****



**研 究 生 毕 业 论 文**

**（申请工程硕士学位）**

|  |  |
| --- | --- |
| **论文题目** | 基于深度学习的信息抽取服务的设  计与实现 |
| **作者姓名** | 安 磊 |
| **学科、专业名称** | 工程硕士(软件工程方向) |
| **研究方向** | 软件工程 |
| **指导教师** | 伏晓　副教授 |

**2019年 04 月 11 日**

**学 号： MF1732001**

**论文答辩日期： 年 月 日**

**指 导 教 师：**

**基于深度学习的信息抽取服务的**

**设计与实现**

|  |  |
| --- | --- |
| **作 者:** | **安磊** |
| 指导教师: | **伏晓　副教授** |

|  |
| --- |
| **南京大学研究生毕业论文** |
| **(申请工程硕士学位)** |

|  |
| --- |
| **南京大学软件学院** |
| **2019年04月** |

**The Design and Implementation of Information Extraction Service Based on Deep Learning**

**An, Lei**

**Submitted in partial fulfillment of the requirements for the degree of Master of Engineering**

Supervised by

Associate Professor **Fu, Xiao**

Software Institute

**NANJING UNIVERSITY**

Nanjing, China

Apr, 2019

# 摘 要

随着人工智能技术的快速发展，越来越多的算法得到应用。自然语言处理作为人工智能领域中的一个重要分支一直是许多人关注的焦点，而知识图谱是自然语言处理中一个重要的研究方向。构建知识图谱的数据来源包括结构化文本和非结构化文本，作为构建知识图谱的重要数据来源之一的非结构化文本，如何从中提取高质量的信息成为一个研究热点。

面对上文所述的问题，一个能够提供高质量数据的信息抽取服务就显得十分重要。本文以从非结构化文本中进行信息抽取为背景，以构建知识图谱时所需的三元组作为目标，实现从非结构化数据到知识图谱三元组的数据处理与转换。

本文基于构建非结构化文本的信息抽取服务的具体项目。从模型设计和具体的使用场景出发，本文选择了两种不同的命名实体识别模型和一种关系抽取模型进行组合，并将模型封装在Docker容器中，部署到Kubernetes集群上构建一个信息抽取服务系统。

本文首先介绍了信息抽取的两个子任务命名实体识别和关系抽取的研究背景，阐述了本文的一些主要工作。然后本文对实现信息抽取服务中运用到的一些技术进行简单的介绍与说明。之后根据信息抽取服务系统的架构进行需求分析，包括详细的需求分析、项目的架构分析以及模型设计的详细分析。此后介绍了每个模块的具体实现过程，对所有模型进行实验分析，信息抽取服务进行部署。最后对整个项目进行总结，并对未来的研究工作进行规划与展望。

**关键词**：信息抽取，命名实体识别，关系抽取，深度学习，服务

# Abstract

With the rapid development of artificial intelligence, more and more algorithm were applied.Natural Language Processing has always been the focus of many people as an important branch in the field of artificial intelligence,and Knowledge Graph is an important research direction in Natural Language Processing.The data sources for constructing Knowledge Graph include structured text and unstructured text.As an unstructured text that is one of the important data sources for constructing Knowledge Graphs,how to extract high quality information from it becomes a research hot spot.

In the face of the problems described above,an information extraction service that provides high quality data is very important.Based on the background of information extraction from unstructured texts,this thesis aims to construct data graphs form the unstructured data to the Knowledge Graph triples.

This thesis is based on the specific project of information extraction service for unstructured texts.From the model design and specific usage scenarios,the thesis combine two different Named Entity Recognition models and one Relation Extraction model,package the models in a Docker container and deploy to the Kubernetes cluster to build an information extraction service system.

The thesis firstly introduces the research background of information extraction`s two sub-task Named Entity Recognition and Relation Extraction,explain some main work of this thesis.Then,the thesis briefly introduces and explains some technologies used in the implementation of information extraction service.After that,the thesis describes the requirements analysis,including detailed requirements analysis,project architecture analysis and detailed analysis of model design.Next,the thesis introduces the specific implementation process of each module,conduct experimental analysis on all models and deploy information extraction service.Finally,the thesis summaries the entire project,plans and forecasts the future research work.

**Keywords:**Information Extraction, Named Entity Recognition, Relation Extraction, Deep Learning, Service

目 录

[第一章 绪论](#_Toc4854_WPSOffice_Level1) [1](#_Toc4854_WPSOffice_Level1)

[1.1 项目背景](#_Toc443_WPSOffice_Level2) [1](#_Toc443_WPSOffice_Level2)

[1.2 国内外研究现状](#_Toc7644_WPSOffice_Level2) [2](#_Toc7644_WPSOffice_Level2)

[1.2.1 命名实体识别研究现状](#_Toc443_WPSOffice_Level3) [2](#_Toc443_WPSOffice_Level3)

[1.2.2 关系抽取研究现状](#_Toc7644_WPSOffice_Level3) [4](#_Toc7644_WPSOffice_Level3)

[1.3 本文主要的工作](#_Toc29847_WPSOffice_Level2) [6](#_Toc29847_WPSOffice_Level2)

[1.4 本文的组织结构](#_Toc14334_WPSOffice_Level2) [6](#_Toc14334_WPSOffice_Level2)

[第二章 技术综述](#_Toc443_WPSOffice_Level1) [8](#_Toc443_WPSOffice_Level1)

[2.1 引言](#_Toc31796_WPSOffice_Level2) [8](#_Toc31796_WPSOffice_Level2)

[2.2 词向量](#_Toc5318_WPSOffice_Level2) [8](#_Toc5318_WPSOffice_Level2)

[2.2.1 基于矩阵的分布表示](#_Toc29847_WPSOffice_Level3) [9](#_Toc29847_WPSOffice_Level3)

[2.2.2 基于神经网络的分布表示](#_Toc14334_WPSOffice_Level3) [9](#_Toc14334_WPSOffice_Level3)

[2.2.3 词向量的使用](#_Toc31796_WPSOffice_Level3) [10](#_Toc31796_WPSOffice_Level3)

[2.3 双向长短时记忆模型](#_Toc10788_WPSOffice_Level2) [11](#_Toc10788_WPSOffice_Level2)

[2.3.1 循环神经网络](#_Toc5318_WPSOffice_Level3) [11](#_Toc5318_WPSOffice_Level3)

[2.3.2 长短时记忆模型](#_Toc10788_WPSOffice_Level3) [12](#_Toc10788_WPSOffice_Level3)

[2.3.3 双向长短时记忆模型](#_Toc24764_WPSOffice_Level3) [13](#_Toc24764_WPSOffice_Level3)

[2.4 深度残差网络](#_Toc24764_WPSOffice_Level2) [13](#_Toc24764_WPSOffice_Level2)

[2.5 注意力机制](#_Toc23819_WPSOffice_Level2) [14](#_Toc23819_WPSOffice_Level2)

[2.5.1 Encoder-Decoder框架](#_Toc15178_WPSOffice_Level3) [14](#_Toc15178_WPSOffice_Level3)

[2.5.2 注意力模型](#_Toc19637_WPSOffice_Level3) [15](#_Toc19637_WPSOffice_Level3)

[2.6 远程监督](#_Toc15178_WPSOffice_Level2) [17](#_Toc15178_WPSOffice_Level2)

[2.7 深度学习框架Tensorflow](#_Toc19637_WPSOffice_Level2) [18](#_Toc19637_WPSOffice_Level2)

[2.9 本章小结](#_Toc25326_WPSOffice_Level2) [18](#_Toc25326_WPSOffice_Level2)

[第三章 基于深度学习的命名实体识别模型](#_Toc7644_WPSOffice_Level1) [19](#_Toc7644_WPSOffice_Level1)

[3.1 模型的整体架构](#_Toc5123_WPSOffice_Level2) [19](#_Toc5123_WPSOffice_Level2)

[3.2 模型的标注策略](#_Toc27641_WPSOffice_Level2) [20](#_Toc27641_WPSOffice_Level2)

[3.3 输入层](#_Toc3884_WPSOffice_Level2) [22](#_Toc3884_WPSOffice_Level2)

[3.3.1 字向量](#_Toc7744_WPSOffice_Level3) [22](#_Toc7744_WPSOffice_Level3)

[3.3.2 额外特征](#_Toc22826_WPSOffice_Level3) [23](#_Toc22826_WPSOffice_Level3)

[3.4 网络层](#_Toc7744_WPSOffice_Level2) [24](#_Toc7744_WPSOffice_Level2)

[3.5 输出层](#_Toc22826_WPSOffice_Level2) [25](#_Toc22826_WPSOffice_Level2)

[3.5.1 全连接层](#_Toc18750_WPSOffice_Level3) [25](#_Toc18750_WPSOffice_Level3)

[3.5.2 CRF层](#_Toc7396_WPSOffice_Level3) [25](#_Toc7396_WPSOffice_Level3)

[3.6 实验](#_Toc14704_WPSOffice_Level2) [27](#_Toc14704_WPSOffice_Level2)

[3.6.1 实验数据与采集](#_Toc6646_WPSOffice_Level3) [27](#_Toc6646_WPSOffice_Level3)

[3.6.2 实验评价指标](#_Toc30499_WPSOffice_Level3) [31](#_Toc30499_WPSOffice_Level3)

[3.6.3 模型训练的实现](#_Toc27939_WPSOffice_Level3) [31](#_Toc27939_WPSOffice_Level3)

[3.6.4 模型参数设置](#_Toc636_WPSOffice_Level3) [37](#_Toc636_WPSOffice_Level3)

[3.6.5 不同模型的性能对比](#_Toc15507_WPSOffice_Level3) [38](#_Toc15507_WPSOffice_Level3)

[3.7 本章小结](#_Toc18750_WPSOffice_Level2) [39](#_Toc18750_WPSOffice_Level2)

[第四章 基于深度学习的关系抽取模型](#_Toc29847_WPSOffice_Level1) [40](#_Toc29847_WPSOffice_Level1)

[4.1 模型的整体架构](#_Toc7396_WPSOffice_Level2) [40](#_Toc7396_WPSOffice_Level2)

[4.2 输入层](#_Toc6646_WPSOffice_Level2) [41](#_Toc6646_WPSOffice_Level2)

[4.2.1 字向量](#_Toc1426_WPSOffice_Level3) [41](#_Toc1426_WPSOffice_Level3)

[4.2.2 相对位置向量](#_Toc25146_WPSOffice_Level3) [41](#_Toc25146_WPSOffice_Level3)

[4.3 卷积层](#_Toc30499_WPSOffice_Level2) [42](#_Toc30499_WPSOffice_Level2)

[4.4 残差卷积块](#_Toc27939_WPSOffice_Level2) [42](#_Toc27939_WPSOffice_Level2)

[4.5 最大池化层](#_Toc636_WPSOffice_Level2) [44](#_Toc636_WPSOffice_Level2)

[4.6 输出层](#_Toc15507_WPSOffice_Level2) [44](#_Toc15507_WPSOffice_Level2)

[4.7 实验](#_Toc16217_WPSOffice_Level2) [44](#_Toc16217_WPSOffice_Level2)

[4.7.1 实验数据](#_Toc20460_WPSOffice_Level3) [44](#_Toc20460_WPSOffice_Level3)

[4.7.2 实验评价标准](#_Toc16453_WPSOffice_Level3) [46](#_Toc16453_WPSOffice_Level3)

[4.7.3 模型训练的实现](#_Toc27465_WPSOffice_Level3) [46](#_Toc27465_WPSOffice_Level3)

[4.7.4 模型参数设置](#_Toc11139_WPSOffice_Level3) [53](#_Toc11139_WPSOffice_Level3)

[4.7.1 不同模型的性能对比](#_Toc18104_WPSOffice_Level3) [55](#_Toc18104_WPSOffice_Level3)

[4.8 本章小结](#_Toc1426_WPSOffice_Level2) [55](#_Toc1426_WPSOffice_Level2)

[第五章 总结与展望](#_Toc14334_WPSOffice_Level1) [56](#_Toc14334_WPSOffice_Level1)

[5.1 总结](#_Toc25146_WPSOffice_Level2) [56](#_Toc25146_WPSOffice_Level2)

[5.2 进一步工作展望](#_Toc11644_WPSOffice_Level2) [56](#_Toc11644_WPSOffice_Level2)

[参 考 文 献](#_Toc31796_WPSOffice_Level1) [57](#_Toc31796_WPSOffice_Level1)

[致 谢](#_Toc5318_WPSOffice_Level1) [61](#_Toc5318_WPSOffice_Level1)

[版权及论文原创性说明](#_Toc24764_WPSOffice_Level1) [62](#_Toc24764_WPSOffice_Level1)

# 第一章 绪论

## 1.1 项目背景

自然语言处理（Nature Language Processing，简称NLP）是用计算机来处理、理解以及运用人类语言（如中文、英文等），它属于人工智能（Artificial Intelligence，简称AI）的一个发展方向，是计算机科学和语言学的交叉学科，又常被称为计算机语言学。由于自然语言是人类与其他动物的区别的主要标志，没有语言，人类的思维也就无从可谈，所以自然语言处理体现了人工智能的最高任务与境界。也就是说，只有当计算机具备了处理自然语言的能力时，机器才算实现了真正的智能。近十年，随着计算机设备和硬件的发展、计算设备的算力极大提升，自然语言处理的研究得到了很大的发展，并在各个领域都有了广泛应用。

