

## ABSTRACT

KASA, SUMA. Adaptor Grammar for Unsupervised and Semi-Supervised Entity Extraction. (Under the direction of Dr. Munindar P. Singh.)

Entity recognition, also known as entity extraction, is one of the important problems in natural language processing that identifies the parts of the text that belong to predefined categories. Unsupervised grammar induction is one of the non-trivial and interesting approaches in natural language processing for entity recognition. Popular grammar induction approaches include modelling natural language in the form of context-free grammar (CFG) or probabilistic context-free grammar (PCFG). While these approaches laid an important step, these grammars make independence assumptions that limit the ability to model natural language with these grammars.

Adaptor grammar is a variant of probabilistic context-free grammar that addresses some of its limitations by relaxing its independence assumptions. Though Adaptor Grammar was proposed first in 2006, their potential remains under-researched. The thesis explores novel ways of using Adaptor Grammar for Unsupervised Entity Extraction. On that front, this work proposes two important contributions to expand on the utility of Adaptor Grammar for Entity Recognition. First, a semi-supervised approach based on adaptor grammar to identify entities from unstructured text documents in a novel domain. Such domains lack sufficient annotated labels to apply traditional named entity recognition (NER). We adopt adaptor grammar since they have proved useful in unsupervised entity recognition. Existing adaptor grammar approaches for NER tasks require as input grammar rules based on domain knowledge or heuristics. However, creating grammar rules manually is tedious and doesn't generalize across datasets. Therefore, we propose a semi-supervised approach where we learn the grammar rules from one dataset with annotated labels and apply them to another dataset with unknown labels. Our approach produces strong results in comparison to state-of-the-art unsupervised approaches.

As its second contribution, this thesis proposes a novel variant of the adaptor grammar neural network based approach to detect named entities from unstructured text documents in novel domains. Adaptor grammar has proved useful in various unsupervised NER tasks such as identifying brands and products in online shopping queries. It also showed compelling performance in identifying the techniques and applications, referred to as concepts from titles of academic work. The traditional adaptor grammar takes a Bayesian non-parametric approach that generates a distribution over the parse trees. The posterior joint distribution of a sequence of trees in the adaptor grammar is represented in terms of a Pitman-Yor process. To remodel adaptor grammar to a neural network approach, we parameterize the probability of generating the sentence in terms of feed-forward neural network and marginalise using the Inside-Outside algorithm. Our approach is evaluated on two distinct named entity recognition tasks on multiple datasets. Our evaluation shows a significant improvement over the state-of-the-art grammar induction approaches.

© Copyright 2021 by Suma Kasa

All Rights Reserved

Adaptor Grammar for Unsupervised and Semi-Supervised  
Entity Extraction

by  
Suma Kasa

A thesis submitted to the Graduate Faculty of  
North Carolina State University  
in partial fulfillment of the  
requirements for the Degree of  
Master of Science

Computer Science

Raleigh, North Carolina

2021

APPROVED BY:

---

Dr. Anup Kalia  
External Member

---

Dr. Kemafor Ogan

---

Dr. Bitu Akram

---

Dr. Munindar P. Singh  
Chair of Advisory Committee

## **DEDICATION**

To my mother.

## **BIOGRAPHY**

Suma Kasa was born in rural parts of India to parents who value education more than anything else. She completed her school education moving across different places. She always wanted to be a scientist and found an interest in computers when she was in high school and exploring the subjects for college. At the age of 17, she started college in Indian Institute of Technology Bombay. The experience was overwhelming with a completely different culture. A few years later, she got an admit to the Masters program in Computer Science from North Carolina State University. This work materializes her work during the program.

## **ACKNOWLEDGEMENTS**

I would like to thank my advisors Dr. Munindar Singh and Dr. Anup Kalia for their tremendous help and understanding throughout. This work would not have been possible without their motivation. I extend my gratitude to my committee members Dr. Kemafor Ogan and Dr. Bitu Akram. I would also like to thank my colleagues Hui Guo and Parth Diwanji for the very fruitful discussions. Finally, I would like to thank Dr. George Rouskas and Kathy Luca for accommodating me and administrative support through difficult times.

