hours
Sparse Non-Negative Matrix LM [49]	52.9	33B	-	-
RNN-1024 + MaxEnt 9-gram [6]	51.3	20B	24 GPUs	10 days
LSTM-1024-512 [28]	48.2	0.82B	40 GPUs	10 hours
LSTM-2048-512 [28]	43.7	0.83B	40 GPUs	10 hours
LSTM + CNN input [28]	30.0	1.04B	40 GPUs	3 weeks
GCNN-13 [12]	38.1		1 GPU	2 weeks
FOFE-FFNN-LM 3x256-2048-2048-2048-640	**45.4**	0.73B	**1 GPU**	6 days
FOFE-FFNN-LM 3x256-4096-4096-4096-720	**43.5**	0.73B	**1 GPU**	12 days

Table 4.3: PPL on GBW on various LMs. Some cells are blank because they are not reported.

5 FOFE in Entity Discovery

5.1 Local Detection

Our FOFE-based local detection approach for NER, called **FOFE-NER** hereafter, is motivated by the way how human actually infers whether a word segment in text is an entity or mention, where the entity types of the other entities in the same sentence is not a must. Particularly, the dependency between adjacent entities is fairly weak in NER. Whether a fragment is an entity or not, and what class it may belong to, largely depend on the internal structure of the fragment itself as well as the left and right contexts in which it appears. To a large extent, the meaning and spelling of the underlying fragment are informative to distinguish named entities from the rest of the text. Contexts play a very important role in NER or MD when it involves multi-sense words/phrases or out-of-vocabulary (OOV) words.

As shown in Figure 5.1, our proposed **FOFE-NER** method will examine all possible fragments in text (up to a certain length) one by one. For each fragment, it uses the FOFE method to fully encode the underlying fragment itself, its left

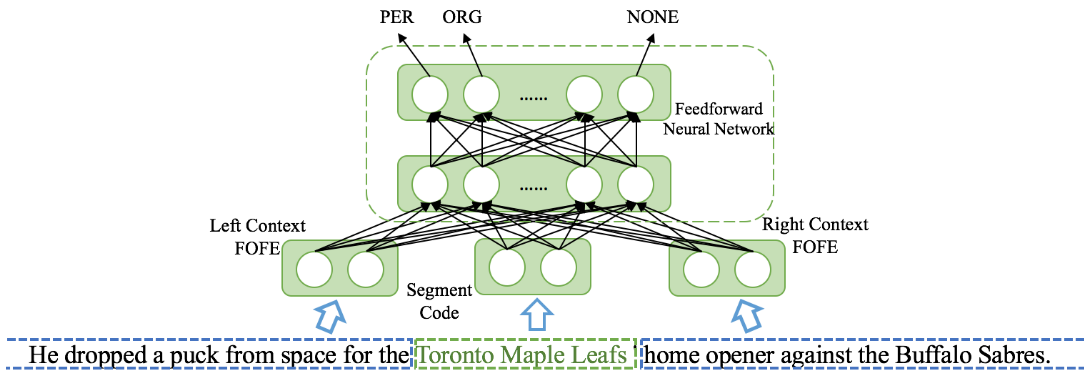

Figure 5.1: FOFE codes are fed into a FFNN. The window currently examines the fragment of *Toronto Maple Leafs*. The window will scan and scrutinize all fragments up to K words.

context and right context into some fixed-size representations, which are in turn fed to an FFNN to predict whether the current fragment is NOT a valid entity mention (*NONE*), or its correct entity type (*PER*, *LOC*, *ORG* and so on) if it is a valid mention. This method is appealing because the FOFE codes serves as a theoretically lossless representation of the hypothesis and its full contexts. FFNN is used as a universal approximator to map from text to the entity labels.

5.2 Feature Extraction

In this work, we use FOFE to explore both word-level and character-level features for each fragment and its contexts.

5.2.1 Word-level Features

FOFE-NER generates several word-level features for each fragment hypothesis and its left and right contexts as follows:

- Bag-of-word (BoW) of the fragment, e.g. bag-of-word vector of 'Toronto', 'Maple' and 'Leafs' in Figure 5.1.

- FOFE code for left context including the fragment, e.g. FOFE code of the word sequence of "*... puck from space for the Toronto Maple Leafs*" in Figure 5.1.

- FOFE code for left context excluding the fragment, e.g. the FOFE code of the word sequence of "*... puck from space for the*" in Figure 5.1..

- FOFE code for right context including the fragment, e.g. the FOFE code of the word sequence of "*... against opener home ' Leafs Maple Toronto*" in Figure 5.1.

- FOFE code for right context excluding the fragment, e.g. the FOFE code of the word sequence of "*... against opener home* " in Figure 5.1.

Moreover, all of the above word features are computed for both case-sensitive words in raw text as well as case-insensitive words in normalized lower-case text. These FOFE codes are projected to lower-dimension dense vectors based on two word embedding matrices, $\mathbf{W}_s$ and $\mathbf{W}_i$, for case-sensitive and case-insensitive FOFE codes respectively. These two projection matrices are initialized by word embeddings trained by *word2vec*, and fine-tuned during the learning of the neural networks.