信息抽取作为自然语言处理中的一项基础任务，在许多上层应用中都有着十分重要的影响。在上市公司新闻舆情分析中的关键信息就是新闻文本中的实体，如：公司、人、行业等实体信息以及实体间的关系信息，利用信息抽取中的命名实体识别和关系抽取就可以从新闻文本中抽出这些重要信息，进而对新闻舆情进行后续分析；在搜索引擎上进行搜索时的关键信息就是输入的搜索信息中的实体信息，但只通过分词进行输入文本的切分很容易把一个多词组成的命名实体切分成多个词，造成搜索不准确甚至搜索的结果完全不符合预期。如果能对输入文本进行命名实体识别就可以准确识别搜索的内容，再建立索引进行搜索，可以提高搜索的准确率也可以提升搜索的效率；在利用非结构化文本信息构建知识图谱的任务中关键信息就是非结构化文本中的关联实体以及实体间的关系组成的三元组，如：从“阿里巴巴集团的创始人是马云”这段描述中可以提取出“阿里巴巴，创始人，马云”这个三元组，可以表示“阿里巴巴的创始人是马云”这条信息。这里即用到了命名实体识别识别出了阿里巴巴和马云这两个实体，又用到了关系抽取来对阿里巴巴和马云这两个实体的关系进行识别和抽取。最后从海量非结构化文本中提取出大量三元组，经过实体消歧、知识融合后构建成一张知识图谱。

命名实体识别（Named Entity Recognition，简称NER）作为信息抽取中的一个基本任务，最早是在MUC-6（the Sixth of the Message Understanding Conference）上将其作为信息抽取的一项子任务引入测评中[MUC-6, 1996]。关系抽取（Relation Extraction，简称RE）作为信息抽取中的另一个基本任务，早在2000年就有基于句法解析增强的方法来实现关系抽取任务[Milleret al. 2000]。而本文同时结合命名实体识别和关系抽取两项任务作为利用非结构化文本中的三元组信息来构建知识图谱的关键环节，如何确保实体识别和关系抽取的准确性是构建一个高质量、高精度的知识图谱的关键所在。

## 1.2 国内外研究现状

## 1.2.1 命名实体识别研究现状

自从在MUC-6 [MUC-6, 1996]上将命名实体识别作为信息抽取的一个子任务之后，随着数据时代的到来，命名实体在各个领域的信息抽取任务中都得到了充分的利用和发展。

在国外命名实体识别的研究中，考虑到英文本身的特性，英文的命名实体识别只需要考虑词本身的特征而不需要关注别的问题例如分词等问题的影响，所以实现的难度相对较小，准确率也较高，

中文的命名实体识别由于中文文字内在的特殊性导致了在做命名实体识别之前要先做分词、句法分析等步骤，导致中文命名实体相对英文而言难度大大提高。在MUC-6中国内专家学者们把“人名”、“地名”、“组织机构名”作为实体的名称表达式，将这三类实体作为主要研究目标。在随后的研究中，逐渐地有人将“时间”、“货币”、“百分数”等也作为各种实体的表达式加入到命名实体识别的任务中来。

最早在命名实体识别任务中，主要有基于专家知识构建规则的方法以及基于数理统计的方法两类。基于词典和规则的方法主要是靠人工建立规则体系，在构建了大量基于专家知识的词法和语义规则之后，系统会根据规则对输入文本进行解析，基于人工构建的有限规则集合对文本中可能的命名实体进行推理和识别。虽然在特定的数据集上基于专家知识构建规则的方法相对于基于统计方法的命名实体识别而言准确度更高，但这种方法在数据量逐渐增大、数据内容逐渐复杂之后会变得不再可行，因为基于某一小部分语料构建的规则体系在别的语料上讲不再适用，人们无法去扩充和维护一个十分庞大的规则体系，且随着近些年机器学习理论的不断完善和计算机计算性能的提高，大部分学者开始转而研究基于统计方法的命名实体识别。

基于统计方法的命名实体识别主要有有监督（Supervised）、半监督（Semi-Supervised）的机器学习方法，其中有监督的机器学习方法在拥有大批量的标注语料的前提下在不同领域的文本中都拥有更高的识别准确率和精度，从而被更多的学者、从业者们所广泛使用。有监督学习方法中在命名实体识别任务中表现比较出色的方法主要有：基于隐马尔科夫模型（Hidden Markov Models，简称HMM）的命名实体识别模型[Bikel et al. 1999]、最大熵模型（Maximum Entropy Models，简称MEM）[Tsai et al. 2004]、条件随机场（Conditional Random Fields，简称CRFs）[McCallum et al. 2003]。基于统计方法的命名实体识别对特征选取的要求比较高，需要从文本中挖掘出对于实体识别有用的单词信息、上下文信息、句法信息、语义信息等作为特征。随着研究的不断深入，大量实验结果表明条件随机场结合了HMM和MEM的有点，成为中文命名实体识别任务中表现更优秀的统计学习方法。有学者针对条件随机场的特征选择与交叉组合进行了研究，通过实验得出不同的特征以及特征组合在训练时的贡献大小[张祝玉等, 2008]。

中文命名实体识别（Chinese Named Entity Recognition，简称CNER）相对于英文命名实体识别任务仍存在许多不同，由于中文本身的特点造成许多特殊的难点：

1. 中文单词的边界相比于英文较模糊

英文中的单词之间都有分隔符来标识边界，命名实体如人名、地名等单词的首字母为大写，这些信息能很好的为命名实体的识别做边界和位置标识，而中文单词没有这些信息。

1. 中文命名实体的词语结构更加复杂

有的类型的命名实体单词长度没有限制；不同实体有不同的组成结构（嵌套、简称、别名等）；音译词没有统一的构词规范；组织机构名和人名、地名会有很多交叉重复的地方（如：星环信息科技（上海）有限公司）。

1. 中文命名实体的一词多义与一义多词

随着互联网的极速发展，越来越多的网络词汇不断出现，很多原本的命名实体在现在都有了新的称呼（如称淘宝为某宝），这些词汇的新称呼的出现大大增加了命名实体识别的难度。

随着深度学习的影响不断增加，用深度学习方法解决命名实体识别任务也获得的一些显著的成果。[Wu et al. 2015]提出使用深度神经网络（Deep Neural Network，简称DNN）从语料中训练词向量，再输入到另一个深度神经网络中进行命名实体识别，实验结果好于传统机器学习中效果最好的CRF模型。[Z Huang et al. ]提出了使用双向长短时记忆模型(Bidirectional Long-Short Term Memory Model，简称Bi-LSTM)和条件随机场进行命名实体识别。还有一系列深度学习方法对命名实体识别任务做了很多尝试，在不同的数据集上相较于传统方法都有不小的提升。表1.1为目前命名实体识别的一些方法汇总。

|  |
| --- |
| 模型名称 |
| CRF |
| DNN |
| LSTM |
| BiLSTM |
| LSTM+CRF |
| BiLSTM+CRF |

表 1.1 命名实体识别方法汇总

## 1.2.2 关系抽取研究现状

关系抽取的方法主要有：

1. 有监督的学习方法

把关系抽取任务当做一个分类问题，从训练数据中设计并提取有效的特征并训练学习各种分类模型进行关系预测。需要大量的人工标注语料，比较耗时耗力，但预测效果较好。

1. 半监督的学习方法

先人工设定若干种子实例，再迭代地从文本中抽取相应的关系模板和更多关系实例。

1. 无监督的学习方法

假设有相同语义的命名实体同样拥有相似的上下文信息，可根据每个命名实体对的上下文信息来代表它们的语义关系，并对所有实体对的语义进行聚类。

与另外两种方法相比，有监督的学习方法在抽取的准确率和召回率上都有着更好的表现。

自从2000年Miller等提出基于句法解析增强的方法来实现关系抽取[Milleret al. 2000]后，越来越多的实体间关系抽取方法被提出：基于逻辑回归的方法[Kambhatla et al. 2004]、基于核函数的方法[Zhao and Grishman 2005]、基于条件随机场的方法[Culotta et al. 2006]。在有监督学习中针对需要大量人工标注的情况下，Mintz等人[Mintz et al. 2009]提出了使用远程监督（Distant Supervision）的方法来扩充标注语料，可以有效解决关系抽取的标注数据规模问题。

随着计算机硬件和计算能力的极速发展，深度学习开始逐渐被学术界和业界所关注，很多学者也开始将深度学习运用到关系抽取任务中。[Socher et al. 2012]提出使用递归神经网络（Recurrent Neural Network，简称RNN）来解决关系抽取问题，通过递归神经网络学习到句子的词汇特征、句法特征、语义特征再用于关系分类和抽取。[Zeng et al. 2014]提出使用卷积神经网络（Convolutional Neural Network，简称CNN）来解决关系抽取问题，采用词向量和词的相对位置作为卷积神经网络的输入，通过卷积、池化、非线性计算等操作得到句子表示并用于关系抽取。[Santos et al. 2015]提出了一种新的卷积神经网络结构用于解决关系抽取问题，在这个新的结构中采用了新的Ranking损失函数并得到了更好的效果。[Miwa er al. 2016]提出一种基于端到端（End to End）神经网络的关系抽取模型，使用双向长短时记忆模型和树形长短时记忆模型同时对实体和句子进行建模。[Lin et al. 2016]提出基于句子级注意力机制的神经网络模型来解决关系抽取问题，文中的方法可以根据不同关系为每个实体对分配不同的权重。关系抽取的一些方法汇总如表1.2所示。

|  |
| --- |
| 模型名称 |
| RNN |
| CNN |
| CNN+Attention |
| BiLSTM+Attention |
| PCNN+Attention |
| ResCNN |

表1.2 关系抽取方法汇总

## 1.3 本文主要的工作

本文在总结现有的一些研究成果的基础上，结合命名实体识别和关系抽取两项信息抽取的基本任务，从多个深度学习模型中选择合适的模型，结合自己人工标注的部分金融领域新闻文本语料结合远程监督进行模型训练，将训练好的模型固化成pb模型，把模型进行组合封装到Docker中，并部署在集群上来提供信息抽取服务。

本文的主要工作有：

1. 对命名实体识别和关系抽取的研究现状和相关技术进行综述，分析了命名实体识别和关系抽取在构建知识图谱前的信息抽取环节的重要性。
2. 对本文的信息抽取服务系统进行了详细设计，包括需求分析、产品功能分析、本文选择的信息抽取模型的详细设计以及系统中每个模块的详细设计。
3. 利用远程监督扩充人工标注数据，缓解人工标注数据费时费力的缺点。
4. 将训练好的模型封装成ModelServer部署到集群上，构建一套信息抽取服务系统。

## 1.4 本文的组织结构

本文的各章节内容安排如下：

第一章 绪论。本章介绍了论文的项目背景，信息抽取任务中的两个子任务命名实体识别和关系抽取，国内外在这两个方向上的研究成果，本文所做的主要工作和论文的组织结构。

第二章 技术综述。本章详细介绍了论文涉及的相关技术，包括词向量、双向长短时记忆模型、深度残差网络、注意力机制（attention）、远程监督等。

第三章 基于深度学习的命名实体识别模型。本章详细介绍了本文选择的命名实体识别模型的各层网络结构和参数设定的对比实验和分析。

第四章 基于深度学习的关系抽取模型。本章详细介绍了本文选择的关系抽取模型的各层网络结构和参数设定的对比实验和分析。

第五章 论文的总结与展望。总结在论文期间做了哪些工作，在信息抽取任务中未来的可改进的地方以及对接下来的需要做的工作的进一步展望。

# 第二章 技术综述

## 2.1 引言

本章主要介绍了本文中实现的基于深度学习的命名实体识别和关系抽取所涉及的相关技术，包括词向量、双向长短时记忆模型、深度残差网络、注意力机制和远程监督。针对这些模型和算法的各自特点和实际使用过程，分析了它们的优缺点。

## 2.2 词向量

人类可以轻松理解所有的自然语言，而计算机是不能直接识别自然语言的，要想让计算机能够读懂自然语言，必须要对自然语言进行进一步处理才行，因此词向量（Word Embedding）的概念应运而生。计算机的通用语言是二进制数字，所以词向量就是一些包含了自然语言中词汇和语义信息的计算机能够识别的数字。

最早的词表示方法是One-Hot编码，即用N个状态编码器对N个状态进行编码，这种方法会造成一篇篇幅有n个字的文本，会用一个n\*n的矩阵来表示，每个n维向量中只有它对应的那个字的位置上是1，其余位置都是0。这种词表示方法在传统机器学习中，如条件随机场、逻辑回归、最大熵、支持向量机等模型中可以较好的完成一些自然语言处理的任务。但是这种词表示方法中默认每个单词之间是独立存在的，忽略了文本中单词间的语义关联，且通常向量的维度会很高，在用深度学习处理自然语言处理问题时如果使用One-Hot编码容易造成“维度灾难”[Bengio et al. 2002]。