## TABLE OF CONTENTS

<b>LIST OF TABLES</b> . . . . .	<b>vii</b>
<b>LIST OF FIGURES</b> . . . . .	<b>viii</b>
<b>Chapter 1 INTRODUCTION</b> . . . . .	<b>1</b>
1.1 Overview . . . . .	1
1.1.1 Rule Based Approaches . . . . .	1
1.1.2 Dictionary Based Approaches . . . . .	2
1.1.3 Grammar Based Approaches . . . . .	2
1.2 Contribution . . . . .	3
1.3 Thesis Outline . . . . .	3
<b>Chapter 2 BACKGROUND</b> . . . . .	<b>5</b>
2.1 Probability Basics . . . . .	5
2.1.1 Probability Models . . . . .	5
2.1.2 Probability axioms . . . . .	6
2.1.3 Probability rules . . . . .	6
2.1.4 Random Variables . . . . .	7
2.1.5 Probability Distributions . . . . .	7
2.1.6 Expectation . . . . .	8
2.2 Basics of Statistics . . . . .	8
2.3 Statistical Learning . . . . .	9
2.3.1 Maximum Likelihood Estimation . . . . .	9
2.3.2 Maximum A Posterior . . . . .	9
2.4 Inference by Approximation . . . . .	9
2.4.1 Sampling based inference methods . . . . .	10
2.4.2 Variational Inference . . . . .	12
2.5 Probabilistic Context Free Grammar (PCFG) . . . . .	14
2.6 Inside-Outside Probabilities . . . . .	15
2.6.1 Inside Probabilities . . . . .	15
2.6.2 Outside Probabilities . . . . .	15
2.7 Viterbi Algorithm . . . . .	16
2.8 Adaptor Grammar . . . . .	16
2.8.1 Pitman Yor Adaptor Grammar . . . . .	16
2.9 Deep Learning . . . . .	19
2.9.1 Feed Forward Networks . . . . .	19
2.9.2 Recurrent Neural Networks . . . . .	19
2.9.3 Word Embeddings . . . . .	20
<b>Chapter 3 Semi-Supervised Entity Recognition</b> . . . . .	<b>22</b>
3.1 Research Problem . . . . .	22
3.2 Method . . . . .	22
3.3 Experiments . . . . .	23
3.4 Results and Analysis . . . . .	24
3.5 Limitations . . . . .	25

<b>Chapter 4</b>	<b>Neural Adaptor Grammar</b>	<b>27</b>
4.1	Research Problem	27
4.1.1	Academic Concept Extraction	27
4.1.2	Named Entity Recognition	27
4.2	PUER: Parametric Neural Adaptor Grammar	28
4.2.1	Training	32
4.2.2	Inference	35
4.3	Experiments	35
4.3.1	Datasets	35
4.3.2	Baselines	36
4.3.3	Time and Space Complexity	36
4.3.4	Experimental Design	37
4.3.5	Hyper-parameters	37
4.3.6	Evaluation Measure	37
4.3.7	Number of Parameters	38
4.4	Results and Analysis	38
4.4.1	Dataset sizes	40
4.4.2	Ablation Study	40
4.4.3	Qualitative Analysis	41
4.4.4	Threats to Validity	42
4.5	Limitations	42
<b>Chapter 5</b>	<b>CONCLUSION</b>	<b>45</b>
5.1	Summary	45
5.2	Future Work	46
<b>BIBLIOGRAPHY</b>		<b>47</b>

## LIST OF TABLES

Table 3.1	Semi-supervised entity recognition - internal evaluation . . . . .	25
Table 3.2	Semi-supervised entity recognition - external evaluation . . . . .	25
Table 4.1	Adaptor grammar rules for academic concept extraction . . . . .	28
Table 4.2	Adaptor grammar rules for named entity recognition . . . . .	28
Table 4.3	Grammar rules for concept extraction task in Chomsky form . . . . .	29
Table 4.4	Grammar rules for named entity recognition task in Chomsky form . . . . .	29
Table 4.5	PUER - baseline comparison - test . . . . .	39
Table 4.6	PUER - baseline comparison - validation . . . . .	39
Table 4.7	Concept extraction - internal evaluation - modifier - Test . . . . .	40
Table 4.8	Concept extraction - internal evaluation - concept - test . . . . .	40
Table 4.9	Concept extraction - internal evaluation - modifier - validation . . . . .	41
Table 4.10	Concept extraction - internal evaluation - concept - validation . . . . .	41
Table 4.11	Entity recognition - internal evaluation - test . . . . .	42
Table 4.12	Entity recognition - internal evaluation - validation . . . . .	42
Table 4.13	Performance evaluation with data increase . . . . .	43
Table 4.14	Ablation study - evaluation against baselines . . . . .	43
Table 4.15	Ablation study - academic concept extraction - modifier . . . . .	44
Table 4.16	Ablation study - academic concept extraction - concept . . . . .	44
Table 4.17	Ablation study - named entity recognition . . . . .	44

## LIST OF FIGURES

Figure 2.1	Single layer Perceptron . . . . .	20
Figure 2.2	Multi-layer Perceptron . . . . .	20
Figure 2.3	Recurrent neural network . . . . .	20
Figure 4.1	Dependency graph in non-terminals for concept extraction grammar . . . . .	30
Figure 4.2	Dependency graph in non-terminals for entity recognition grammar . . . . .	31
Figure 4.3	Parametric neural adaptor grammar - autoencoder . . . . .	32
Figure 4.4	Examples of parse trees for academic concept extraction . . . . .	33

## CHAPTER

# 1

# INTRODUCTION

## 1.1 Overview

Named Entity Recognition (NER) is an important task for information extraction and multiple NLP applications like knowledge-base construction, conversational agents, and question answering. NER is particularly challenging for enterprise domains such as IT, healthcare etc. due to the unavailability of labeled datasets in spite of having huge unlabeled text, and evolving set of entities to be recognised. Thankfully, NER has a long history of research associated with diverse approaches. Most of the recent deep learning and language model approaches have been extremely useful for supervised NER. Supervised NER requires annotated data to learn the labels for the entities. Therefore, this thesis focuses on unsupervised approaches. The well-known unsupervised approaches for NER rely on rules, dictionaries, and grammar based approaches. Each of the earlier methods has a set of limitations that restricts the utility of these methods for unsupervised NER. Few such methods and their limitations to described briefly to understand the main motivation of this thesis.