Due to the recursive computation of FOFE codes in eq.(3.1), all of the above FOFE codes can be jointly computed for one sentence or document in a very efficient manner.

5.2.2 Character-level Features

On top of the above word-level features, we also augment character-level features for the underlying segment hypothesis to further model its morphological structure. The aforementioned FOFE method can be easily extended to model character-level feature in NLP. Any word, phrase or fragment can be viewed as a sequence of characters. In this way, based on a pre-defined set of all possible characters, we may apply the same FOFE method to encode the sequence of characters. This always leads to a fixed-size representation, irrelevant to the number of characters in question. For the example in Figure 5.1, the current fragment, *Toronto Maple Leafs*, is considered as a sequence of case-sensitive characters, i.e. "{'T', 'o', ..., 'f' , 's' }", we then add the following character-level features for this fragment:

- Left-to-right FOFE code of the character sequence of the underlying fragment.

That is the FOFE code of the sequence, [*'T', 'o', ..., 'f' , 's'*].

- Right-to-left FOFE code of the character sequence of the underlying fragment. That is the FOFE code of the sequence, [*'s' , 'f' , ..., 'o', 'T'*].

These case-sensitive character FOFE codes are also projected by another character embedding matrix, which is randomly initialized and fine-tuned during model training.

5.3 Training and Decoding Algorithm

Obviously, the above **FOFE-NER** model will take each sentence of words, $S = [w_1, w_2, w_3, ..., w_m]$, as input, and examine all continuous sub-sequences $[w_i, w_{i+1}, w_{i+2}, ..., w_j]$ up to n words in S for possible entity types. All sub-sequences longer than n words are considered as non-entities in this work.

When we train the model, based on the entity labels of all sentences in the training set, we will generate many sentence fragments up to n words. These fragments fall into three categories:

- Exact-match with an entity label, e.g., the fragment "*Toronto Maple Leafs*" in the previous example.
- Partial-overlap with an entity label, e.g., "*for the Toronto*".

- Disjoint with all entity label, e.g. "*from space for*".

For all exact-matched fragments, we generate the corresponding outputs based on the types of the matched entities in the training set. For both partial-overlap and disjoint fragments, we introduce a new output label, **NONE**, to indicate that these fragments are not a valid entity. Therefore, the output nodes in the neural networks contain all entity types plus a rejection option denoted as **NONE**.

During training, we implement a producer-consumer software design such that a thread fetches training examples, computes all FOFE codes and packs them as a mini-batch while the other thread feeds the mini-batches to neural networks and adjusts the model parameters and all projection matrices. Since "partial-overlap" and "disjoint" significantly outnumber "exact-match", they are down-sampled so as to balance the data set. The training procedure is illustrated in Algorithm 1.

During inference, all fragments not longer than n words are all fed to **FOFE-NER** to compute their scores over all entity types. In practice, these fragments can be packed as one mini-batch so that we can compute them in parallel on GPUs. As the NER result, the **FOFE-NER** model will return a subset of fragments only if: i) they are recognized as a valid entity type (not **NONE**); AND ii) their NN scores exceed a global pruning threshold.

Occasionally, some partially-overlapped or nested fragments may occur in the above pruned prediction results. We can use one of the following simple post-

Algorithm 1 FOFE-NER TRAINING algorithm

procedure TRAIN(S, o, d, n)

▷ S: labeled sentence set

▷ n: maximum ner length

▷ o: overlap sample rate

▷ d: disjoint sample rate

$data \leftarrow \{\}$

for $[w_1, w_2, ..., w_l]$ in S **do**

for $i \leftarrow 1...l$ **do**

for $j \leftarrow i, min(l, i+n-1)$ **do**

$f \leftarrow feature([w_i, ..., w_j])$

$t \leftarrow label([w_i, ..., w_j])$

if $overlap([w_i, ..., w_j])$ and $rand(0,1) > o$ **then**

continue

end if

if $disjoint([w_i, ..., w_j])$ and $rand(0,1) > d$ **then**

continue

end if

$data \leftarrow data \cup \{(f, t)\}$

end for

end for

end for

for *minibatch* in *data* **do**

$DNN_{fofe}.train(minibatch)$

end for

end procedure

processing methods to remove overlappings from the final results:

1. *highest-first*: We check every word in a sentence. If it is contained by more than one fragment in the pruned results, we only keep the one with the maximum NN score and discard the rest. It is listed in Algorithm 4.

2. *longest-first*: We check every word in a sentence. If it is contained by more than one fragment in the pruned results, we only keep the longest fragment and discard the rest. It is listed in Algorithm 5.

Either of these strategies leads to a collection of non-nested, non-overlapping, non-NONE entity labels.