除了忽略了单词间语义信息的One-Hot编码，[Hinton, 1986]提出了一种新的词表示方法：分布式表示法（Distributed Reputation）。基于[Harris, 1954]提出的分布假说：上下文相似的词，其语义也相似。词的分布式表示主要分为2类：基于矩阵的分布表示和基于神经网络的分布表示。

## 2.2.1 基于矩阵的分布表示

基于矩阵的分布表示主要是构建“词-上下文”矩阵，通过某种技术从矩阵中获取词的分布表示。其中，矩阵的每一行都表示一个词，列表示上下文信息。常见的上下文有：

1. 文档，即“词-文档”矩阵。
2. 上下文的每个词，即“词-词”矩阵。
3. n-元词组，即“词-n-元词组”矩阵。

矩阵中的每个元素表示词和上下文共现的次数，可以利用词频-逆文本频率指数（TF-IDF）、取对数等技巧进行加权和平滑。矩阵维度较高且较稀疏的情况还可以利用奇异值分解（SVD）和非负矩阵分解（NMF）等方法进行分解降维使之变成低维稠密矩阵。基于矩阵的分布表示的典型代表是Global Vector模型（GloVe模型）[Pennington et al. 2014]。

## 2.2.2 基于神经网络的分布表示

基于神经网络的分布表示一般通过神经网络训练语言模型得到的产物，[Bengio et al. 2006]提出了一种神经网络语言模型（Neural Network Language Model。简称NNLM），考虑对语言模型进行建模。但是NNLM中词向量知识语言模型训练得到的副产物，并没有指出哪一套向量作为词向量的效果更好，所以[Mikolov et al. 2013a; Mikolov et al. 2013b]在NNLM等模型的基础之上，提出了word2vec模型，在word2vec模型中设计了CBOW（Continuous Bag-of-Words）和Skip-gram两个模型，高效地训练获取词向量，CBOW和Skip-gram的示意图如图2.1所示。



图2.1 CBOW和Skip-gram示意图

CBOW模型是根据上下文的信息来预测中间的目标词。CBOW模型对NNLM模型做了简化：隐藏层不再是上文各词词向量的拼接，而是使用上下文各词词向量的平均值，减少了计算量；去掉了tanh隐藏层，提升了模型的训练速度。CBOW模型包含3层：输入层、投影层和输出层。输入层共有n-1个词的one-hot表示作为词向量，组成上下文的表示：公式1。投影层将输入层的n-1个向量做均值计算：公式2。输出层一共V个节点，第i个节点表示中心词是词wi的概率。CBOW模型根据上下文的表示，直接对目标词进行预测：公式3。对于整个语料库而言，CBOW的优化目标为最大化：公式4

Skip-gram模型与CBOW模型刚好相反，它是根据一个单词来预测它的上下文信息。Skip-gram模型训练一个双层神经网络，模型输入为一个词，来预测这个词的前n个词和后n个词出现的概率。Skip-gram和CBOW一样没有隐藏层。Skip-gram的优化目标为最大化：公式5

## 2.2.3 词向量的使用

目前可供使用的训练词向量的方法有很多，因为本文所使用的编程语言是Python，Python编程环境中有Gensim库可以直接调用models.word2vec来训练词向量。[Radford et al. 2017]提出使用任务相关的语料训练得到的词向量相对于别的词向量在特定任务中会有更好的效果。由于本文所涉及的项目是针对金融领域的文本进行信息抽取且命名实体识别的序列标注任务是针对每个字来进行标注的，所以使用金融相关文本作为训练字向量的文本语料，训练了一个包含50102个字的100维的字向量模型。

## 2.3 双向长短时记忆模型

## 2.3.1 循环神经网络

循环神经网络（Recurrent Neural Networks，简称RNN）[Rumelhart et al. 1986]指的是一个序列当前的输出和之前的输出存在某种关联的神经网络。具体的表现形式为网络会对前面的信息进行记忆，保存在网络的内部状态中，并应用于当前输出的计算中。所以循环神经网络常常被用于对序列数据进行建模。循环神经网络的结构图如图2.2所示。

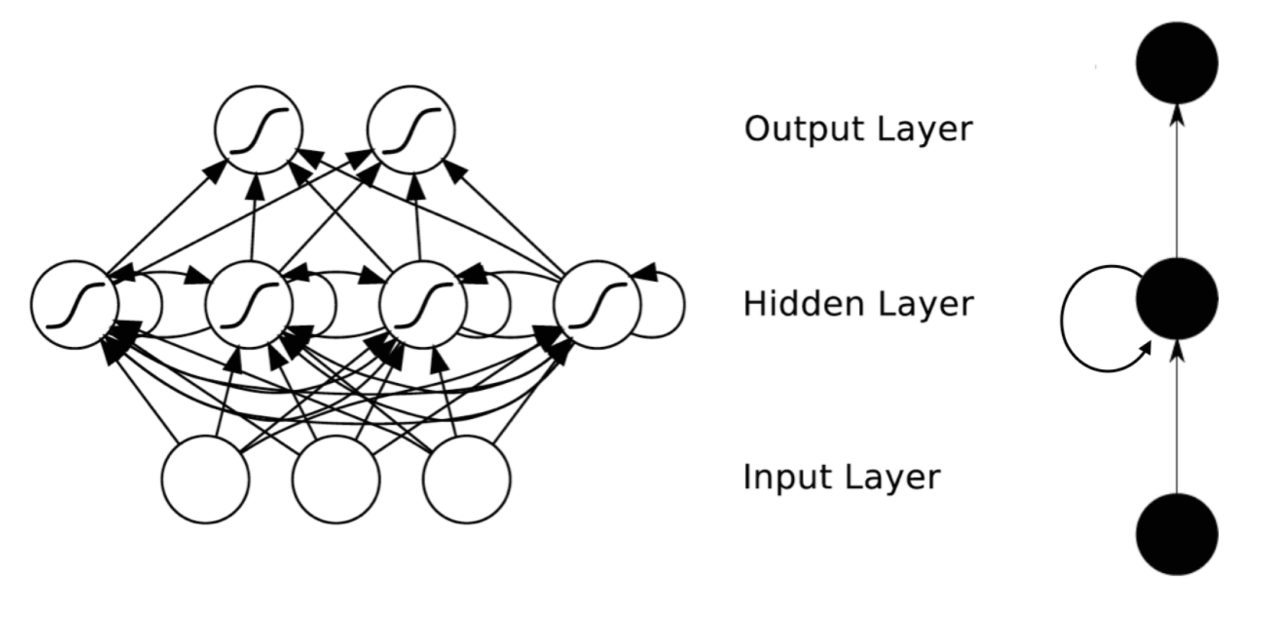


图2.2 循环神经网络的结构图

理论上循环神经网络能够对任何长度的序列数据进行处理，但是在实践中发现”梯度消失”会影响循环神经网络的训练，随着神经网络层数的增加，“梯度消失”的风险就会越大，当层数很大时，底层的神经元将接收不到返回的信号，出现“梯度消失”的问题。而长短时记忆模型（Long-Short Term Model，简称LSTM）[Hochreiter et al. 1997]的出现缓解了这个问题。

## 2.3.2 长短时记忆模型

长短时记忆模型的提出是为了解决循环神经网络的“梯度消失”问题，它之所以能够缓解这个问题主要取决于LSTM特有的门型结构，如图2.3所示。



图2.3 LSTM结构示意图

可以看到，LSTM主要包括遗忘门、输入门和输出门等结构，其中：

遗忘门：公式7

输入门：公式8

输出门：公式9

新记忆单元：公式10

最终记忆单元：公式11

隐藏层：公式12

三个不同的门型结构和记忆单元是LSTM的核心所在，记忆单元负责保存历史和当前的重要信息，遗忘门负责控制选择性丢弃之前的记忆单元中的一些信息，输入门负责控制当前的输入中的某些信息加入到记忆单元中，输出门负责控制记忆单元中的哪些信息可以进行输出。这些门控结构可以帮助LSTM在长时和短时的记忆中都能处理并解决数据依赖以及选择是否丢弃信息（保留重要信息并丢弃不重要的信息）。三个门相互合作是LSTM能够解决RNN梯度消失/梯度爆炸的关键所在。

## 2.3.3 双向长短时记忆模型

一般我们所说的RNN、LSTM都是单向的，也就是说在一个方向上进行隐藏状态的传递，然而在某些任务中需要考虑的不仅仅是当前输入单词前面的文本信息，还需要考虑当前输入单词后面的文本信息。双向长短时记忆模型就可以解决这种类型的问题，它不仅可以从前向后传递隐藏状态的信息，还可以从向前传递隐藏状态的信息，这样在计算输出时考虑的就是完整的上下文信息。可以把双向长短时记忆模型看成是两个方向相反的LSTM的叠加，在分别做计算后把两个状态向量拼接得到最后的输出。

## 2.4 深度残差网络

深度学习中对于网络深度加深遇到的主要问题是梯度消失和梯度爆炸，传统对应的解决方案是数据的初始化（normalized initialization）和（batch normalization）正则化。

深度残差网络（Deep Residual Network，简称DRN）[He K M. et al. 2015]的提出是源于一个“反常”的现象：在训练深度神经网络的时候，训练误差和测试误差都随着网络深度的增加而增加（Degradation）。在自然语言处理中，利用网络结构训练文本数据，浅层的网络更容易学习到文本中的词汇信息，而深层的网络更容易学到文本中的语义信息，但是由于层数的增加会导致训练误差的累积，残差网络将浅层的网络与深层的网络进行级联再输出，很好地控制了训练误差的累积，在自然语言处理的各项任务中取得了很好的效果。

假设深度网络中的某隐藏层为公式13，如果多个非线性层组合可以近似于一个复杂函数，那么同样可以认为隐藏层的残差近似于某个复杂函数，那么可以将隐藏层表示为公式13。这样一来就可以得到一种全新的残差结构单元，如图2.4所示。



图2.4 残差结构单元示意图

残差单元的输出由多个卷积层级联的输出和输入元素相加（维度相同的情况下），再经过ReLU激活后得到。将多个这样的结构级联起来，就得到了残差网络。

可以注意到残差网络有这样几个特点：1.网络较瘦，控制了参数数量；2.存在明显层级，特征图个数逐层递进，保证输出特征表达能力；3.使用了较少的池化层，大量使用下采样，提高传播效率；4.没有使用Dropout，利用BN和全局平均池化进行正则化，加快了训练速度。

## 2.5 注意力机制

深度学习中的注意力机制从本质上讲和人类的选择性视觉注意力机制类似，核心目标是从众多信息中选择出对当前任务的目标更关键的信息。

## 2.5.1 Encoder-Decoder框架

要深入理解深度学习中的注意力机制，就必须先介绍一下Encoder-Decoder框架，因为目前大多数的注意力机制的实现主要依靠Encoder-Decoder框架。

Encoder-Decoder框架是深度学习的一种研究模式，应用在很多场景之中。图2.5是文本处理中Encoder-Decoder框架的一种简单表示。



图2.5 Encoder-Decoder框架简单示意图

文本处理任务中可以把Encoder-Decoder框架看作一个由一篇文本生成另一篇文本的通用的模型。对于文本对[Source, Target]，目标是输入Source，通过模型最后生成Target。Encoder的作用是对输入文本Source=(x1,x2,...,xm)进行编码，通过非线性变化转化为中间语义表示C=F(x1,x2,...,xm)。再通过Decoder对中间语义表示C和已经生成的历史信息y1,y2,...,yt-1来生成t时刻要生成的单词yt：yt=G(C,y1,y2,...,yt-1)。每个yt依次生成，最后看上去整个模型就是输入Source最后生成了目标文本Target。

## 2.5.2 注意力模型

在Encoder-Decoder模型中，在生成目标句子的单词时，无论生成哪个单词，它们使用的中间语义表示C都是一样的没有任何区别。而中间语义表示C是由Source经过Encoder编码而成，这意味着无论生成哪个单词，Source中的任意单词对生成它们时所产生的影响大小都是相同的。当输入文本较短时这样的缺点暴露的还不够明显，一旦输入文本很长，此时所有的语义完全通过一个中间语义向量来表示将会丢失很多重要信息。这也是为什么要引入注意力模型：让模型更多地把注意力集中在对生成输出更重要的输入单词上。引入注意力机制后的Encoder-Decoder模型示意图如图2.6所示。



图2.6 引入注意力机制后的Encoder-Decoder示意图

其中Ci的计算公式为：公式6 。其中Lx代表输入Source的长度，aij代表在Target输入第i个单词时Source中第j个单词的注意力分配系数，hj是Source中第j个词的语义编码。中间语义表示的形成过程类似于图2.7所示。



图2.7 Attention的形成过程

本质上来说，注意力机制就是计算每一个输入单词对所有输入单词的相似度，得到一个分数即权重，然后根据这个权重对所有的输入进行加权求和。如图2.8所示说明了注意力分配概率分布值的通用计算过程。



图2.8 注意力分配概率计算图

## 2.6 远程监督

远程监督（Distant Supervision）是目前关系抽取中比较常见的一种做法。它既不是传统意义上的监督学习，也不是无监督学习，它属于一种用KB（Knowledge Base）去对齐朴素文本的标注方法：1.使用NET（Named Entity Tagger）标注；2.对已标注的金融领域文本中出现的三元组提取特征（从所有出现该三元组的句子中），构造训练数据；3.采用多类别逻辑回归进行分类。