### 1.1.1 Rule Based Approaches

One of the early rule-based approaches proposed by Collins and Singer [1999] considers initial seed rules for their DL-Cotrain algorithm. The approach takes proper nouns as input and classifies if the proper noun is a Person, Location or Organisation. The rules are of two kinds - spelling and contextual. Spelling rules match the string with exact spelling, for example, if the string matches “New York” or “California”, then the proper noun is identified as a location. Contextual rules check the context in which the proper noun appears. The contextual rules check the string preceding and succeeding the proper noun.

For example, if the proper noun is preceded by “Mr.”, then the proper noun is a person or if the proper noun is succeeded by “Incorporated”, then the proper noun is an organisation. This works also proposes a boosting algorithm which trains a weighted combination of the multiple classifiers based on the earlier rules. The main limitation of this approach is that manual work and domain knowledge is necessary to come up with the initial rules. In addition, the rules that apply for one domain may not generalize to other domains.

Another important approach described in Etzioni et al. [2005] starts with a domain-specific set of predicates and domain independent extraction patterns. The extraction algorithm is recursive with the predicates forming the base case for recursion, e.g., City(“New York”) and extraction patterns like NP1 such as NP2 imply that NP1 and NP2 have will have the same label. This method confines itself to eight extraction patterns and a defined set of predicates. Even this approach is limited by the manual effort required to come up with the predicates and extraction rules.

### **1.1.2 Dictionary Based Approaches**

Mosallam et al. [2014] use knowledge bases such as Freebase, Wikipedia, and Yago to extract entities. The main limitation of using dictionaries to extract entities is that there is a scope of ambiguity. This approach proposes a statistical method to disambiguate the entities based on the context the nouns appear in. Neelakantan and Collins [2014] use seed entities to extract relevant phrases, filter them, and build a dictionary for NER. Since this approach is applied and tested on virus and disease dataset from BioMed Central corpus<sup>1</sup>, the seed entities for virus include terms like influenza and hepatitis, and seed entities for Disease include terms like jaundice, tumor etc. The paper makes use of these seed terms and canonical correlation analysis (CCA) to train the low-dimensional embeddings by learning the similarity with the seed entities. In addition, Shang et al. [2018] propose AutoNER, a Fuzzy-LSTM-CRF that builds on the earlier approaches to handle falsely labeled entities from dictionaries. Such deep learning approaches have come into popularity recently to extract entities with the use of dictionaries. This approach also suggests dictionary tailoring to update the dictionary used for NER. The limitations with the dictionary based approaches is that external publicly available dictionaries may not contain domain-specific entities as discussing and developing dictionaries for domain specific entities requires manual effort [Mohapatra et al., 2018].

### **1.1.3 Grammar Based Approaches**

In contrast to dictionary and rule based approaches, grammar based approaches such as the probabilistic context-free grammar (PCFG) [Johnson, 1998, 2010] and the adaptor grammar [Elsner et al., 2009; Johnson, 2010; Krishnan et al., 2017; Zhai et al., 2016] are data driven and robust to noise. PCFG is defined by a set of production rules along with their probabilities. PCFG is used to generate parse trees for a sentence based on the production rules. The probability of generating the sentence is computed by multiplying the probabilities of the production rules that generated the parse tree for the sentence. PCFG

---

<sup>1</sup><https://www.biomedcentral.com/>

posits a strong independence assumption on the structure in terms of the rules of the grammar and the probabilistic dependencies by which the structure is realised.

The adaptor grammar presents a variation on PCFG to weaken its independence assumption. The adaptor grammar provides an option to define adaptors, a flexibility by which the probability distributed is realised that can expand a symbol based on how it has been rewritten in the past. The adaptor grammar redistributes the probability generated by PCFG based on the adaptor, thus mitigating the independence assumptions of a PCFG. Adaptor grammar has been successfully used in NER tasks from different domains such as brands and products extraction [Zhai et al., 2016], word segmentation [Johnson et al., 2007], person, organization, location detection [Elsner et al., 2009], and academic concept extraction [Krishnan et al., 2017].

The most important limitations with an adaptor grammar is that it takes a Bayesian approach and hence it needs its priors to be set. Such earlier approaches use prior distributions whose posterior computation is intractable, thereby relying on sampling algorithms to compute the posterior.

## 1.2 Contribution

The main contribution of this thesis is two-fold. First, a semi-supervised approach that uses an annotated dataset from one domain to learn the grammar rules from one dataset that can be leveraged to extract entities from a different dataset of a novel domain based on a postulate similarity in sentence structures is proposed and evaluated. Specifically, annotated MIT Restaurant dataset [Liu et al., 2013] is used to develop a grammar for each entity of the restaurant dataset and this grammar is applied on the MIT Movie dataset [Liu et al., 2013] to identify the entities that have similar salience to the entities in the earlier dataset.

Second, a novel parametric neural variant of the adaptor grammar is proposed by using distributed word representations and recurrent neural networks to parameterize the probability of generating a parse tree. The inside-outside approach [Baker, 1975] is used to model the infinite recursion behavior of adapted non-terminals. The inside-outside algorithm estimates the probability of the start symbol expanding to the whole sentence by recursively calculating the inside probability of a non-terminal expanding to a fixed span and the outside probability of the outside trees of this span.