In some tasks, it may require to label all nested entities. This has imposed a big challenge to the sequence labeling methods. However, the above post-processing can be slightly modified to generate nested entities' labels. In this case, we first run either *highest-first* or *longest-first* to generate the first round result. For every entity survived in this round, we will recursively run either *highest-first* or *longest-first* on all entities in the original set, which are completely contained by it. The decoding precedure is detailed in Algorithm 2 and Algorithm 3. This will generate more prediction results. This process may continue to allow any levels of nesting. For example, for a sentence of "w_1 w_2 w_3 w_4 w_5", if the model first generates the prediction results after the global pruning, as ["w_2w_3", PER, 0.7], ["w_3w_4", LOC, 0.8], ["$w_1w_2w_3w_4$", ORG, 0.9], if we choose to run *highest-first*, it will generate the first entity label as ["$w_1w_2w_3w_4$", ORG, 0.9]. Secondly, we will run *highest-first* on the two fragments that are completely contained by the first one, i.e., ["w_2w_3", PER, 0.7], ["w_3w_4", LOC, 0.8], then we will generate the second nested entity label as ["w_3w_4", LOC, 0.8]. Fortunately, in any real NER and MD tasks, it is pretty rare to have overlapped predictions in the NN outputs. Therefore, the extra expense to run this recursive post-processing method is minimal.

5.4 Second-Pass Augmentation

As we know, CRF brings marginal performance gain to all taggers (but not limited to NER) because of the dependencies (though fairly weak) between entity types. We may easily add this level of information to our model by introducing another

Algorithm 2 ELIMINATE OVERLAP algorithm

procedure DEL_RECUR($scores$, $[algo_1, algo_2, ..., algo_x]$, $[t_1, t_2, ..., t_x]$)

$\triangleright$ x: desired level of nested mention

$\triangleright$ $[t_1, t_2, ..., tx]$: threshold at each nested level

$\triangleright$ $scores$: $(i, j, label_{i,j}, score_{i,j})$ tuples

$\triangleright$ n: maximum ner length

  assert $algo_1 \in \{\ highest\text{–}first, longest\text{–}first\}$

  $candidates \leftarrow \{(i, j, label_{i,j}, score_{i,j})|$

    $(i, j, label_{i,j}, score_{i,j}) \in scores$ and $score_{i,j} \geq t_1\}$

  $result \leftarrow algo_1(candidates)$

  $nested \leftarrow \{\}$

  for $(i, j, label_{i,j}, score_{i,j}) \in result$ **do**

    $sub \leftarrow \{(x, y, label_{x,y}, score_{x,y})|$

      $(x, y, label_{x,y}, score_{x,y}) \in scores$ and $(i \leq x \leq y < j$ or $i < x \leq y \leq j)\}$

    $nested \leftarrow nested \cup$ DEL_RECUR$(sub, [algo_2, algo_3, ..., algo_x], [t_2, t_3, ..., t_x])$

  end for

  return $result \cup nested$

end procedure

Algorithm 3 FOFE-NER DECODING algorithm

procedure DECODE($[w_1, w2, ..., w_l]$,

$[algo_1, algo_2, ..., algo_x]$, $[t_1, t_2, ..., t_x]$, n)

$\triangleright$ $[w_1, w2, ..., w_l]$: a sentence

$\triangleright$ n: maximum ner length

$\triangleright$ x: level of desired nested mention to detect

$\triangleright$ $[t_1, t_2, ..., t_x]$: threshold at each nested level

$\triangleright$ $[algo_1, algo_2, ..., algo_x]$: algorithms used to eliminate overlapping of each nested level

    $r \leftarrow \{\}$ $\triangleright$ r: temporary results

    for $i \leftarrow 1...l$ **do**

        for $j \leftarrow i...min(l, i + n - 1)$ **do**

            $f \leftarrow feature([w_i, ..., w_j])$

            $label_{i,j}, score_{i,j} \leftarrow DNN_{fofe}.eval(f)$

            if $label_{i,j} \neq NONE$ **then**

                $r \leftarrow r \cup \{(i, j, label_{i,j}, score_{i,j})\}$

            end if

        end for

    end for

end procedure

Algorithm 4 HIGHEST-FIRST DECODING algorithm

function HIGHEST1ST($scores$, l)

  $\triangleright$ $scores$: $(i, j, label_{i,j}, score_{i,j})$ tuples

  $\triangleright$ l: sentence length

  for $k \leftarrow 1...l$ **do**

    $removed \leftarrow \{\}$

    $candidates \leftarrow \{(i, j, label_{i,j}, score_{i,j})|$

      $(i, j, label_{i,j}, score_{i,j}) \in scores$ and $i \leq k \leq j\}$

    $candidates \leftarrow sortByScore(candidates)$

    if $|candidates| > 0$ **then**

      $candidates \leftarrow removeHighestScore(candidates)$

    end if

    $removed \leftarrow removed \cup candidates$

  end for

  $estimate \leftarrow sortByStartIdx(scores - removed)$

  $estimate \leftarrow mergeAdjacientOfSameType(estimate)$

  return $estimate$

end function

Algorithm 5 LONGEST-FIRST DECODING algorithm

function LONGEST1ST($scores$, l)