在测试使用时，先使用命名实体识别模型对目标文本中的命名实体进行标注，抽取其中的命名实体对和特征。如果多个句子的命名实体对一样，则把它们的特征合并在同一个特征向量中，再利用逻辑回归分类器对关系名称进行识别。

简单来说，如果训练语料中的句子所包含的实体对在已有的KB中有关系的体现，则认为语料库中所包含相同实体对的句子都表达相应关系。虽然这样做可以减少关系抽取任务对人工标注数据的依赖，但是会引入很多噪声数据。因为两个实体之间可能有别的关系（不同于KB中已有的关系）或者是没有关系，这样的训练数据就会对我们的关系抽取任务产生一定影响。

近些年来，也有很多国内外学者在研究如何降低远程监督中的噪声数据对关系抽取造成的影响，如使用Multi-instance从训练集中抽取置信度较高的训练样例来训练模型、利用Attention机制对数据进行全方位的权重计算从而得到全面而不失“选择”的训练数据等方法。

本文在利用远程监督处理标注数据时，结合Attention机制和残差网络（ResNet）共同作用减少噪声数据对关系抽取模型带来的影响。

## 2.7 深度学习框架Tensorflow

Tensorflow既是一个实现机器学习算法的接口，也是执行机器学习算法的框架。它前端支持Python、C++、Go、Java等多种开发语言，后端使用C++、CUDA等写成。除了执行深度学习算法，Tensorflow还可以用来实现很多传统机器学习算法，如逻辑回归、随机森林等。

Tensorflow使用数据流式图来规划计算流程，可以把计算映射到不同的硬件甚至是操作系统。Tensorflow的计算可以表示为有状态的数据流式图，可以让用户简单地实现并行计算，同时使用不用的硬件资源进行训练，同步或异步地更新全局共享的模型参数和状态。

## 2.9 本章小结

本章主要介绍了本文对信息抽取中的命名实体识别和关系抽取的研究中涉及的相关技术。先是介绍了作为模型输入的深度学习自然语言问题的基础：词向量；之后又分别介绍了双向长短时记忆模型、深度残差网络、注意力机制、远程监督等，并介绍了本文实现模型使用的深度学习框架Tensorflow。

# 

# 第三章 基于深度学习的命名实体识别模型

在总结和归纳了各种已发表的命名实体识别技术相关的论文后，本章基于已有的相关技术与模型，选择了双向长短时记忆模型+条件随机场来解决命名实体识别问题。本章讲详细介绍模型的每一部分具体的网络结构和各种参数的对比实验。

## 3.1 模型的整体架构

本文使用了一个端到端的模型来解决命名实体识别问题。本章采用了Bi-LSTM+CRF的模型结构来实现命名实体识别，模型示意图如图3.1所示。



图3.1 Bi-LSTM+CRF模型结构示意图

先把输入文本按字组成一个输入文本的字典，字典中每个字都有一个唯一的ID（类似于唯一标识符），再把输入文本对照字典得到每个字在字典上的相对位置和字向量进行嵌入（embedding）操作，得到每一个字的向量。本文使用的字向量模型是基于金融领域上市公司新闻、金融公告、行业信息等中文语料，使用Gensim库中的word2vec模型训练得到的50102个字的100维字向量。把输入文本的字向量和词汇特征作为模型的输入，在中间网络层上使用Bi-LSTM，通过前向隐藏状态传递和后向隐藏状态传递分别对输入序列进行编码操作（即进行特征提取），有效地利用了前向和后向的所有信息（即上下文信息），最后利用CRF层从这些特征中计算出序列中每一个元素的标签。如图3.2所示为模型训练的具体步骤。

|  |
| --- |
| 1：for each epoch do  2： for each batch do  3： 1) Bi-LSTM-CRF model forward pass:  4： forward pass for forward state LSTM  5： forward pass for backward state LSTM  6： 2) CRF layer forward and backward pass  7： 3) Bi-LSTM-CRF model pass:  8： forward pass for forward state LSTM  9： forward pass for backward state LSTM  10： 4) update parameters  11： end for  12：end for |

图3.2 Bi-LSTM-CRF模型训练具体步骤

## 3.2 模型的标注策略

由于本文所研究的是金融领域下的信息抽取任务，所以标注语料都是来自于金融领域的新闻文本、上市公司信息、金融公告等。在做金融领域的命名实体识别时，本文将实体的类型分为公司（C）、人（P）、组织机构（INS）、行业（IND）、地点（L）、主营业务（PRO）六大类。本文对中文语料采用的是BIO标注方式，具体格式如图3.3所示。

|  |
| --- |
| 星 B-C  环 I-C  科 I-C  技 I-C  C O  E O  O O  孙 B-P  元 I-P  浩 I-P  将 O  亲 O  自 O  来 O  南 B-INS  京 I-INS  大 I-INS  学 I-INS  进 O  行 O  宣 O  讲 O  活 O  动 O  。 O |

图3.3 BIO标注格式示例

B-标签1作为命名实体第一个字的标签，如果命名实体的字数大于1，则后面的字的标签都为I-标签1；如果命名实体只有一个字的话，则这个字的标签为B-标签2（中文中一个字作为命名实体的情况较少）；如果文本中的字不为命名实体的组成部分，则标签为O。所以最终得到的标签有：B-C、B-P、B-L、B-IND、B-INS、B-PRO、I-C、I-P、I-L、I-IND、I-INS、I-PRO、O共13种标签类型。

这样做的好处是能够把命名实体和非命名实体的字词有效地分隔开来，有利于之后的输出层进行标签分类。

## 3.3 输入层

模型的输入层是将文本输入的单词序列进行处理转化成单词的特征向量序列汇总后传输进网络层进行处理，而单词的特征向量是由输入文本的字向量和额外特征向量拼接组成的。如图3.4所示。



图3.4 句子级别命名实体识别模型架构图

### 3.3.1 字向量

### 在利用深度学习模型解决中文自然语言处理问题时，通常将一个单词转化成词向量的形式作为模型的输入，由于本文的任务是中文命名实体识别，它属于一项序列标注任务，标注的对象是文本中的每个字，得到的输出也是针对每个字进行一个预测标签的计算，所以本文使用的是字向量。

使用预训练好的字向量能够使输入序列包含了字符的语义信息，所以能够提高模型的效果。图3.5所示为词向量使用示意图。



图3.5 词向量使用示意图

### 3.3.2 额外特征

对于输入文本，除了可以用文本的字向量作为其特征向量外，还可以引入额外的特征向量。本文在选取额外的特征向量时，把输入文本的词汇信息作为额外特征。本文采用jieba分词（jieba分词为Pyhton编程环境下的一个分词工具包），并结合金融领域的自定义词典对输入文本进行分词，对分词后的文本进行标记：若单词的字符长度大于等于3，则单词的第一个字符被标记为1，最后一个字符被标记为3，中间的所有字符被标记为2；如果单词的字符长度等于2，则单词的第一个字符被标记为1，第二个字符被标记为2；如果单词的字符长度为1（即1个字），则这个字符被标记为0，如图3.6所示。

|  |
| --- |
| seg\_feature:  北京市海淀区国资委已与江苏银行北京分行签署战略合作协议  [1,2,3,1,2,3,1,2,3,0,0,1,2,2,2,2,2,2,3,1,3,1,2,2,2,2,3] |

图3.6 词汇信息示例

最后将输入文本的字向量和分词后的词汇特征向量做拼接后作为模型的输入层结果输入到网络层中进行编码操作。

由于深度学习模型中模型的参数很多，而标注好的训练样本又比较少，所以训练深度学习模型的时候最容易出现的问题就是模型的过拟合问题，模型的过拟合具体表现在：模型在训练数据上的损失函数值较小，预测的准确率较高。但在测试数据上的损失函数值较大，预测的准确率较低。本文在输入层和网络层之间加了Dropout来缓解神经网络过拟合的发生[Hinton et al. 2012]，在一定程度上起到了正则化的作用。

Dropout之所以可以缓解神经网络过拟合的发生，是因为在前向传播时，Dropout让某个神经元的激活值以一定的概率p停止工作，这样可以使得模型的泛化能力变强，因为这样可以让模型不会太依赖某些局部的特征。

## 3.4 网络层

一般来说同一个任务的不同模型主要的差异就是网络层选择的编码器不同，本文采用的是Bi-LSTM作为编码器对模型输入的特征向量进行处理，得到每个字符的预测特征向量，再传输给CRF层计算得到每个字符最终的预测标签。



图3.7 基于Bi-LSTM的编码层结构图

如图3.7是一个基于双向长短时记忆模型的编码层的结构图。假设图中维度为n的向量xt代表当前时刻输入的一个字对应的总特征向量。先将xt输入到网络层的双向长短时记忆模型中，分别与前向层、后向层中传递过来的t时刻的前、后的状态信息进行计算，得到t时刻前、后时刻的状态对当前状态的影响。再把前向层和后向层的当前时刻的输出做拼接，得到长度为2\*n的向量作为网络层的输出。

## 3.5 输出层

输出层的作用是把网络层编码后的输出进行处理后得到最终的每个字符对应的预测标签。本文的输出层是由Softmax全连接层和CRF层联合构成。

### 3.5.1 全连接层

假设输出层得到的网络层输出为X=（x1, x2,...,xm），其中xt为当前时刻的字符对应的模型预测向量，xt会经过如公式3.5的计算得到输入对应的状态分数矩阵，示意图如图3.8所示。



图3.8 全连接层示意图

其中W的维度为(标签类别数量, 字向量的维度)，偏置项b的维度为标签类别数量。对于输出层来说需要训练迭代更新的参数就是W和b。

### 3.5.2 CRF层

在一般的命名实体识别模型中，即使不加入CRF层，即用Softmax层计算得到的状态分数矩阵并取其中的最大值作为预测标签的依据也可以得到最终的分类标签结果，但是那样的结果会存在一些问题导致最终的分类效果不好，如图3.9所示。



图3.9 未使用CRF层的模型示例

显然图中的分类结果并不准确，这也是为什么需要引入CRF层：CRF层可以学习到句子的约束条件，CRF层在训练数据时可以自动学习到这些约束，保证最终预测结果是有效的。

其中可能的约束条件有：

1. 句子的开头应该是“B-”或者“O”，而不是“I-”。
2. “B-标签1 I-标签2 I-标签3...”这样的模式中，标签1、标签2、标签3应该是同一种实体类别。比如，“B-P I-P I-P”这样的序列是对的，而“B-P I-C I-C”这样的序列是错误的。

有了这些约束条件，模型预测出错的概率将会大大降低。

CRF层接收到全连接层计算得到的状态分数矩阵为SC，sci,j代表序列中第i个字符预测为标签j的得分。CRF层中除了状态分数矩阵S，还有一个非常重要的分数---转移分数t。转移分数指的是序列从标签1转移到标签2的分数，如“B-C”转移到“I-C”的分数。为了使转移分数矩阵具有更好的鲁棒性，本文在每个句子的开头加上START标签作为句子开始的标记（所以此时的标签类别个数为14个），在转移分数矩阵中START的分数为0。表3.1为转移分数矩阵的示例。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | START | B-P | I-P | B-INS | I-INS | O |
| START | 0 | 0.8 | 0.007 | 0.7 | 0.0008 | 0.9 |
| B-P | 0 | 0.6 | 0.9 | 0.2 | 0.0006 | 0.6 |
| I-P | -1 | 0.5 | 0.53 | 0.55 | 0.0003 | 0.85 |
| B-INS | 0.9 | 0.45 | 0.0003 | 0.25 | 0.8 | 0.77 |
| I-INS | -0.9 | 0.45 | 0.007 | 0.7 | 0.65 | 0.76 |
| O | 0 | 0.65 | 0.0007 | 0.7 | 0.0008 | 0.9 |

表3.1 转移分数矩阵的示例

从表中可以看出转移矩阵已经学到了一些有用的约束信息，如：从“START”到“I-P”或者“I-C”的分数很低，但到“B-P”和“B-C”的分数很高，可以看出句子的第一个单词的标签应该是“B-”或者“O”，而不是“I-”；“O”后面接“I-标签”的分数很低，可以看出命名实体的开头应该是“B-”而不是“I-”。

设公式3.6为CRF层的转移分数矩阵，ti,j代表从标签i转移到标签j的分数。转移分数矩阵作为Bi-LSTM+CRF模型中的一个参数，在训练模型之前可以随机初始化得到这样一个矩阵T，T中的分数会随着模型训练的更新迭代不断更新，也就是说CRF层可以自己学到这些约束。

假设输入序列X的可能预测序列类别为Y=（y1,y2,...,yn）,每种可能预测序列标签的分数为pi，共有n条路径，那么路径的总分为：公式3.7

其中公式3.8，

那么得到正确的预测序列标签的条件概率为P(y|X):公式3.10

随着模型训练的进行，模型的参数不断更新，使得真实的预测序列标签占所有预测路径的比例越来越大。取Ptotal的负对数作为损失函数Loss，训练的目标为最小化损失函数以此来调整模型参数，即：公式3.9。训练完成后，使用viterbi\_decode()调用维特比算法计算出概率最高的序列标签作为预测序列标签。