## 1.3 Thesis Outline

This thesis document is outlined as follows. Chapter 1 forms the introduction to the problem. Chapter 2 discusses the background knowledge required to understand the remaining portion of the documents. Chapter 2 is divided into multiple sections. First section briefly introduces the probability concepts like prior, posterior, maximum likelihood estimation etc. Statistical concepts like variational inference, Markov Chain Monte Carlo (MCMC) sampling algorithm are discussed in the next sections following the probability concepts. Previous grammar induction techniques like probabilistic context-free grammar, adaptor grammar, inside-outside probabilities, and Viterbi algorithm are discussed deeply after the basics

of statistics. Finally, fundamentals of deep learning like word embeddings, multi-layer perceptrons, and recurrent neural networks are discussed in the last section.

Chapter 3 discusses the first contribution of this work, the semi-supervised approach to generate the rules required to identify entities on dataset from a novel domain from the rules from a different domain. Chapter 4 discusses the second contribution of this work, the Parametric Neural Adaptor Grammar, a parametric variant of the Adaptor Grammar with distributed representations for unsupervised entity extraction. These contributions are discussed in detail with different sections for research problem, method, experiments, results, analysis, limitations and future work, and conclusion.

## CHAPTER

# 2

# BACKGROUND

For this work, unsupervised named entity recognition problem is formulated as a machine learning problem which uses a probabilistic model to generate parse trees. Statistical inference on the training data is used to learn the probability distribution of the parse trees. The knowledge of fundamentals of probability model and the definition of prior, posterior and likelihood is required to better understand the learning process of the probability distributions. Basic understanding of variational inference and Markov Chain Monte Carlo estimation method is also required to understand the shortcomings of the earlier adaptor grammar methods and the baseline approaches against which the parametric neural adaptor grammar is evaluated. The last few sections of the chapter discuss the background of probabilistic context free grammar, adaptor grammar, inside outside probabilities, and Viterbi algorithm in depth.

## 2.1 Probability Basics

Most of the information in this section is inspired from David Dalpiaz's R for Statistical Learning<sup>1</sup>

### 2.1.1 Probability Models

A probability model is defined by two elements:

- Sample space, often denoted as  $\Omega$ , which is a set that contains all possible outcomes
- A probability function that assigns a non-negative number  $P(A)$  to an outcome  $A$ , which denotes how much outcome  $A$  is likely to occur

---

<sup>1</sup><https://davidalpiax.github.io/r4sl/probability-review.html>

### 2.1.2 Probability axioms

The probability function should satisfy the following set of axioms for a sample space,  $\Omega$

- $P(\Omega)=1$ , the probability of the entire sample space is 1
- $P(A)\geq 0$ , the probability of any event is greater than or equal to 0
- For mutually exclusive events  $E_1, E_2$  etc, probability of union of all these events is equal to the sum of the probabilities of individual events.

$$P\left(\bigcup_{i=1}^{\infty} E_i\right) = \sum_{i=1}^{\infty} P(E_i)$$

### 2.1.3 Probability rules

- Given an event A, and its complement  $A^c$ , the probability for the event follows these rules

$$P(A^c) = 1 - P(A)$$

- Given two events A and B, the addition rule is

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

- If A and B are disjoint, then

$$P(A \cup B) = P(A) + P(B)$$

- For n mutually exclusive events  $E_1, E_2, \dots, E_n$ , probability of union of all these events is equal to the sum of the probabilities of individual events.

$$P\left(\bigcup_{i=1}^n E_i\right) = \sum_{i=1}^n P(E_i)$$

- Conditional probability is defined as the probability of event A given that B occurs, if  $P(B)>0$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

- Multiplication rule can be obtained from the above conditional probability rule,

$$P(A) = P(B) \cdot P(A|B)$$

- For multiple events  $E_1, E_2, \dots, E_n$ , multiplication rule expands to chain rule

$$P\left(\bigcap_{i=1}^n E_i\right) = P(E_1) \cdot P(E_2|E_1) \cdot \dots \cdot P(E_n | \bigcap_{i=1}^{n-1} E_i)$$

- Suppose, events  $E_1, E_2, \dots, E_n$  are such that  $P(\bigcup_{i=1}^n E_i) = 1$  and  $P(E_i) > 0$ , then for any event  $B$  such that  $P(B) > 0$ , Bayes' rule dictates

$$P(E_i|B) = \frac{P(E_i)P(B|E_i)}{P(B)} = \frac{P(E_i)P(B|E_i)}{\sum_{i=1}^n P(E_i)P(B|E_i)}$$

- Two events  $A$  and  $B$  are independent if

$$P(A \cap B) = P(A)P(B)$$

- The events  $E_1, E_2, \dots, E_n$ , are independent if for every subset  $S$  of  $1, 2, \dots, n$

$$P\left(\bigcap_{i \in S} E_i\right) = \prod_{i=1}^n P(E_i)$$

## 2.1.4 Random Variables

A random variable is a real-valued function that maps a subset of the sample space to a real value. The random variable represents a function that is of interest and we ideally want to find out the probability of the random variables. Random variables can either be continuous or discrete. If the possible values of the random variables are continuous, they are called conditional random variables and if the possible values are discrete, they are called discrete random variables.