▷ $scores$: $(i, j, label_{i,j}, score_{i,j})$ tuples

▷ l: sentence length

  $best \leftarrow createEmptyList()$

  $candidate \leftarrow createHashedMap()$

  for $(i, j, label_{i,j}, score_{i,j}) \in scores$ **do** $candidate[(i, j)] \leftarrow label_{i,j}$

  end for

  for $i \leftarrow 1...l$ **do**

    $cur \leftarrow createEmptyList()$

    if $(1, i) \in candidate$ **then**

      $cur.insert((1, i))$

      $best.insert((1, cur))$

    else

      $best.insert((0, cur))$

    end if

    for $j \leftarrow 1...i$ **do**

      $newLen \leftarrow best[j][0] + i - j + 1$

      if $(j, i) \in candidate$ and $newLen > head(lastOf(best))$ **then**

        $best.pop()$

            $best.insert((newLen, (j+1, i+1))$

        end if

    end for

    $last, secondLast \leftarrow lastOf(best), secondLastOf(best)$

    if $i > 1$ and $head(last) > head(secondLast)$ **then**

        $best.pop()$

        $best.insert(secondLast)$

    end if

end for

$estimate \leftarrow mergeAdjacientOfSameType(estimate)$

return $estimate$

end function

pass of **FOFE-NER**. We call it *2nd-pass FOFE-NER*.

In 2nd-pass FOFE-NER, another set of model is trained on outputs from the first-pass **FOFE-NER**, including all predicted entities. For example, given a sentence

$$S = [w_1, w_2, ...w_i, ...w_j, ...w_n]$$

and an underlying word segment $[w_i, ..., w_j]$ in the second pass, every predicted entity outside this segment is substituted by its entity type predicted from the first pass. For example, in the first pass, a sentence like *"Google has also recruited Fei-Fei Li, director of the AI lab at Stanford University."* is predicted as: *"<ORG> has also recruited Fei-Fei Li, director of the AI lab at <ORG>."* In 2nd-pass FOFE-NER, when examining the segment "*Fei-Fei Li*", the predicted entity types *<ORG>* are used to replace the actual named entities. The 2nd-pass FOFE-NER model is trained on the outputs of the first pass, where all detected entities are replaced by their predicted types as above. The training process is depicted in Figure 5.2.

During inference, the results returned by the 1st-pass model are substituted in the same way. The scores for each hypothesis from 1st-pass model and 2nd-pass model are linear interpolated and then decoded by either *highest-first* or *longest-first* to generate the final results of 2nd-pass FOFE-NER.

Obviously, 2nd-pass FOFE-NER may capture the semantic roles of other en-

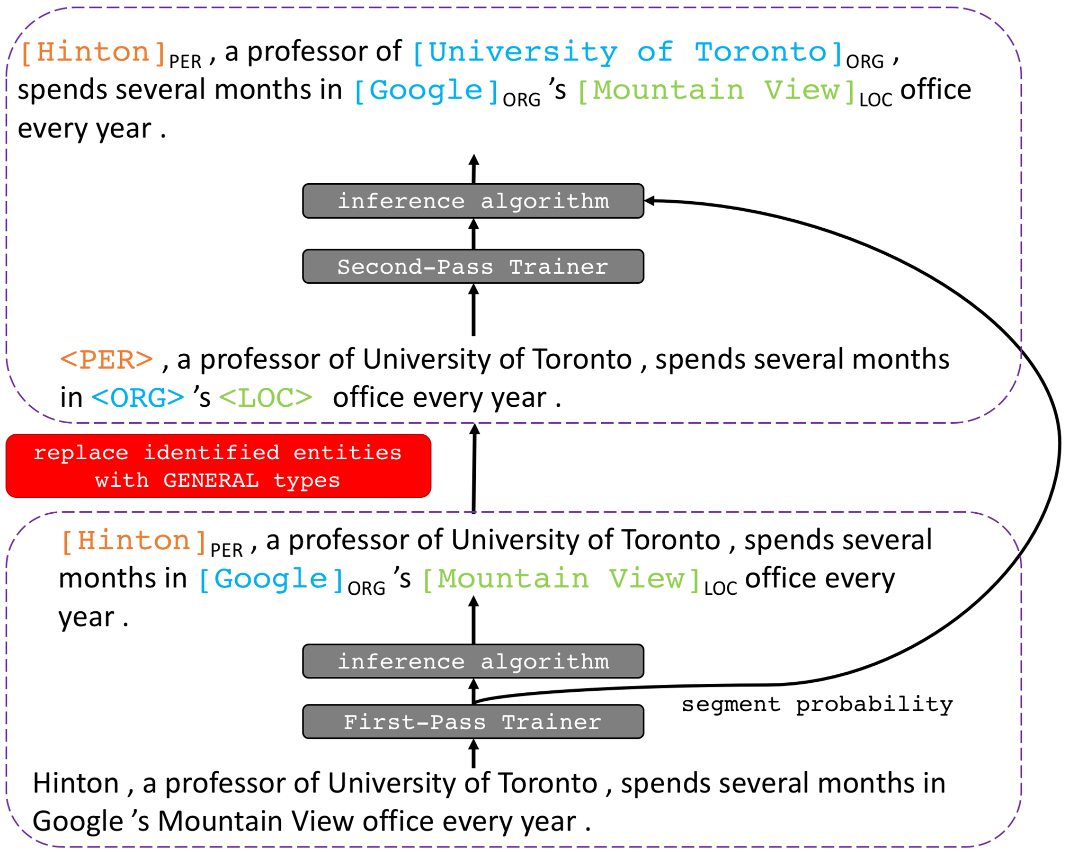

Figure 5.2: Illustration of the 2nd-pass Augmentation

tities while filtering out unwanted constructs and sparse combinations. On the other hand, it enables longer context expansion, since FOFE memorizes contextual information in an unselective decaying fashion.