## 3.6 实验

本节实验将从实验数据、实验评价标准、模型训练、模型参数设置、实验结果及对比等方面详细介绍整个实验的实现过程及结果分析。

### 3.6.1 实验数据与采集

本文所实现的项目是基于金融领域的信息抽取任务，所以实验数据是来自于金融界、新浪财经、东方财富、每日经济新闻等网站的上市公司的新闻文本、公司公告、最新资讯等文本信息，而采集这些数据的方式是数据爬虫。

### 3.6.1.1 数据爬虫

本文的数据爬虫利用Scrapy爬虫框架实现。Scrapy爬虫框架的架构图如图3.10所示。



图3.10 Scrapy爬虫框架架构图

其中Scrapy Engine（引擎）负责所有模块的中间通讯、信号、数据传递；Scheduler（调度器）负责接收引擎发来的Request请求，按照一定的方式整理排列、入队，当引擎需要的时候交还给引擎；Downloader（下载器）负责下载引擎发送的所有Request请求，将获取到的Response交还给引擎，再由引擎交给Spider进行后续处理；Spider（爬虫）负责处理所有Responses，分析并从中提取数据，将需要跟进的URL提交给引擎再次进入Scheduler；Item Pipline（管道）负责处理Spider中获取的Item，并进行后续处理，如数据分析、过滤、存储等；Middlewares（中间件）是一个可以自定义扩展功能的组件。

在编写爬虫时，Items.py中设置了爬虫所要爬取的网页字段：有新闻标题、新闻主题内容；settings.py中设置了名为“valuelist”的爬虫对应的下载延迟、管道信息；pipelines.py中设置了如何存储Spider中获取的item；spiders文件夹下的valuelist.py是一个爬虫脚本，设置了爬虫名称、进行爬虫的网站域名、添加了浏览器header和用户认证的cookie信息，并将所有的Responses进行处理，利用xpath解析并提取网页信息。

### 3.6.1.1 实验数据

经过数据爬虫、数据清洗、人工数据标注后最后得到的实验数据共有55000条数据，数据如图3.11所示。

|  |
| --- |
| {"entities": [{"word": "中诚信国际", "start": 1, "end": 6, "type": "C"}, {"word": "重庆能源", "start": 11, "end": 15, "type": "C"}], "text": "据中诚信国际发布的关于重庆能源2016年度跟踪评级报告显示，该公司2013年、2014年、2015年和2016年一季度的营收分别为344.90亿元、341.12亿元、360.08亿元、75.74亿元、净利润分别为0.67亿元、0.06亿元、-1.77亿元和"}  {"entities": [{"word": "重庆能源", "start": 23, "end": 27, "type": "C"}, {"word": "煤炭", "start": 35, "end": 37, "type": "IND"}], "text": "《每日经济新闻》记者注意到，上述评级报告在阐述重庆能源的关注点时提及“煤炭板块盈利能力较弱，拖累公司经营性业务利润”"}  {"entities": [{"word": "中国", "start": 3, "end": 5, "type": "L"}], "text": "然而，中国公务机运营商自有机队规模不断扩大的同时，利用率却严重不足"}  {"entities": [{"word": "张洪超", "start": 1, "end": 4, "type": "P"}, {"word": "GLOBAL WINGS", "start": 34, "end": 46, "type": "C"}], "text": "\"张洪超在接受《每日经济新闻》等媒体采访时表示，正是基于这两大痛点，GLOBAL WINGS成立了"} |

图3.11 实验数据示例

再通过数据预处理，将数据转换成每个字和对应的标签为一行的形式，句与句之间用空行隔开，数据预处理实现如图3.12所示。

|  |
| --- |
| import json  class entityJsonToTrain(object):      def \_\_init\_\_(self):          self.vocab = [ ',' , '。' , '，']      def read(self, path):          jsonList = []          with open(path,encoding='utf-8') as T:              for i  in T.readlines():                  i = i.strip('\n')                  jsonList.append(json.loads(i))          return jsonList      def processJson(self, jsonList):          result = []          n = 0          for i in jsonList:              n += 1              type = ['O' for j in range(0,len(i['text']))]              for j in i['entities']:                  type[j['start']] = 'B-' + j['type']                  for t in range(j['start']+1,j['end']):                      type[t] = 'I-' + j['type']              for p,q in enumerate(i['text']):                  temp = “”                  if q not in self.vocab:                      temp = q + " " + type[p]                  result.append(temp)              result.append('')          return result      def write(self, path, result):          with open(path,'w',encoding='utf-8') as T:              for i in result:                  T.write(i+"\n")      def test(self, jsonList):          result = []          for i in jsonList:              print(i['entities'])              for j in i['entities']:                  print(1)  A = entityJsonToTrain()  jsonList = A.read('xxx.txt')  result = A.processJson(jsonList)  A.write('xxx\_train.txt',result) |

图3.12 命名实体识别任务数据预处理代码

最后得到的输入数据的形式如图3.13所示。

|  |
| --- |
| 中 B-L  国 I-L  公 O  务 O  机 O  运 O  营 O  商 O  自 O  有 O  机 O  队 O  规 O  模 O  不 O  断 O  扩 O  大 O |

图3.13 输入数据示例

按照样本均衡的角度来看应该所有类别实体的占比都为1，由于爬取的文本中有些类别的实体过少，所以数量不足，最后55000条数据中六种类别实体所占比例大致为C:P:INS:IND:PRO:L为3:3:1.5:1:1:1。将训练集、验证集、测试集按3:1:1的比例结合不同类别的实体占比进行切分。

### 3.6.2 实验评价指标

本文的实验中采用准确率（Precision）、召回率（Recall）和F1值作为模型的评价指标。其中准确率是指模型预测正确的实体个数占模型预测的所有实体个数的比例，而召回率是指模型预测正确的实体个数占样本中含有的实体总数的比例，FI值则是准确率和召回率的调和平均数：公式3.11。

### 3.6.3 模型训练的实现

针对输入数据，先做数据预处理，将输入文本转化为词表和标签列表并存入maps.pkl中供后续操作进行读取，具体实现如图3.14所示。

|  |
| --- |
| # create maps if not exist  if not os.path.isfile(FLAGS.map\_file):      # create dictionary for word      if FLAGS.pre\_emb:          dico\_chars\_train = char\_mapping(train\_sentences, FLAGS.lower)[0]          dico\_chars, char\_to\_id, id\_to\_char = augment\_with\_pretrained(              dico\_chars\_train.copy(),              FLAGS.emb\_file,              list(itertools.chain.from\_iterable(                  [[w[0] for w in s] for s in test\_sentences])              )          )      else:          \_c, char\_to\_id, id\_to\_char = char\_mapping(train\_sentences, FLAGS.lower)      # Create a dictionary and a mapping for tags      \_t, tag\_to\_id, id\_to\_tag = tag\_mapping(train\_sentences)      with open(FLAGS.map\_file, "wb") as f:          pickle.dump([char\_to\_id, id\_to\_char, tag\_to\_id, id\_to\_tag], f)  else:      with open(FLAGS.map\_file, "rb") as f:          char\_to\_id, id\_to\_char, tag\_to\_id, id\_to\_tag = pickle.load(f) |

图3.14 词表和标签列表的存储

再把数据按batch\_size的大小进行切分，这里用到了自定义的BatchManager类，如图3.15所示。

|  |
| --- |
| class BatchManager(object):      def \_\_init\_\_(self, data,  batch\_size):          self.batch\_data = self.sort\_and\_pad(data, batch\_size)          self.len\_data = len(self.batch\_data)      def sort\_and\_pad(self, data, batch\_size):          num\_batch = int(math.ceil(len(data) / batch\_size))          sorted\_data = sorted(data, key=lambda x: len(x[0]))          batch\_data = []          for i in range(num\_batch):              a = int(i\*batch\_size)              b = int((i+1)\*batch\_size)              batch\_data.append(self.pad\_data(sorted\_data[a:b]))          return batch\_data      @staticmethod      def pad\_data(data):          strings = []          chars = []          segs = []          targets = []          max\_length = max([len(sentence[0]) for sentence in data])          for line in data:              string, char, seg, target = line              padding = [0] \* (max\_length - len(string))              strings.append(string + padding)              chars.append(char + padding)              segs.append(seg + padding)              targets.append(target + padding)          return [strings, chars, segs, targets]      def iter\_batch(self, shuffle=False):          if shuffle:              random.shuffle(self.batch\_data)          for idx in range(self.len\_data):              yield self.batch\_data[idx] |

图3.15 BatchManager类的实现代码

数据预处理完成后，便可以开始进行模型训练：建立模型、设置步长、训练模型、打印日志，如图3.16所示。

|  |
| --- |
| tf\_config = tf.ConfigProto()  tf\_config.gpu\_options.allow\_growth = True  steps\_per\_epoch = train\_manager.len\_data  with tf.Session(config=tf\_config) as sess:      model = create\_model(sess, Model, FLAGS.ckpt\_path, load\_word2vec, config, id\_to\_char, logger)      logger.info("start training")      loss = []      for i in range(100):          for batch in train\_manager.iter\_batch(shuffle=True):              step, batch\_loss = model.run\_step(sess, True, batch)              loss.append(batch\_loss)              if step % FLAGS.steps\_check == 0:                  iteration = step // steps\_per\_epoch + 1                  logger.info("iteration:{} step:{}/{}, "                              "NER loss:{:>9.6f}".format(                      iteration, step%steps\_per\_epoch, steps\_per\_epoch, np.mean(loss)))                  loss = []          best = evaluate(sess, model, "dev", dev\_manager, id\_to\_tag, logger)          if best:              save\_model(sess, model, FLAGS.ckpt\_path, logger)          evaluate(sess, model, "test", test\_manager, id\_to\_tag, logger) |

图3.16 模型训练的具体实现

模型结构的具体实现如图3.17所示。

|  |
| --- |
| # embeddings for chinese character and segmentation representation  embedding = self.embedding\_layer(self.char\_inputs, self.seg\_inputs, config)  # apply dropout before feed to lstm layer  lstm\_inputs = tf.nn.dropout(embedding, self.dropout)  # bi-directional lstm layer  lstm\_outputs = self.biLSTM\_layer(lstm\_inputs, self.lstm\_dim, self.lengths)  # logits for tags  self.logits = self.project\_layer(lstm\_outputs)  # loss of the model  self.loss = self.loss\_layer(self.logits, self.lengths)  with tf.variable\_scope("optimizer"):      optimizer = self.config["optimizer"]      if optimizer == "sgd":          self.opt = tf.train.GradientDescentOptimizer(self.lr)      elif optimizer == "adam":          self.opt = tf.train.AdamOptimizer(self.lr)      elif optimizer == "adgrad":          self.opt = tf.train.AdagradOptimizer(self.lr)      else:          raise KeyError      # apply grad clip to avoid gradient explosion      grads\_vars = self.opt.compute\_gradients(self.loss)      capped\_grads\_vars = [[tf.clip\_by\_value(g, -self.config["clip"], self.config["clip"]), v]                           for g, v in grads\_vars]      self.train\_op = self.opt.apply\_gradients(capped\_grads\_vars, self.global\_step)  # saver of the model  self.saver = tf.train.Saver(tf.global\_variables(), max\_to\_keep=5) |

图3.17 模型结构的具体实现

在模型结构的具体实现中，self.embedding\_layer、self.biLSTM\_layer、self.project\_layer、self.loss\_layer分别对应了输入层、网络层、全连接层和CRF层，它们的实现如图3.18所示。