## 2.1.5 Probability Distributions

A probability distribution for a random variable represents the possible values of random variable and the probabilities of each possible value of the random variable. The probability distribution of a discrete random variable is specified by a probability mass distribution  $p(x) = p_X(x) = P(X = x)$ . The probability distribution of a continuous random variable is specified by the probability density function  $f(x)$ . The probability that a random variable  $X$  is between  $a$  and  $b$

$$P(a < X < b) = \int_a^b f(X) dX$$

### 2.1.6 Expectation

Expectation of a function ( $g(X)$ ) of a random variable ( $X$ ) is the weighted average of the values of the function with respect to the probability of the random variable  $X$ .

The expectation of  $g(X)$  for a discrete random variable  $X$  is given by

$$\mathbb{E}_{g(X)} = \sum_X g(X)p(X)$$

The expectation of  $g(X)$  for a continuous random variable  $X$  is given by

$$\mathbb{E}_{g(X)} = \int_X g(X)p(X)dX$$

## 2.2 Basics of Statistics

While statistics is a vast area that deals with a wide range of problems, we discuss the concepts that are essential to understand the adaptor grammar. Typically, we assume that the this probabilistic model belongs to a known probability distribution or a combination of such distributions and learn the parameters of the probability distributions that optimizes with respect to a loss function. We define the concepts of likelihood and posterior that are used in learning these parameters where  $\theta$  represents the model parameters and  $X$  represents data:

Few concepts of statistics are discussed in this section that are essential to understand the adaptor grammar. A statistical model defines a mathematical model that generalises the data generation process under a set of assumptions. Given the observed data  $X$ , a mathematical model  $\theta$  defines the probability distribution of generating  $X$  under the assumptions of  $\theta$ . The goal of the approach is to develop a probabilistic model  $P$  defined by the probability axioms and rules that generates the sample text data  $S$  and effectively generalises to other similar text data. Typically, this probability distribution is assumed to belong to a known probability distribution family or a combination of such distributions and the parameters of the probability distribution are learnt by optimizing with respect to a loss function. Generally, the optimization is done with respect to likelihood or posterior. The concepts of prior, posterior, and likelihood are discussed below.

- **Prior:** The prior is the initial probability distribution that might represent the beliefs about the data
- **Likelihood:** Conditional probability of data given the model parameters  $P(X|\theta)$
- **Posterior:** Conditional probability of the model parameters given the data  $P(\theta|X)$

## 2.3 Statistical Learning

Statistical learning deals with finding the function that can be best used to predict the properties of data. In other words, statistical learning is used to learn the parameters  $\theta$  that generalises best to the data  $X$  under the assumptions that the model belongs to a particular probability distribution. Maximum Likelihood Estimation (MLE) and Maximum A Posterior (MAP) are two inference methods to estimate the model parameters  $\theta$ .

### 2.3.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) approximates the model parameters ( $\theta_{MLE}$ ) by finding the parameters that maximizes the likelihood function for the given sample data

$$\begin{aligned}\theta_{MLE} &= \operatorname{argmax}_{\theta} P(X|\theta) \\ &= \operatorname{argmax}_{\theta} \log P(X|\theta)\end{aligned}$$

### 2.3.2 Maximum A Posterior

Maximum A Posterior (MAP) approximates the model parameters ( $\theta_{MAP}$ ) by finding the parameters that maximizes the posterior function for the given sample data

$$\begin{aligned}\theta_{MAP} &= \operatorname{argmax}_{\theta} P(\theta|X) \\ &= \operatorname{argmax}_{\theta} \log P(\theta|X) \\ &= \operatorname{argmax}_{\theta} \log \frac{P(\theta, X)}{P(X)} \\ &= \operatorname{argmax}_{\theta} \log \frac{P(X|\theta)P(\theta)}{P(X)} \\ &= \operatorname{argmax}_{\theta} (\log P(X|\theta) + \log P(\theta) - \log P(X)) \\ &= \operatorname{argmax}_{\theta} (\log P(X|\theta) + \log P(\theta))\end{aligned}$$

$P(X)$  is independent of  $\theta$

MAP inference takes both likelihood and prior into account to estimate the best parameters of the model. Often, MAP inference is too complex to compute because the posterior needs to be maximized among all the priors available, which makes the computation intractable and inefficient and relies on approximate inference methods as described in the next section.

## 2.4 Inference by Approximation

We discuss two main families of approximation algorithms for inference: sampling-based methods and variational inference.

### 2.4.1 Sampling based inference methods

In addition to MAP, sampling can also be used in computing expectations ( $\mathbb{E}[f(X)]$ ) of random variables that follow a given probabilistic distribution. Numerous sampling methods have been proposed to estimate intractable quantities because the distribution of the sample space is unknown or even if it known, sometimes there is no mathematical method to compute the integrals. In this section, two sampling algorithms of Markov Chain Monte Carlo (MCMC) family that are used in the earlier implementations of adaptor grammar are discussed.