5.5 Experiment Results

In this section, we evaluate the effectiveness of our proposed methods on several popular NER and MD tasks, including CoNLL 2003 NER task and TAC-KBP2015 and TAC-KBP2016 Tri-lingual Entity Discovery and Linking (EDL) tasks. [5]

5.5.1 CoNLL 2003 NER task

The CoNLL2003 dataset [47] consists of newswire from the Reuters RCV1 corpus tagged with four types of non-nested named entities: location (LOC), organization (ORG), person (PER), and miscellaneous (MISC). Table 5.2 shows the data distribution of CoNLL2003 dataset.

The top 100,000 words, are kept as vocabulary, including punctuations. For the case-sensitive embedding, an OOV is mapped to <unk>; if it contains no upper-case letter and <UNK>; otherwise. We perform grid search on several hyper-parameters using a held-out dev set. Here we summarize the set of hyper-parameters used in our experiments:

- *Learning rate*: initially set to 0.128 and is multiplied by a decay factor each epoch so that it reaches 1/16 of the initial value at the end of the training;
- *Network structure*: 3 fully-connected layers of 512 nodes with ReLU activation, randomly initialized based on a uniform distribution between $-\sqrt{\frac{6}{N_i+N_o}}$ and $\sqrt{\frac{6}{N_i+N_o}}$ [15];
- *Character embeddings*: 64 dimensions, randomly initialized between -1 and 1.

[5] We have made our codes available at `https://github.com/xmb-cipher/fofe-ner` for readers to reproduce the results in this paper.

FEATURE			P	R	F1
word-level	case-insensitive	context FOFE incl. word fragment	86.64	77.04	81.56
		context FOFE excl. word fragment	53.98	42.17	47.35
		BoW of word fragment	82.92	71.85	76.99
	case-sensitive	context FOFE incl. word fragment	88.88	79.83	84.12
		context FOFE excl. word fragment	50.91	42.46	46.30
		BoW of word fragment	85.41	74.95	79.84
char-level	Char FOFE of word fragment		67.67	52.78	59.31
	Char CNN of word fragment		78.93	69.49	73.91
all case-insensitive features			90.11	82.75	86.28
all case-sensitive features			90.26	86.63	88.41
all word-level features			92.03	86.08	88.96
all word-level & Char FOFE features			91.68	88.54	**90.08**
all word-level & Char CNN features			91.80	88.58	**90.16**
all word-level & all char-level features			93.29	88.27	**90.71**

Table 5.1: Effect of various FOFE feature combinations on the CoNLL2003

	Articles	Sentences	Tokens	LOC	MISC	ORG	PER
train	946	14,987	203,621	7,140	3,438	6,321	6,600
dev	216	3,466	51,362	1,837	922	1,341	1,842
test	231	3,684	46,435	1,668	702	1,661	1,617

Table 5.2: Data distribution of CoNLL2003

- *mini-batch*: 512;
- *Dropout Rate*: initially set to 0.4, slowly decreased during training until it reaches 0.1 at the end.
- *Number of Epochs*: 128;
- *Word Embedding*: case-sensitive and case-insensitive word embeddings of 256 dimensions, trained from Reuters RCV1;
- We stick to the official data train-dev-test partition.
- *Forgetting Factor*: $\alpha = 0.5$. [6]
- *Character CNN*: The same method in Section 2.2.2 is applied. 8 sets of kernels are used, whose heights range from 2 to 9. Each set of kernels of depth 16.

[6]The choice of the forgetting factor α is empirical. We've evaluated $\alpha = 0.5, 0.6, 0.7, 0.8$ on a development set in some early experiments. It turns out that $\alpha = 0.5$ is the best. As a result, $\alpha = 0.5$ is used for all NER/MD tasks throughout this paper.

The character embedding is not shared with FOFE character embedding but initialized in the same way.

We have investigated the performance of our method on the CoNLL-2003 dataset by using different combinations of the FOFE features (both word-level and character-level). The detailed comparison results are shown in Table 5.1. In Table 5.3, we have compared our best performance with some top-performing neural network systems on this task. As we can see from Table 5.3, our system (highest-first decoding) yields very strong performance (90.85 in F_1 score) in this task, outperforming most of neural network models reported on this dataset. More importantly, we have not used any hand-crafted features in our systems, and all features (either word or char level) are automatically derived from the data. Highest-first and longest-first perform similarly. In [8][7], a slightly better performance (91.62 in F_1 score) is reported but a customized gazetteer is used in theirs.