|  |
| --- |
| def embedding\_layer(self, char\_inputs, seg\_inputs, config, name=None):  embedding = []      with tf.variable\_scope("char\_embedding" if not name else name), tf.device('/cpu:0'):          self.char\_lookup = tf.get\_variable(                  name="char\_embedding",                  shape=[self.num\_chars, self.char\_dim],                  initializer=self.initializer)          embedding.append(tf.nn.embedding\_lookup(self.char\_lookup, char\_inputs))          if config["seg\_dim"]:              with tf.variable\_scope("seg\_embedding"), tf.device('/cpu:0'):                  self.seg\_lookup = tf.get\_variable(                      name="seg\_embedding",                      shape=[self.num\_segs, self.seg\_dim],                      initializer=self.initializer)                  embedding.append(tf.nn.embedding\_lookup(self.seg\_lookup, seg\_inputs))          embed = tf.concat(embedding, axis=-1)      return embed  def biLSTM\_layer(self, lstm\_inputs, lstm\_dim, lengths, name=None):  with tf.variable\_scope("char\_BiLSTM" if not name else name):          lstm\_cell = {}          for direction in ["forward", "backward"]:              with tf.variable\_scope(direction):                  lstm\_cell[direction] = rnn.CoupledInputForgetGateLSTMCell(                      lstm\_dim,                      use\_peepholes=True,                      initializer=self.initializer,                      state\_is\_tuple=True)          outputs, final\_states = tf.nn.bidirectional\_dynamic\_rnn(              lstm\_cell["forward"],              lstm\_cell["backward"],              lstm\_inputs,              dtype=tf.float32,              sequence\_length=lengths)      return tf.concat(outputs, axis=2)  def project\_layer(self, lstm\_outputs, name=None):  with tf.variable\_scope("project"  if not name else name):          with tf.variable\_scope("hidden"):              W = tf.get\_variable("W", shape=[self.lstm\_dim\*2, self.lstm\_dim],                                  dtype=tf.float32, initializer=self.initializer)              b = tf.get\_variable("b", shape=[self.lstm\_dim], dtype=tf.float32,                                  initializer=tf.zeros\_initializer())              output = tf.reshape(lstm\_outputs, shape=[-1, self.lstm\_dim\*2], name="output")              hidden = tf.tanh(tf.nn.xw\_plus\_b(output, W, b),name="hidden")          # project to score of tags          with tf.variable\_scope("logits"):              W = tf.get\_variable("W", shape=[self.lstm\_dim, self.num\_tags],                                  dtype=tf.float32, initializer=self.initializer)              b = tf.get\_variable("b", shape=[self.num\_tags], dtype=tf.float32,                                  initializer=tf.zeros\_initializer())              pred = tf.nn.xw\_plus\_b(hidden, W, b, name="pred")          return tf.reshape(pred, [-1, self.num\_steps, self.num\_tags], name="scores")  def loss\_layer(self, project\_logits, lengths, name=None):      with tf.variable\_scope("crf\_loss"  if not name else name):          small = -1000.0          # pad logits for crf loss          start\_logits = tf.concat(              [small \* tf.ones(shape=[self.batch\_size, 1, self.num\_tags]),               tf.zeros(shape=[self.batch\_size, 1, 1])],               axis=-1)          pad\_logits = tf.cast(small \* tf.ones([self.batch\_size, self.num\_steps, 1]), tf.float32)          logits = tf.concat([project\_logits, pad\_logits], axis=-1)          logits = tf.concat([start\_logits, logits], axis=1)          targets = tf.concat(              [tf.cast(self.num\_tags\*tf.ones([self.batch\_size, 1]),              tf.int32), self.targets],              axis=-1)          self.trans = tf.get\_variable(              "transitions",              shape=[self.num\_tags + 1, self.num\_tags + 1],              initializer=self.initializer)          log\_likelihood, \_ = crf\_log\_likelihood(              inputs=logits,              tag\_indices=targets,              transition\_params=self.trans,              sequence\_lengths=lengths+1)          return tf.reduce\_mean(-log\_likelihood) |

图3.18 模型中各个层的具体实现

### 3.6.4 模型参数设置

模型使用金融领域文本语料训练的字向量，向量维度为100维，其余超参数设置如表3.2所示。

|  |  |
| --- | --- |
| 参数名称 | 参数大小 |
| 字向量维度 | 100 |
| 词汇特征维度 | 20 |
| LSTM隐藏层大小 | 100 |
| batch\_size大小 | 64 |
| 学习率 | 0.001 |
| Dropout比例 | 0.5 |
| 优化函数 | Adam |

表3.2 命名实体识别模型参数设置

经过测试发现，LSTM隐藏层的大小、学习率大小对模型最终的测试结果影响相对较大，表3.3为不同数值下的LSTM隐藏层(h)和学习率(lr)对应的模型的测试结果。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 参 数  设 置 | C（%） | | | P（%） | | | INS（%） | | |
| P | R | F1 | P | R | F1 | P | R | F1 |
| h=50,  lr=0.001 | 93.49 | 94.85 | 94.16 | 95.61 | 96.08 | 95.84 | 75.25 | 73.79 | 74.51 |
| h=50,  lr=0.005 | 91.68 | 88.26 | 89.94 | 90.33 | 87.65 | 88.97 | 70.28 | 69.56 | 69.92 |
| h=50,  lr=0.01 | 89.37 | 86.25 | 87.78 | 90.45 | 89.86 | 90.15 | 70.83 | 68.67 | 69.73 |
| h=100,  lr=0.001 | 90.62 | 86.72 | 88,63 | 89.81 | 85.42 | 87.56 | 73.32 | 70.51 | 71.89 |
| h=100,  lr=0.005 | 88.86 | 83.57 | 86.13 | 86.28 | 83.36 | 84.79 | 69.98 | 67.57 | 68.75 |
| h=100,  lr=0.01 | 87.35 | 82.19 | 84.69 | 86.45 | 83.86 | 85.14 | 68.72 | 65.43 | 67.03 |

表3.3 不同超参数下模型的测试结果

针对2个参数的不同取值进行排列组合后，选择模型中实体数量相对平衡的三个类别“公司C”、“人名P”、“组织机构INS”进行测试。对比测试结果可以发现，当LSTM隐藏层大小为100，学习率为0.001时，训练出来的模型测试效果最好。

### 3.6.5 不同模型的性能对比

针对本章需要解决的命名实体识别任务，为了对本文选择的模型进行更全面、更准确的性能评估，本节将列出在相同的输入和语料环境下，CRF、LSTM、BiLSTM等模型解决命名实体识别任务时的性能对比，如表3.4所示。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 模 型  名 称 | C（%） | | | P（%） | | | INS（%） | | |
| P | R | F1 | P | R | F1 | P | R | F1 |
| BiLSTM+CRF | 93.49 | 94.85 | 94.16 | 95.61 | 96.08 | 95.84 | 75.25 | 73.79 | 74.51 |
| CRF | 83.26 | 82.35 | 82.80 | 84.19 | 83.24 | 83.71 | 66.25 | 63.32 | 64.75 |
| LSTM | 84.12 | 82.62 | 83.36 | 85.32 | 83.37 | 84.33 | 66.28 | 64.58 | 65.42 |
| BiLSTM | 87.18 | 85.27 | 86.21 | 88.93 | 86.54 | 87.72 | 69.93 | 67.72 | 68.81 |

表3.4 不同模型在相同输入下的性能对比

CRF的分数不高的原因可能是由于CRF无法学习到较长文本中的关联信息；LSTM只学习到了当前词之前的文本信息，没有考虑到下文对当前词的影响； BiLSTM没有CRF层来学习约束条件，所以整体效果不如本文所选的BiLSTM+CRF模型。

## 3.7 本章小结

本章主要介绍了本文实现信息抽取任务中命名实体识别子任务的模型设计细节和详细的实现过程，并把实验结果与一些通用的方法做了对比分析，并分析了各个参数对模型最终效果造成的影响以及模型最后选取的参数。

# 

# 第四章 基于深度学习的关系抽取模型

在总结和归纳了各种已发表的命名实体识别技术相关的论文后，本章基于已有的相关技术与模型，选择了残差网络+卷积神经网络来解决关系抽取问题。本章讲详细介绍模型的每一部分具体的网络结构和各种参数的对比实验。

## 4.1 模型的整体架构

深度残差网络作为一种比较新的深度神经网络模型常被用于图像领域相关任务，因为它在层数较深时的表现十分优异。本文使用了一种新的CNN模型结合残差网络的方法用于关系抽取问题的研究，不同于以往残差网络的结构较深的特点，本文采用的卷积神经网络结构只有9层，并且得到了不错的分类效果。实验中还发现使用身份映射（identity mapping）能够显著提升远程监督关系抽取问题的性能。

ResCNN主要的结构分为输入层、卷积层、残差卷积块、最大池化层和输出层几部分。模型的结构图如图4.1所示。



图4.1 ResCNN模型结构示意图

输入层包括词向量层和位置向量层；卷积层是对输入层输出的文本特征向量做卷积操作来提取特征；残差卷积块利用残差学习结合卷积神经网络对卷积层的输出进行进一步处理；最大池化层对残差卷积块的输出进行最大池化操作并取其中的最大值；最后把这些特征传入Softmax层进行计算得到最终的预测结果。本章将会对每层的网络结构和设计细节做详细的介绍。

## 4.2 输入层

### 模型的输入层是将文本输入的单词序列进行处理转化成单词的特征向量序列汇总后传输进后续的卷积层进行卷积处理，而单词的特征向量是由输入文本的词向量和相对位置向量拼接组成的。

假设xi是输入文本的第i个词，(e1, e2)是文本中的实体对，将xi在预训练好的字向量表中对应的字向量设为WFi，相对位置向量设为PFi，最终把WFi和PFi拼接起来变成词xi的总特征向量vi=[WFi, PFi]。

### 4.2.1 字向量

由于本章关系抽取的任务是接着上一章的命名实体识别之后继续对金融领域的文本进行实体间关系抽取，所以本章所采用的关系抽取模型使用的字向量也是基于之前的金融领域上市公司新闻、金融公告、行业信息等中文语料，利用Gensim库中的word2vec方法进行预训练得到的字向量模型。

每个xi对应的总特征向量vi都是一个实值向量，所有的向量组成了一个特征向量矩阵公式4.1，其中V是一个输入文本的所有单词构成的词表。

### 4.2.2 相对位置向量

在对输入文本做特征提取时，通常的做法是对输入文本中的单词做一个词向量转换来学习到文本中的语义信息。然而在关系抽取任务中，要想最终获得两个实体间的关系，不仅可以从词汇信息和语义信息的角度出发，还可以从当前单词与实体之间的相对位置中获取一些有用的信息[Zeng et al., 2014]。

相对位置向量PFi结合了当前字符xi分别与两个实体e1和e2的相对距离，比如在句子“x0x1e1x2x3x4e2x5.”中，x2与e1和e2的相对距离分别为1和-3。之后通过一个随机初始化的相对位置矩阵公式4.2将相对位置距离转化成实值向量。如果当前字符距离命名实体的距离过远，那么可能会被认为与实体间的关系没有关联而在最后分类的时候被忽略。

最后把每个字的字向量和相对位置向量做拼接，再把句子中的每个字的总特征向量拼接成一个向量：公式4.3

## 4.3 卷积层

卷积层的作用是把输入层输出的总特征向量进行卷积操作后学习到输入文本的潜在特征。

用vi:i+j来表示特征向量vi,vi+1,...,vi+j拼接得到的向量。卷积操作就是用一个大小为h的窗口在卷积层的输入特征向量上进行“滑动“，对窗口中的所有字符对应的特征向量元素做矩阵计算，最终得到新的输出作为特征C。特征ci是由一个滑动窗口中的字符对应的特征向量vi:i+h-1通过公式公式4.4计算得到的。其中b为偏置项，f是一个非线性函数。卷积神经网络的窗口大小也就是卷积核的大小，卷积核不断滑动直到遍历整个输入矩阵，将输入通过计算加权和的方式对输入数据做处理。而输入数据是否能进入窗口扫描的区域直接决定了数据经过卷积核的计算后的输出结果，所以卷积核的尺寸直接决定了新的特征C汇聚了多少输入特征。如果是深度神经网络（DNN），那么使用标准的全连接层产生的参数个数就会大大增多，每个输出都是所有输入数据的加权求和得到的结果。但卷积操作的话只允许用filter\_size\*filter\_size个参数来实现这个变换，每个输出特性并不需要“利用”每个输入特征，而只“利用”来自大致相同位置（周边位置）的输入特征。

最终通过滑动窗口使所有的特征向量经过卷积操作得到了特征c=[c1, c2, ..., cn-h+1]作为卷积层的输出。

## 4.4 残差卷积块

残差学习将低层的信息直接与高层的信息进行联接来消除深层网络中的梯度消失问题，本文中的残差卷积块就是利用了快捷连接（shortcut connections）对卷积层的输出进行处理。

每个残差卷积块是一个由2层卷积神经网络组成的序列，所有卷积神经网络中的卷积核大小都为h，每层卷积神经网络后还接了一个ReLU激活函数。ReLU激活函数常在训练神经网络时作为中间隐藏层的一种激活函数，其表达式为公式4.9。相比于最常用的Sigmoid激活函数，由于优化参数时一般会用到误差反向传播算法，要对激活函数进行求导，而Sigmoid激活函数的导数表达式为公式4.10，对应的图形如图4.2所示。



图4.2 Sigmoid激活函数

由图可以看出Sigmoid激活函数的导数从0开始后很快就会趋近于0，容易产生“梯度消失”现象，而ReLU激活函数就不会存在这样的情况，它的导数表达式为公式4.11，对应的图形如图4.3所示。



图4.3 ReLU激活函数

使用ReLU作为激活函数的好处有：

1. 单侧抑制。即使所有的负值都变为0，而正值的数值不变。正因为ReLU的单侧抑制才使得神经网络中的神经元具有了稀疏激活性，也就是说当模型的层数增加了N倍后，ReLU神经元的激活率将降低2的N次方倍。
2. 对于线性函数而言，ReLU的表达能力更强，尤其在网络很深的情况下。
3. 对于非线性函数而言，ReLU由于非负区间的梯度为常数，所以不存在梯度消失的问题，而且模型的收敛速度维持在一个比较稳定的状态。

设残差卷积块中的2层卷积神经网络的卷积过滤器分别为w1和w2，公式4.5。第一个卷积过滤器（filter）：公式4.6。第二个卷积过滤器：公式4.7.