Markov Chain Monte Carlo <sup>2</sup> methods provide an intelligent search mechanism over the high dimensional posterior distribution space. Monte Carlo methods involve random search over the search space and Markov chain methods implement search by hopping from one point in the space (referred to as state) to the other based on some predetermined mechanism. Combining these two methods, Markov Chain Monte Carlo methods perform memoryless searches with heuristically determined jumps, such that the final outcome of the sampling algorithm does not depend on the start state.

Most of the MCMC methods follow a specific algorithm as follows.

- Start with a random state from the sample space called current state  $\theta_{curr}$
- Determine the next state to jump to  $\theta_{new}$  according to the mechanism
- Use prior information and data to determine probabilistically if the jump is to be accepted or rejected
- Update the current state  $\theta_{curr}$  to the new state  $\theta_{new}$ , if the move is accepted. Else, reject
- Return all the accepted jumps after a fixed number of tries

The different Markov chain sampling algorithms vary on how  $\theta_{new}$  is selected and what the criterion is for acceptance or rejection. A Markov chain of samples,  $\theta_1, \theta_2, \dots, \theta_k$  starting from  $\theta_1$  is generated and these samples are used to find the MAP or posterior based on what the MCMC inference is intended for. For example, if the MCMC inference is being used for MAP, the distribution that maximizes  $\log(P(X|\theta)) + \log(P(\theta))$  is identified. If the MCMC inference is used to find the posterior, the normalizing quantity  $P(X)$  is estimated by adding the product of prior and likelihood at each of the accepted samples.

$$\begin{aligned} P(\theta|X) &= \frac{P(X|\theta)P(\theta)}{P(X)} \\ P(X) &= \int_{\theta} P(X|\theta)P(\theta) d\theta \\ &= \sum_{i=1}^k P(X|\theta_i)P(\theta_i) \end{aligned}$$

---

<sup>2</sup><https://www.quantstart.com/articles/Markov-Chain-Monte-Carlo-for-Bayesian-Inference-The-Metropolis-Algorithm/>

### 2.4.1.1 Metropolis-Hastings sampling algorithm

A target distribution  $g(x)$  proportional to the posterior probability is used to determine the next state  $\theta_{\text{next}}$  from the current state  $\theta_{\text{curr}}$ .  $\theta_{\text{next}}$  is obtained by adding a random sample from a normal distribution  $\Delta\theta \sim \mathcal{N}(0, \sigma^2)$ .  $\sigma$  is a parameter of the Metropolis-Hastings algorithm. If the ratio of the target distribution at this new value to the target distribution at the current value is greater than or equal to 1, then the new value is accepted and the process is repeated. If the ratio is less than 1, the new value is accepted with the probability equal to the ratio. The algorithm is outlined below.

1.

$$\begin{aligned}\theta_{\text{next}} &= \theta_{\text{curr}} + \Delta\theta && \text{where } \Delta\theta \sim \mathcal{N}(0, \sigma^2) \\ \rho &= \frac{g(\theta_{\text{next}}|X)}{g(\theta_{\text{curr}}|X)}\end{aligned}$$

---

**Algorithm 1** Metropolis-Hastings sampling

---

```
1: procedure MH-SAMPLE( $\theta_0, \sigma$ )
2:   Initialize  $\theta_0$ 
3:   for  $i < k$  do
4:      $\Delta\theta \sim \mathcal{N}(0, \sigma^2)$ 
5:      $\theta_i = \theta_{i-1} + \Delta\theta$ 
6:      $\rho = g(\theta_i|X)/g(\theta_{i-1}|X)$ 
7:     if  $\rho \geq 1$  then  $i \leftarrow i + 1$ 
8:     else  $i \leftarrow i + 1$  with probability  $\rho$ 
9:   return  $\theta_1, \theta_2, \dots, \theta_k$  ▷ Return all the accepted states
```

---

Markov chains satisfying the three conditions below, converge to a unique satisfying distribution.

1. **Irreducibility:** flexibility to move from one state to any other state
2. **Aperiodicity:** impossibility of getting stuck in an oscillation
3. **Positive Recurrence:** taking finite time to get back to the original state starting from any state

The basic intuition is that the newer states move towards a denser distribution of higher probability. Sampling from a normal distribution ensures that the next value can be chosen from anywhere on the sample space (irreducibility). Accepting or rejecting randomly when the acceptance ratio is less than 1 prevents the algorithm from getting stuck at a local maximum (aperiodicity and positive recurrence). Since only the ratio of the target function is used, computations are inexpensive compared to computing the normalizing factor of the posterior.