5.5.2 KBP2015 EDL Task

Given a document collection in three languages (English, Chinese and Spanish), the KBP2015 tri-lingual EDL task [26] requires to automatically identify entities **(including nested entities)** from a source collection of textual documents in

[7]In their work, they have used a combination of training-set and dev-set to train the model, differing from all other systems (including ours) in Table 5.3.

algorithm	word	char	gaz	cap	pos	F1
CNN-BLSTM-CRF [9]	✓	✗	✓	✓	✗	89.59
BLSTM-CRF [24]	✓	✓	✓	✓	✓	90.10
BLSTM-CRF [46]	✓	✗	✓	✓	✓	89.28
BLSTM-CRF, char-CNN [8]	✓	✓	✓	✗	✗	**91.62**
Stack-LSTM-CRF, char-LSTM [32]	✓	✓	✗	✗	✗	**90.94**
FOFE-NER	✓	✓	✗	✗	✗	**90.71**
FOFE-NER + dev-set						**90.92**
FOFE-NER + 2nd-pass						**90.85**
FOFE-NER + dev-set + 2nd-pass						**91.17**

Table 5.3: Performance (F_1 score) comparison among various neural models reported on the CoNLL2003 dataset, and the different features used in these methods.

	English	Chinese	Spanish	ALL
Train	168	147	129	444
Eval	167	167	166	500

Table 5.4: Number of Documents in KBP2015

multiple languages as in Table 5.4, and classify them into one of the following pre-defined five types: Person (PER), Geo-political Entity (GPE), Organization (ORG), Location (LOC) and Facility (FAC). The corpus consists of news articles and discussion forum posts published in recent years, related but non-parallel across languages.

Three models are trained and evaluated independently. Unless explicitly listed, hyperparameters follow those used for CoNLL2003 as described in section 5.5.1 and 2nd-pass model is not used. Three sets of word embeddings of 128 dimensions are derived from English Gigaword [44], Chinese Gigaword [16] and Spanish Gigaword [38] respectively. Some language-specific modifications are made:

- **Chinese**: Because Chinese segmentation is not reliable, we label Chinese at character level. The analogous roles of case-sensitive word-embedding and case-sensitive word-embedding are played by character embedding and word-embedding in which the character appears. Neither Char FOFE features nor Char CNN features are used for Chinese.

	2015 track best			ours		
	P	R	F_1	P	R	F_1
Trilingual	75.9	69.3	72.4	78.3	69.9	**73.9**
English	79.2	66.7	**72.4**	77.1	67.8	72.2
Chinese	79.2	74.8	**76.9**	79.3	71.7	75.3
Spanish	78.4	72.2	75.2	79.9	71.8	**75.6**

Table 5.5: Entity Discovery Performance of our method on the KBP2015 EDL evaluation data, with comparison to the best systems in KBP2015 official evaluation.

- **Spanish**: Character set of Spanish is a super set of that of English. When building character-level features, we use the mod function to hash each character's UTF8 encoding into a number between 0 (inclusive) and 128 (exclusive).

As shown in Table 5.5, our FOFE-based local detection method has obtained fairly strong performance in the KBP2015 dataset. The overall trilingual entity discovery performance is slightly better than the best systems participated in the official KBP2015 evaluation, with 73.9 vs. 72.4 as measured by F_1 scores. Outer and inner decodings are *longest-first* and *highest-first* respectively.

LANG	OVERALL			2016 BEST		
	P	R	F1	P	R	F1
ENG	0.836	0.680	0.750	0.846	0.710	0.772
CMN	0.789	0.625	0.698	0.789	0.737	0.762
SPA	0.835	0.602	0.700	0.839	0.656	0.736
ALL	0.819	0.639	**0.718**	0.802	0.704	0.756

Table 5.6: Official entity discovery performance of our methods on KBP2016 trilingual EDL track. Neither KBP2015 nor in-house data labels nominal mentions. Nominal mentions in Spanish are totally ignored since no training data is found for them.

	English	Chinese	Spanish	ALL
Eval	167	168	168	503

Table 5.7: Number of Documents in KBP2016

5.5.3 KBP2016 EDL task

In KBP2016, the trilingual EDL task is extended to detect nominal mentions of all 5 entity types for all three languages. In our experiments, for simplicity, we treat nominal mention types as some extra entity types and detect them along with named entities together with a single model.

training data	P	R	F_1
KBP2015	0.836	0.598	0.697
KBP2015 + WIKI	0.837	0.628	**0.718**
KBP2015 + in-house	0.836	0.680	**0.750**

Table 5.8: Our entity discovery official performance (English only) in KBP2016 is shown as a comparison of three models trained by different combinations of training data sets.

5.5.3.1 Data Description

No official training set is provided in KBP2016. We make use of three sets of training data:

- **Training and evaluation data in KBP2015**: as described in 5.5.2
- **Machine-labeled Wikipedia (WIKI)**: When terms or names are first mentioned in a Wikipedia article they are often linked to the corresponding Wikipedia page by hyperlinks, which clearly highlights the possible named entities with well-defined boundary in the text. We have developed a program to automatically map these hyperlinks into KBP annotations by exploring the infobox (if existing) of the destination page and/or examining the corresponding Freebase types. In this way, we have created a fairly large amount of weakly-supervised trilingual training data for the KBP2016 EDL task. Meanwhile, a gazeteer is created and used in KBP2016.
- **In-house dataset**: A set of 10,000 English and Chinese documents is manually labeled using some annotation rules similar to the KBP 2016 guidelines.