其中b1和b2都是偏置项。最后再通过残差学习操作：公式4.8。

本文实现的模型中使用了9个这样的残差卷积块，进行了一系列的卷积+残差连接的操作后最后得到的特征矩阵仍记为C。

## 4.5 最大池化层

对残差卷积块得到的特征c采用最大池化操作，得到每个特征c对应的最大值max{c}。由于每个过滤器都可以提取一个特征，所以共提取到m个特征（本文中m为32）。

## 4.6 输出层

模型的输出层是一个全连接的Softmax层，输入为最大池化层中提取出来的m个特征，输出的是通过使用Dropout进行全连接得到的关系的概率分布。

## 4.7 实验

本节实验将从实验数据、实验评价标准、模型训练、模型参数设置、不同模型的性能对比等方面详细介绍整个实验的实现过程及结果分析。

## 4.7.1 实验数据

本文中信息抽取的两个子任务命名实体识别和关系抽取所用的实验数据是同一批数据，都是根据3.6.1中的爬虫得到的金融领域上市公司相关的文本数据。由于关系抽取任务的人工标注相对于命名实体识别任务的人工标注而言更加困难和耗时，所以本文采用了远程监督的方法来扩充人工标注的关系抽取语料。在经过数据清洗、人工标注和使用远程监督后共有154340条可供训练的数据样本，经过人工筛选，共选出33种关系作为关系抽取模型的关系标签，表4.1为关系标签列表。

|  |
| --- |
| unknown 0  实际控制人 1  主营业务 2  子公司 3  合作伙伴 4  一致行动人 5  收购 6  调查 7  处罚 8  法人 9  行业 10  关联关系 11  高管 12  供应商 13  监事 14  总统 15  重组 16  下属公司 17  附属公司 18  夫妻 19  执行事务合伙人 20  股东 21  股权转让 22  质押 23  设立 24  创始人 25  董事 26  成立 27  地址 28  曾用名 29  CEO 30  职工 31  亲属 32  并购 33 |

表4.1 关系标签列表

其中没有关系的两个实体间的关系设为unknown。最后得到的输入数据的格式如图4.4所示。

|  |
| --- |
| 金力泰 吴国政 股东 金力泰公告，12月13日，公司控股股东、实际控制人吴国政先生协议转让给柯桥领英的公司47034000股无限售条件流通股 （占公司总股本10.00％）已完成了过户登记手续  金力泰 吴国政 实际控制人 金力泰公告，12月13日，公司控股股东、实际控制人吴国政先生协议转让给柯桥领英的公司47034000股无限售条件流通股 （占公司总股本10.00％）已完成了过户登记手续  吴国政 柯桥领英 股权转让 金力泰公告，12月13日，公司控股股东、实际控制人吴国政先生协议转让给柯桥领英的公司47034000股无限售条件流通股 （占公司总股本10.00％）已完成了过户登记手续  艾迪西 申通快递 收购 艾迪西收购申通快递100%股权过户完成；2、科林环保拟以自筹资金5000万元设立全资子公司；3、勤上光电全资子公司拟筹划股权收购事宜，13日起停牌……科林环保午间公告称，公司为满足业务发展的需要，拟向控股股东重庆东诚瑞业投资有限公司（以下简称“东诚瑞业”）借款，借款总额不超过2500万元，借款利率为4.75%，借款期限为12个月，在借款到期时一次性还本付息，若提前归还借款时，应按实际使用时间付息。 |

图4.4 关系抽取模型的输入文本示例

将154340条数据按训练集、验证集、测试集比例为3:1:1进行切分后，训练集、验证集、测试集的样本数量分别为92604、30868、30868条数据。

## 4.7.2 实验评价标准

本文的实验中采用准确率（Precision）、召回率（Recall）和F1值作为模型的评价指标。其中准确率是指模型预测正确的实体个数占模型预测的所有实体个数的比例，而召回率是指模型预测正确的实体个数占样本中含有的实体总数的比例，FI值则是准确率和召回率的调和平均数：公式3.11。

## 4.7.3 模型训练的实现

针对模型的输入文本，先定义一个DataManager类，用来处理输入数据，如图4.5所示。

|  |
| --- |
| class DataManager:      def \_\_init\_\_(self, sequence\_length):          self.word\_vector\_dim = 0          self.sequence\_length = sequence\_length          self.word2index = {}          self.index2vector = []          self.relations = {}          self.bags\_test = {}          self.training\_data = []          self.load\_word2vec()          self.load\_relations()      def load\_word2vec(self):          word\_vector = list(open("../../origin\_data/vec.txt", 'r', encoding='utf-8').readlines())          del word\_vector[0]          word\_vector = [s.split() for s in word\_vector]          self.word\_vector\_dim = len(word\_vector[0]) - 1          self.word2index["UNK"] = 0          self.index2vector.append(np.zeros(self.word\_vector\_dim))          index = 1          for vec in word\_vector:              a = np.zeros(self.word\_vector\_dim)              for i in range(self.word\_vector\_dim):                  a[i] = float(vec[i+1])              self.word2index[vec[0]] = index              self.index2vector.append(a)              index += 1          print("word total=\t", len(self.index2vector))          print("word dimension=\t", self.word\_vector\_dim)      def load\_relations(self):          relation\_data = list(open("../../origin\_data/relation2id\_company.txt", 'r', encoding='utf-8').readlines())          relation\_data = [s.split() for s in relation\_data]          for relation in relation\_data:              r = Relation(relation[0], int(relation[1]))              self.relations[relation[0]] = r          for r in self.relations:              self.relations[r].generate\_vector(len(self.relations))          print("relationTotal:"+str(len(self.relations)))      def load\_training\_data(self, filename="../../origin\_data/train\_total.txt"):          print("start loading training data.")          print("===============")          training\_data = list(open(filename, 'r', encoding='utf-8').readlines())          training\_data = [s.split(' ') for s in training\_data]          for data in training\_data:              en1 = data[0]              en2 = data[1]              if data[2] not in self.relations:                  relation = self.relations["unknown"]              else:                  relation = self.relations[data[2]]              # sent = list(jieba.cut(data[3], cut\_all=True))              sent = self.cut\_word(data[3])              s = Sentence(en1, en2, relation, sent)              self.training\_data.append(s)          return self.training\_data      def load\_testing\_data(self):          print("start loading test data.")          print("===========")          testing\_data = list(open("../../origin\_data/test\_total.txt", 'r', encoding='utf-8').readlines())          testing\_data = [s.split() for s in testing\_data]          for data in testing\_data:              en1 = data[0]              en2 = data[1]              if data[2] not in self.relations:                  relation = self.relations["unknown"]              else:                  relation = self.relations[data[2]]              # sent = list(jieba.cut(data[3], cut\_all=True))              sent = self.cut\_word(data[3])              s = Sentence(en1, en2, relation, sent)              if data[0]+"\t"+data[1] not in self.bags\_test:                  self.bags\_test[en1+" "+en2] = [s]              else:                  self.bags\_test[en1+" "+en2].append(s)          return self.bags\_test |

图4.5 DataManager类的具体实现

在读取数据后就开始进入模型训练：建立模型、设置步长、选择优化函数、计算梯度等，如图4.6所示。

|  |
| --- |
| print("Start Training")  with tf.Graph().as\_default():      session\_conf = tf.ConfigProto(        allow\_soft\_placement=FLAGS.allow\_soft\_placement,        log\_device\_placement=FLAGS.log\_device\_placement)      sess = tf.Session(config=session\_conf)      with sess.as\_default():          cnn = RECnn(              FLAGS.sequence\_length,              len(datamanager.relations),              FLAGS.embedding\_dim,              5,              list(map(int, FLAGS.filter\_sizes.split(","))),              FLAGS.num\_filters,              FLAGS.l2\_reg\_lambda)          # Define Training procedure          global\_step = tf.Variable(0, name="global\_step", trainable=False)          optimizer = tf.train.AdamOptimizer(1e-3)          grads\_and\_vars = optimizer.compute\_gradients(cnn.loss)          train\_op = optimizer.apply\_gradients(grads\_and\_vars, global\_step=global\_step)          saver = tf.train.Saver(tf.global\_variables())          print("Initialize variables.")          sess.run(tf.global\_variables\_initializer())          def train\_step(x\_batch, y\_batch, p1\_batch, p2\_batch):              feed\_dict = {                cnn.input\_x: x\_batch,                cnn.input\_y: y\_batch,                cnn.input\_p1: p1\_batch,                cnn.input\_p2: p2\_batch,                cnn.dropout\_keep\_prob: FLAGS.dropout\_keep\_prob              }              \_, step, loss, accuracy = sess.run(                  [train\_op, global\_step, cnn.loss, cnn.accuracy],                  feed\_dict)              # time\_str = datetime.datetime.now().isoformat()              # print("{}: step {}, loss {:g}, acc {:g}".format(time\_str, step, loss, accuracy))              return loss |

图4.6 模型训练的具体实现

在模型训练中，RECnn()就是本文所述的关系抽取模型的结构，具体实现如图4.7所示。

|  |
| --- |
| class RECnn(object):      def \_\_init\_\_(        self, sequence\_length, num\_classes,        embedding\_size, position\_size, filter\_sizes, num\_filters, l2\_reg\_lambda=0.0):          with tf.device('/cpu:0'):              self.input\_x = tf.placeholder(tf.float32, [None, sequence\_length, embedding\_size], name="input\_x")              self.input\_p1 = tf.placeholder(tf.int32, [None, sequence\_length], name="input\_p1")              self.input\_p2 = tf.placeholder(tf.int32, [None, sequence\_length], name="input\_p2")              self.input\_y = tf.placeholder(tf.float32, [None, num\_classes], name="input\_y")              self.dropout\_keep\_prob = tf.placeholder(tf.float32, name="dropout\_keep\_prob")              self.l2\_loss = tf.constant(0.0)              with tf.name\_scope("position-embedding"):                  W = tf.Variable(tf.random\_uniform([141, 5],                       minval=-math.sqrt(6/(3\*position\_size+3\*embedding\_size)),                       maxval=math.sqrt(6/(3\*position\_size+3\*embedding\_size))),                       name="W")                  self.input\_x\_p1 = tf.nn.embedding\_lookup(W, self.input\_p1)                  self.input\_x\_p2 = tf.nn.embedding\_lookup(W, self.input\_p2)                  self.x = tf.concat([self.input\_x, self.input\_x\_p1, self.input\_x\_p2], 2)                  # 添加一维，作为output\_channel，shape为[batch\_size,sequence\_length,embedding\_size+2\*position\_size,1]                  self.embedded\_chars\_expanded = tf.expand\_dims(self.x, -1)              pooled\_outputs = []              for i, filter\_size in enumerate(filter\_sizes):                  with tf.name\_scope("conv-maxpool-%s" % filter\_size):                      # Convolution Layer                      print(filter\_size)                      filter\_shape = [filter\_size, embedding\_size+2\*position\_size, 1, num\_filters]                      W = tf.Variable(tf.truncated\_normal(filter\_shape, stddev=0.1), name="W")                      b = tf.Variable(tf.constant(0.1, shape=[num\_filters]), name="b")                      conv = tf.nn.conv2d(                          self.embedded\_chars\_expanded,                          W,                          strides=[1, 1, 1, 1],                          padding="VALID",                          name="conv")                      # Apply non-linearity, shape of h is [batch\_size, sequence\_length-2, 1, num\_filters]                      h = tf.nn.relu(tf.nn.bias\_add(conv, b), name="relu")                      self.l2\_loss += tf.nn.l2\_loss(W)                      self.l2\_loss += tf.nn.l2\_loss(b)                      # nine convolution layers                      for i in range(3):                          h2 = self.Cnnblock(num\_filters, h, i)                          h = h2+h                      pooled = tf.nn.max\_pool(                          h,                          ksize=[1, sequence\_length - filter\_size + 1, 1, 1],                          strides=[1, 1, 1, 1],                          padding='VALID',                          name="pool")                      pooled\_outputs.append(pooled)              num\_filters\_total = num\_filters \* len(filter\_sizes)              self.h\_pool = tf.concat(pooled\_outputs, 3)              self.h\_pool\_flat = tf.reshape(self.h\_pool, [-1, num\_filters\_total], name="hidden\_feature")              with tf.name\_scope("dropout"):                  self.h\_drop = tf.nn.dropout(self.h\_pool\_flat, self.dropout\_keep\_prob)              with tf.name\_scope("MLP"):                  W0 = tf.Variable(tf.truncated\_normal([num\_filters\_total, num\_filters\_total], stddev=0.1), name="W0")                  b0 = tf.Variable(tf.constant(0.1, shape=[num\_filters\_total]), name="b0")                  h0 = tf.nn.relu(tf.nn.xw\_plus\_b(self.h\_drop, W0, b0))                  self.l2\_loss += tf.nn.l2\_loss(W0)                  self.l2\_loss += tf.nn.l2\_loss(b0)                  W1 = tf.Variable(tf.truncated\_normal([num\_filters\_total, num\_filters\_total], stddev=0.1), name="W1")                  b1 = tf.Variable(tf.constant(0.1, shape=[num\_filters\_total]), name="b1")                  self.h1 = tf.nn.relu(tf.nn.xw\_plus\_b(h0, W1, b1))                  self.l2\_loss += tf.nn.l2\_loss(W1)                  self.l2\_loss += tf.nn.l2\_loss(b1)              with tf.name\_scope("output"):                  W = tf.get\_variable(                      "W",                      shape=[num\_filters\_total, num\_classes],                      initializer=tf.contrib.layers.xavier\_initializer())                  b = tf.Variable(tf.constant(0.1, shape=[num\_classes]), name="b")                  self.l2\_loss += tf.nn.l2\_loss(W)                  self.l2\_loss += tf.nn.l2\_loss(b)                  self.scores = tf.nn.xw\_plus\_b(self.h1, W, b, name="scores")                  self.predictions = tf.argmax(self.scores, 1, name="predictions")              with tf.name\_scope("loss"):                  losses = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=self.scores, labels=self.input\_y)                  self.loss = tf.reduce\_mean(losses) + l2\_reg\_lambda \* self.l2\_loss              with tf.name\_scope("accuracy"):                  correct\_predictions = tf.equal(self.predictions, tf.argmax(self.input\_y, 1))                  self.accuracy = tf.reduce\_mean(tf.cast(correct\_predictions, "float"), name="accuracy")      def Cnnblock(self, num\_filters, h, i):          W1 = tf.get\_variable(              "W1\_"+str(i),              shape=[3, 1, num\_filters, num\_filters],              initializer=tf.contrib.layers.xavier\_initializer\_conv2d())          b1 = tf.Variable(tf.constant(0.1, shape=[num\_filters]), name="b1\_"+str(i))          conv1 = tf.nn.conv2d(              h,              W1,              strides=[1, 1, 1, 1],              padding="SAME")          h1 = tf.nn.relu(tf.nn.bias\_add(conv1, b1), name="relu1")          self.l2\_loss += tf.nn.l2\_loss(W1)          self.l2\_loss += tf.nn.l2\_loss(b1)          # W2 = tf.Variable(tf.truncated\_normal([3, 1, num\_filters, num\_filters], stddev=0.1), name="W2")          W2 = tf.get\_variable(              "W2\_"+str(i),              shape=[3, 1, num\_filters, num\_filters],              initializer=tf.contrib.layers.xavier\_initializer\_conv2d())          b2 = tf.Variable(tf.constant(0.1, shape=[num\_filters]), name="b2\_"+str(i))          conv2 = tf.nn.conv2d(              h1,              W2,              strides=[1, 1, 1, 1],              padding="SAME")          h2 = tf.nn.relu(tf.nn.bias\_add(conv2, b2), name="relu2")          self.l2\_loss += tf.nn.l2\_loss(W2)          self.l2\_loss += tf.nn.l2\_loss(b2)          return h2 |