### 2.4.1.2 Gibbs sampling

Gibbs sampling is used to estimate the joint probability distribution of a vector of parameters  $\theta = (\theta_1, \theta_2, \dots, \theta_k)$  by sampling from the conditional distributions of each parameter with respect to the data and the remaining parameters as observed. Every new state is accepted. Iterating over all the parameters once finishes one cycle. Gibbs sampling can be thought of as a search in a grid fashion because each sampling moves the parameters in one direction. A target distribution like  $g(x)$  is not required for Gibbs sampling, which makes it efficient compared to Metropolis-Hastings algorithm. But if the conditional probability of any parameter is not easy to compute, Metropolis-Hastings can be used in the place of conditional probability.

$$\begin{aligned} & p(\theta_1 | \theta_2, \dots, \theta_k, X) \\ & p(\theta_2 | \theta_1, \theta_3, \dots, \theta_k, X) \\ & \dots \\ & p(\theta_k | \theta_1, \dots, \theta_{k-1}, X) \end{aligned}$$

### 2.4.2 Variational Inference

Variational inference works by finding a function  $q(\theta) \in Q$ , where  $Q$  is a family of distributions, that approximates the posterior  $p(\theta|X)$  by minimizing the KL divergence of the posterior  $p(\theta|X)$  from the approximate target function,  $q(\theta) \in Q$ . KL divergence is a measure of mismatch between two distributions over the same domain.

$$\begin{aligned} \text{KL}(q(\theta) || p(\theta|X)) &= \int q(\theta) \log \frac{q(\theta)}{p(\theta|X)} \\ F(q) &= \min_{q(\theta) \in Q} \text{KL}(q(\theta) || p(\theta|X)) \end{aligned}$$

KL divergence of distribution  $q$  from distribution  $p$ , ( $\text{KL}(q||p)$ ) has the following properties.

- $\text{KL}(q||p) \geq 0$
- $\text{KL}(q||p) = 0 \implies q = p$
- $\text{KL}(q||p) \neq \text{KL}(p||q)$

Two important issues to minimize the KL divergence need to be addressed to use variational inference. First, the posterior distribution  $p(\theta|X)$  is not known to compute the KL divergence. Second, it is not possible to perform an optimization of a distribution with respect to a distribution. To alleviate these issues, a different optimization function can be used with the following justification.

$$\begin{aligned}
\log p(X) &= \int q(\theta) \log p(X) d\theta \\
&= \int q(\theta) \log \frac{p(X, \theta)}{p(\theta|X)} d\theta \\
&= \int q(\theta) \log \frac{p(X, \theta) q(\theta)}{p(\theta|X) q(\theta)} d\theta \\
&= \int q(\theta) \log \frac{p(X, \theta)}{q(\theta)} d\theta - \int q(\theta) \log \frac{p(\theta|X)}{q(\theta)} d\theta \\
&= \int q(\theta) \log \frac{p(X, \theta)}{q(\theta)} d\theta + \int q(\theta) \log \frac{q(\theta)}{p(\theta|X)} d\theta \\
&= \mathcal{L}(q(\theta)) + \text{KL}(q(\theta) || p(\theta|X))
\end{aligned}$$

$\mathcal{L}(q(\theta))$  is called Evidence Lower Bound (ELBO).  $p(X)$  is independent of  $\theta$ , which means that the sum of the two terms on the right side of the equation is constant. Therefore, minimizing KL divergence yields the same results as maximizing ELBO. The final optimization function is now  $\mathcal{L}(q(\theta))$  which needs to be optimized in contrast to minimizing the KL divergence. Further breaking this down,

$$\begin{aligned}
\mathcal{L}(q(\theta)) &= \int q(\theta) \log \frac{p(X, \theta)}{q(\theta)} d\theta \\
&= \int q(\theta) \log \frac{p(X|\theta)p(\theta)}{q(\theta)} d\theta \\
&= \int q(\theta) \log p(X|\theta) d\theta - \int q(\theta) \log \frac{q(\theta)}{p(\theta)} d\theta \\
&= \mathbb{E}[\log p(X|\theta)] - \text{KL}(q(\theta) || p(\theta))
\end{aligned}$$

The next issue to be addressed is to find the appropriate distribution for  $q(\theta)$ . Two such widely used distributions are discussed below.

### 2.4.2.1 Mean Field Approximation

Suppose  $\theta$  is a vector of  $k$  parameters  $(\theta_1, \theta_2, \dots, \theta_k)$ , the function  $q(\theta)$  is split into product of different functions of each parameter, i.e.  $q_i(\theta_i)$  for each  $\theta_i$ .

$$q(\theta) = \prod_{j=1}^m q_j(\theta_j)$$

With this factorization,  $(q(\theta))$  is optimized by block co-ordinate ascent. Block coordinate ascent optimizes the function by fixing all factors  $q_i(\theta_i)_{i \neq j}$  except one  $(q_i(\theta_i))$  and maximizing with respect to  $\theta_i$ . The main restriction of this approach is that each of the factors are considered independent, which is a steep assumption.

### 2.4.2.2 Parametric Approximation

A parametric family of variational distributions is fixed and optimized the evidence lower bound with respect to the parameters for parametric approximation. The main limitations of this method is that the parametric family might not be complex enough to model the data and even if the complexity is enough, the data might not be sufficient to train it well. If evidence lower bound is differentiable with respect to  $\theta$ , a numerical optimization solver can be used to find the parameters.

$$q(\theta) = q(\theta|\lambda) \quad \lambda - \text{parameters}$$

$$\mathcal{L}(q(\theta|\lambda)) = \arg \max_{\lambda} \int q(\theta|\lambda) \log \frac{p(X, \theta)}{q(\theta|\lambda)} d\theta$$