We split the available data into training, validation and evaluation sets in a ratio of 90:5:5. The models are trained for 256 epochs if the in-house data is not used, and 64 epochs otherwise.

5.5.3.2 Effect of various training data

In our first set of experiments, we investigate the effect of different training data sets on the final entity discovery performance. Different training runs are conducted on different combinations of the aforementioned data sources. In Table 5.8, we have summarized the official English entity discovery results from several systems we submitted to KBP2016 EDL evaluation round I and II. The first system, using only the KBP2015 data to train the model, has achieved 0.697 in F_1 score in the official KBP2016 English evaluation data. After adding the weakly labeled data, WIKI, we can see the entity discovery performance is improved to 0.718 in F_1 score. Moreover, we can see that it yields even better performance by using the KBP2015 data and the in-house data sets to train our models, giving 0.750 in F_1 score.

5.5.3.3 The official trilingual EDL performance in KBP2016

The official best results of our system are summarized in Table 5.6. We have broken down the system performance according to different languages and categories of entities (named or nominal). Our system, achieving 0.718 in F_1 score in the KBP2016 trilingual EDL track, ranks second among all participants. Note that our result is produced by a single system while the top system is a combination of two different models, each of which is based on 5-fold cross-validation [33].

6 Conclusion

6.1 Conclusion

In this thesis, we propose to use Fixed-size Ordinally Forgetting Encoding (FOFE), which demonstrates powerful modeling ability and great computational efficiency. Unique encoding is theoretically guaranteed. Any sequence of variable length can be encoded into a fixed-size vector. Dimensionality depends on vocabulary size only. It is much more friendly to machine learning algorithms that requires fixed-size input, such as FeedForward Neural Network (FFNN).

By applying FOFE, we nicely address two fundamental problems in NLP, Language Modeling (LM) and Named Entity Recognition (NER). Extensive experiments are conducted and the efficiency is evaluated in detail. To the best of our knowledge, this is the first piece of work of sequence modeling without any recurrent feedback in deep learning, which obtains great parallelism while maintains strong performance.

Inspired by the way how human finds and classifies, we also propose a local detection algorithm for NER. Contextual information is losslessly encoded by FOFE. Our local decision is made by global information. It ranked 2nd place and is the best single-model system in the EDL track of KBP2016 [25].

6.2 Future Works

There are several possible improvements, which could be done in the future. We divide them into three directions:

Language Modeling Since character FOFE is shown to be beneficial in NER, it is worth investigating whether character FOFE is a good complement or replacement of word embedding. Complicated network structures such as residual network seem to be good candidates of avoiding gradient vanishing. We may plug FOFE into other hidden layers.

Named Entity Recognition Chinese is currently modeled at character level. However, character can be further decomposed into radicals. The analogous role of character embedding in English could be played by radical embedding in Chinese.

Other Applications FOFE is a general approach for sequence modeling. We would like to explore other tasks requiring sequence modeling, such as Question Answering (QA) and Word Sense Disambiguation (WSD).

7 Accomplishment during MSc Study

The author of this work has the following publications accomplished during his MSc Study. This thesis is the extension of some of them.

7.1 Publication

- **A Local Detection Approach for Named Entity Recognition and Mention Detection** [53].

  In this paper, a novel approach for named entity recognition (NER) and mention detection (MD) in natural language processing (NLP) is investigated. It was accepted in the Conference of Association of Computational Linguistics ACL2017, and was awarded as one of the 22 **outstanding** papers.

- **Word Embeddings based on Fixed-Size Ordinally Forgetting Encoding** [48].

  In this paper, the authors propose to learn word embeddings based on the recent fixed-size ordinally forgetting encoding (FOFE) method. It was accepted in Conference on Empirical Methods in Natural Language Processing (EMNLP2017)

- **The YorkNRM Systems for Trilingual EDL Tasks at TAC KBP 2016** [54].

  This paper describes the YorkNRM systems submitted to the Trilingual Entity Detection and Linking (EDL) track in 2016 TAC Knowledge Base Population (KBP) contests.

- **The Fixed-Size Ordinally-Forgetting Encoding Method for Neural Network Language Models** [56].

  In this paper, the new fixed-size ordinally-forgetting encoding (FOFE) method is proposed, which can almost uniquely encode any variable-length sequence of words into a fixed-size representation. It was accepted in ACL2015.

- **Cardinality Estimation Using Neural Networks** [34].

  This paper presents a novel approach using neural networks to learn and approximate selectivity functions that take a bounded range on each column as input, effectively estimating selectivities for all relational operators. It was accepted in Center for Advanced Studies Conference (CASCON) organized by IBM in 2015. It was awarded the best student paper in the conference.

7.2 Patents

- **Cardinality estimation Using Artificial Neural Networks** [10, 11].

  This invention generates an artificial neural network to estimate the selectivity of one or more predicates in database management system (DBMS), which in turn benefits query optimization.

7.3 Awards

- **2nd Place of EDL in KBP2016** [25].

  The author submitted a system to the Entity Discover and Linking (EDL) track in Knowledge Base Population (KBP) contest organized by NIST. The submitted system ranked **second place** even though it was the author's first participation. It was also the best single-model system among all participants.