图4.7 ResCNN模型结构的具体实现

## 4.7.4 模型参数设置

本章所述模型的超参数具体设置如表4.2所示。

|  |  |
| --- | --- |
| 参数名称 | 参数大小 |
| 卷积核大小(filter\_size) | 3 |
| 字向量维度 | 100 |
| 相对位置向量维度 | 5 |
| 最大相对位置 | 50 |
| 最小相对位置 | 50 |
| 过滤器数量 | 32 |
| Batch\_size | 32 |
| 学习率 | 0.001 |
| Dropout比例 | 0.5 |

表4.2 关系抽取模型超参数设置

卷积核的大小决定了在卷积操作中滑动窗口每次“滑过”的特征数量，也就等于决定了一次可以对多少个特征进行计算；字向量是用的本文所涉及的金融领域信息抽取任务的预训练字向量；过滤器数量表明可以对输入信息进行多少种特征计算；学习率可以控制模型的学习进度，如果学习率过大容易震荡且损失值很大，如果学习率过小容易产生过拟合且收敛速度太慢。本文主要通过这几个参数的对比实验来确认不同参数对模型效果的影响，实验采用F1值作为模型性能的评价标准，实验结果如表4.3所示。

|  |  |
| --- | --- |
| 参数设置 | F1值 |
| 卷积核大小=3，过滤器数量=50，  学习率=0.001 | 69.24 |
| 卷积核大小=3，过滤器数量=50，  学习率=0.005 | 68.96 |
| 卷积核大小=3，过滤器数量=50，  学习率=0.01 | 66.32 |
| 卷积核大小=5，过滤器数量=50，  学习率=0.001 | 66.07 |
| 卷积核大小=5，过滤器数量=50，  学习率=0.005 | 64.23 |
| 卷积核大小=5，过滤器数量=50，  学习率=0.01 | 62.18 |
| 卷积核大小=7，过滤器数量=50，  学习率=0.001 | 61.34 |
| 卷积核大小=7，过滤器数量=50，  学习率=0.005 | 59.35 |
| 卷积核大小=7，过滤器数量=50，  学习率=0.01 | 58.27 |
| 卷积核大小=3，过滤器数量=100，  学习率=0.001 | 63.45 |
| 卷积核大小=3，过滤器数量=200，  学习率=0.001 | 63.26 |

表4.3 不同超参数下模型的测试结果

根据前9行实验对比可以发现，当卷积核大小为3，学习率为0.001时模型的表现效果更好，所以最后2行直接使卷积核大小为3、学习率为0.001来对比过滤器的数量。最后可以发现当过滤器数量为50的时候，模型的表现效果更好，最终得到了表4.2中的超参数设置。

## 4.7.1 不同模型的性能对比

本文除了对比了ResCNN中不同的超参数对模型效果的影响，在本章节还对比了ResCNN外别的模型在相同输入和条件下的性能进行模型评估，其它的模型包括：CNN、CNN+Attention，实验的评价标准为模型的F1值，实验结果如表4.4所示。

|  |  |
| --- | --- |
| 模型名称 | F1值 |
| ResCNN | 69.24 |
| CNN | 53.37 |
| CNN+Attention | 65.48 |

表4.4 不同模型在相同输入下的性能对比

可以看出，由于关系类别较多以及语料数量较少的原因，模型的表现效果都不是特别好，不过相对而言ResCNN比CNN以及CNN+Attention的性能更加优秀。随着人工标注数据的不断增多结合一些人为制定的规则，关系抽取的效果会越来越好、

## 4.8 本章小结

本章主要介绍了本文实现信息抽取任务中关系抽取子任务的模型设计细节和详细的实现过程，并把实验结果与一些该任务中常用的方法做了对比分析，并分析了各个参数对模型最终效果造成的影响以及模型最后选取的参数。

# 第五章 总结与展望

## 5.1 总结

本文以上海某大数据人工智能公司的知识图谱产品中的信息抽取模块为背景，介绍了该项目中信息抽取模块的两个子任务命名实体识别和关系抽取的研究现状，综述了基于深度学习的信息抽取任务所涉及的相关技术，包括：词向量、双向长短时记忆模型、深度残差网络、注意力机制和远程监督。阐述了文本所选择模型的详细设计和具体实现。

在模型选择方面，命名实体识别模型在对比了BiLSTM+CRF、CRF、LSTM、BiLSTM在数据集上的表现后，本文选择了性能较好的BiLSTM+CRF模型。此模型考虑了上下文信息的影响，也学习到了一些约束条件，有较好的分类效果；关系抽取模型是在对比了CNN、CNN+Attention和ResCNN在数据集上的表现后，选择了效果更好的ResCNN。ResCNN利用了残差学习的特性使网络结构学习到了更多的语言特性从而得到了更好的效果。最后，本文将两个模型固化成pb模型，可供后续进行预测使用。

## 5.2 进一步工作展望

由于项目的研发时间较短，所以关系抽取模型的效果相对来说还比较一般，希望在未来的时间里有机会在现有模型的基础上进行更多地尝试：

1. 加入Attention机制；
2. 利用PCNN代替CNN；
3. 利用新的损失函数。

# 

# 参 考 文 献

**[MUC-6, 1996] MUC-6, the Sixth in a Series of Message Understanding Conferences, was held in November 1996[OL].**<http://cs.nyu.edu/cs/faculty/grishman/muc6.html> **.**

**[Milleret al., 2000] Miller, Scott, Heidi Fox, Lance Ramshaw, and Ralph Weischedel. “A novel use of statistical parsing to extract information from text.” *In Proceedings of NAACL*, 2000.**

**[Bikel et al., 1999] Bikel D M,Schwarta R,Weischedel R M.An Algorithm that Learns What`s in a Name[J].*Machine Learning Journal Special Issue on Natural Language Learning*, 1999, 34(1-3): 211-231.**

**[Tsai et al., 2004] Tsai T,WU S,Lee C, et al. Mencius: A Chinese Named Entity Recognizer Using the Maximum Entropy based Hybrid Model[J]. *International Journal of Computational Linguistics & Chinese Language Processing*, 2004, 9(1):65-81.**

**[McCallum et al., 2003] McCallum A,Li W.Early Results for Named Entity Recognition with Conditional Random Fields, Features Induction and Web-enhanced Lexicons[C]. *In Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL*,2003: 188-191.**

**[张祝玉等, 2008] 张祝玉，任飞亮，朱靖波. 基于条件随机场的中文命名实体识别特征比较研究[C]. 见: *第4届全国信息检索与内容安全学术会议论文集*.2008.**

[Wu et al., 2015] Yonghui Wu, Min Jiang, Jianbo Lei,Hua Xu. Named Entity Recognition in Chinese Clinical Text Using Deep Neural Network. Stud Health Technol Inform. 2015;216:624-8.

[Z Huang et al., 2015] Zhiheng Huang, Wei Xu, Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv*, 2015, 1508.01991 [cs.CL]

**[Kambhatla, 2004] Kambhatla, Nanda.”Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations.” *In Proceedings of ACL*, 2004.**

**[Zhao and Grishman, 2005] Zhao, Shubin, and RalphGrishman. Extracting relations with integrated information using kernel methods. *In Proceedings of ACL*, 2005.**

[Culotta et al., 2006] Culotta, Aron, Andrew McCallum, and Jonathan Betz. Integrating probabilistic extraction models and datamining to discover relations and patterns in text. *In Proceedings of HLT-NAACL*, 2006.

[Mintz et al., 2009] Mintz, Mike, Steven Bill, RionSnow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. *In Proceedings of ACL-IJCNLP*, 2009.

[Socher et al. 2012] Socher , Richard, et al. Semantic compositionality through recursive matrix-vectorspaces. *In Proceedings of EMNLP-CoNLL*, 2012.

[Zeng et al., 2014] Daojian Zeng, Kang Liu, et al. Relation classification via Convolutional Deep Neural Network. *In Proceedings of COLING*, 2014

[Santos et al., 2015] Cicero Nogueira dos Santos, Bing Xiang, Bowen Zhou. Classifying Relations by Ranking with Convolutional Neural Networks. *In Proceedings of ACL*, 2015.

[Miwa er al., 2016] Makoto Miwa, Mohit Bansal. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. *In Proceedings of ACL*, 2016.

[Lin et al., 2016] Yankai Lin, Shiqi Shen, Zhiyuan Liu, et al. Neural Relation Extraction with Selective Attention over Instances. *In Proceedings of ACL*, 2016.

[Bengio et al., 2002] Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult[J]. *IEEE Trans Neural Netw*, 2002, 5(2):157-166.

[Hinton, 1986] Hinton G E. Learning distributed representations of concepts[C]. *Eighth Conference of the Cognitive Science Society*. 1986.

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

[Pennington et al., 2014] Jeffrey Pennington, Richard Socher, and Christopher D. GloVe: Global Vectors for Word Representation. *Manning*. 2014.

[Bengio et al., 2006] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. *Journal of Machine Learning Research*, 2006, 3(6):1137-1155.

[Mikolov et al., 2013a] Mikolov T, Chen K, Corrado G, et al. Efficient Estimation of Word Representations in Vector Space[J]. *Computer Science*, 2013.

[Mikolov et al., 2013b] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]. *International Conference on Neural Information Processing Systems*. Curran Associates Inc. 2013:3111-3119.

[Radford et al., 2017] Radford A, Jozefowicz R, Sutskever I. Learning to Generate Reviews and Discovering Sentiment[J]. 2017.

[Rumelhart et al., 1986] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. Learning representations by back-propagating errors[J]. *Nature*, 1986, 323(3):533-536.

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

[He K M et al., 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. Deep Residual Learning for Image Recognition[J]. *arXiv*, 2015, 1512.03385.

**[Hinton et al., 2012]** Hinton G E, Srivastava N, Krizhevsky A, et al. Improving neural networks by preventing co-adaptation of feature detectors[J]. *arXiv preprint arXiv*:1207.0580, 2012.

# 致 谢

转眼之间我的研究生生涯就要结束了，对南大有一丝丝不舍，对未来有一丝丝憧憬和期待，在此我要向南大的研究生老师们和同学们表达最真挚的感谢。

首先我要感谢我的毕业论文指导老师伏晓老师，感谢伏老师对我毕业论文的指导和修正。毕业论文开题前，伏老师与我就毕业论文事宜进行了详谈，给我毕业论文的选题和方向做出了指导和建议，在论文的写作过程中，伏老师一直耐心地给我写作意见，伏老师对毕业论文负责任的态度，给了我莫大的帮助。

其次我要感谢南京大学软件学院的所有老师们，感谢老师们严谨的教学态度和学术态度，帮助我培养了良好的学习习惯和学习态度。还要感谢周围同学们在我学习生活中给予的帮助，给了我继续前进的动力。

再次我要感谢我的家人，他们在我最困难的时候默默地支持和鼓励着我，让我战胜了困难，最终进入南京大学就读研究生，实现了自己的梦想。

我还要感谢所有在我实习的时候热心帮助我的同事们，在实验的时候有很多不懂的地方都是靠着同事们的帮助帮助我踩过一个又一个坑。

最后，由衷地感谢在百忙之中评阅论文的各位专家和教授。

# 版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期： 年 月