Bibliography

[1] R. E. Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[2] Y. Bengio and R. Ducharme. A neural probabilistic language model. *Proceeding of NIPS*, 13, 2001.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, mar 1994.

[5] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proc. of the Sixth Workshop on Very Large Corpora*, volume 182, 1998.

[6] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[7] X. Chen, Y. Wang, X. Liu, M. J. Gales, and P. C. Woodland. Efficient gpu-based training of recurrent neural network language models using spliced sentence bunch. In *Interspeech*, volume 14, pages 641–645, 2014.

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

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[10] V. Corvinelli, H. Liu, M. Xu, Z. Yu, and C. P. Zuzarte. Cardinality estimation using artificial neural networks, Mar. 5 2015. US Patent App. 14/639,157.

[11] V. Corvinelli, H. Liu, M. Xu, Z. Yu, and C. P. Zuzarte. Cardinality estimation using artificial neural networks, Mar. 23 2016. US Patent App. 15/078,302.

[12] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016.

[13] C. dos Santos, V. Guimaraes, R. Niterói, and R. de Janeiro. Boosting named entity recognition with neural character embeddings. In *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*, page 25. Association for Computational Linguistics (ACL), 2015.

[14] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, mar 1990.

[15] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics. JMLR W&CP:*, volume 15, pages 315–323, 2011.

[16] D. Graff and K. Chen. Chinese gigaword. *LDC Catalog No.: LDC2003T09, ISBN*, 1:58563–58230, 2005.

[17] A. Graves. Neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 15–35. Springer, 2012.

[18] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[19] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

[20] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[21] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

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

[23] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, jan 1991.

[24] Z. Huang, W. Xu, and K. Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[25] H. Ji, J. Nothman, and H. T. Dang. Overview of tac-kbp2016 tri-lingual EDL and its impact on end-to-end cold-start. In *Proceedings of Text Analysis Conference (TAC2016)*, 2016.

[26] H. Ji, J. Nothman, and B. Hachey. Overview of tac-kbp2015 tri-lingual entity discovery and linking. In *Proceedings of Text Analysis Conference (TAC2015)*, 2015.

[27] S. Ji, S. Vishwanathan, N. Satish, M. J. Anderson, and P. Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *arXiv preprint arXiv:1511.06909*, 2015.

[28] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[29] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, mar 1987.

[30] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. In *AAAI*. Citeseer, 2016.

[31] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

[32] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[33] D. Liu, W. Lin, S. Zhang, S. Wei, and H. Jiang. Neural networks models for entity discovery and linking. *arXiv preprint arXiv:1611.03558*, 2016.

[34] H. Liu, M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte. Cardinality estimation using neural networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, pages 53–59. IBM Corp., 2015.

[35] Q. Liu, H. Jiang, S. Wei, Z.-H. Ling, and Y. Hu. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1501–1511, Beijing, China, July 2015. Association for Computational Linguistics.

[36] M. Mahoney. Large text compression benchmark, 2011.

[37] A. A. Markov. Theory of algorithms. 1957.

[38] A. Mendonca, D. A. Graff, and D. DiPersio. *Spanish gigaword second edition*. Linguistic Data Consortium, 2009.

[39] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[40] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, may 2011.

[41] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[42] T. Mikolov and G. Zweig. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, dec 2012.

[43] T.-V. T. Nguyen, A. Moschitti, and G. Riccardi. Kernel-based reranking for named-entity extraction. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 901–909. Association for Computational Linguistics, 2010.

[44] R. Parker, D. Graff, J. Kong, K. Chen, and K. Maeda. English gigaword. *Linguistic Data Consortium*, 2011.

[45] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[46] M.-A. Rondeau and Y. Su. LSTM-based NeuroCRFs for named entity recognition. In *Interspeech 2016*, pages 665–669. International Speech Communication Association, sep 2016.

[47] E. F. T. K. Sang and F. D. Meulder. Introduction to the conll-2003 shared task: Language independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003,*, page 142147, 2003.

[48] J. Sanu, M. Xu, H. Jiang, and Q. Liu. Word embeddings based on fixed-size ordinally forgetting encoding. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 310–315. Association for Computational Linguistics, 2017.

[49] N. Shazeer, J. Pelemans, and C. Chelba. Sparse non-negative matrix language modeling for skip-grams. In *Proceedings Interspeech 2015*, pages 1428–1432, 2015.

[50] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[51] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, apr 1967.

[52] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[53] M. Xu, H. Jiang, and S. Watcharawittayakul. A local detection approach for named entity recognition and mention detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics (ACL), 2017.

[54] M. Xu, F. Wei, S. Watcharawittayakul, Y. Kang, and H. Jiang. The yorknrm systems for trilingual edl tasks at tac kbp 2016. In *Proceedings of the 9th Text Analysis Conference (TAC2016)*, 2016.

[55] S. Zhang, H. Jiang, M. Xu, J. Hou, and L. Dai. A fixed-size encoding method for variable-length sequences with its application to neural network language models. *arXiv preprint arXiv:1505.01504.*, 2015.

[56] S. Zhang, H. Jiang, M. Xu, J. Hou, and L. Dai. The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*

and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). Association for Computational Linguistics (ACL), 2015.