## 2.5 Probabilistic Context Free Grammar (PCFG)

A probabilistic context-free grammar (PCFG) [Johnson, 1998] captures the probabilities over the derivations of a context-free grammar (CFG). A PCFG is represented as a tuple,  $\langle N, W, R(N), \theta \rangle$ , where  $N$  and  $W$  are finite and mutually disjoint sets of *non-terminal* and *terminal symbols*, respectively, and  $R(N) = \{X \rightarrow v\}$  is a finite set of productions or *probabilistic grammar rules* with probability set  $\theta$  for non-terminals  $X \in N$ , where  $v = Y_1 Y_2 \dots Y_n, Y_i \in N \cup W$ . For each non-terminal  $A \in N$  in tree  $T_A$ ,  $\theta$  defines a distribution  $G_A$  over the possible productions for  $A$ ,  $R_A = \{r_A: A \rightarrow \beta \in R(N)\}$ , where  $\beta = B_1 B_2 \dots B_n, B_i \in N \cup W$ , such that

$$\sum_{A \rightarrow \beta \in R(A)} \theta_{A \rightarrow \beta} = 1$$

More specifically, for a non-terminal  $\alpha \in U$ , consider the collection of grammar rules of  $\alpha: r_{\alpha} \rightarrow \beta \in R(N)$  where  $\beta \in N \cup W$ , and each rule is associated with a probability  $\theta_{r_{\alpha} \rightarrow \beta}$  such that  $\sum_{r_{\alpha} \rightarrow \beta \in R(N)} \theta_{r_{\alpha} \rightarrow \beta} = 1$ . PCFGs assume that each non-terminal expands to its children independently of the others (hence context free). Thus,  $G_A$  can be defined recursively over the tree  $T_A$ :

$$G_A = \sum_{A \rightarrow \beta} \theta_{A \rightarrow \beta} \text{TREEDIST}_A(G_{B_1}, \dots, G_{B_n})$$

where  $\text{TREEDIST}_A(G_{B_1}, \dots, G_{B_n})$  is the distribution over trees,  $\text{TREE}_A(T_{B_1}, \dots, T_{B_n})$ , where  $A$  is the root node and each subtree  $T_{B_i}$  is generated independently from  $G_{B_i}$ . This distribution satisfies:

$$\text{TREEDIST}_A(G_{B_1}, \dots, G_{B_n}) = \prod_{i=1}^n G_{B_i}$$

The definition of PCFG provides an efficient way to learn the probability of different parse trees. The probability of a parse tree is the product of probabilities of each rule that forms the parse tree. Probability of a sentence is the sum of the probabilities of all the parse trees generating the sentence. Probability of a non-terminal  $A$  spanning a string  $w$  is the sum of the probabilities of all the parse trees with root label  $A$

and yield  $w$ .

## 2.6 Inside-Outside Probabilities

Inside-outside probabilities are used to compute the likelihood of produce an output  $\mathbf{w} = (w_1, w_2, \dots, w_n)$ ,

$$\sum_{t' \in T(\mathbf{w})} P(t') \quad \text{where } T(\mathbf{w}) \text{ is the set of parse trees yielding } \mathbf{w}$$

Enumerating all the parse trees and calculating the sum of probabilities of the parse trees is computationally very expensive. Instead, the probabilities can be computed by tabulating to avoid calculating the probabilities of subtrees repeatedly. Inside probabilities ( $\alpha$ ) and outside probabilities ( $\beta$ ) are defined as follows to calculate these probabilities.

### 2.6.1 Inside Probabilities

$\alpha$  is a  $N \times W \times W$  matrix where  $N$  is the number of non-terminals and  $W$  is the length of the input word sequence  $w$ .  $\alpha_{A,i,j}$  refers to the sum of the probability of the trees rooted at  $A$  yielding the word sequence  $w_i, w_{i+1}, \dots, w_j$ . Inside probabilities are calculated bottom-up according to the following rules

Base case for  $\alpha$  non-terminal  $A$ ,  $\forall i, 1 \leq i \leq W$  and  $A \in N$

$$\alpha_{A,i,i} = \begin{cases} P(A \rightarrow w_i) & \text{if } A \rightarrow w_i \in R, \\ 0 & \text{else.} \end{cases}$$

$\forall i, j, 1 \leq i, j \leq W$  and  $A \in N$

$$\alpha_{A,i,j} = \sum_{A \rightarrow BC \in R} \sum_{k=i}^j P(A \rightarrow BC) \alpha_{B,i,k} \alpha_{C,k+1,j}$$

### 2.6.2 Outside Probabilities

$\beta$  is also a  $N \times W \times W$  matrix where  $\beta_{A,i,j}$  refers to the sum of the probabilities of parse trees given that the parse tree has a subtree with root  $A$  expanding to the word sequence  $w_i, w_{i+1}, \dots, w_j$ . Outside probabilities are calculated top-down according to the following rules

Base case for  $\alpha$  non-terminal  $A$ ,  $\forall A \in N$

$$\beta_{A,1,W} = \begin{cases} 1 & \text{if } A = S, \\ 0 & \text{else.} \end{cases}$$

$\forall i, j, 1 \leq i, j \leq W$  and  $A \in N$

$$\begin{aligned} \beta_{A,i,j} = & \sum_{B \rightarrow AC \in R} \sum_{k=j+1}^n P(B \rightarrow AC) \beta_{B,i,k} \alpha_{C,k+1,j} \\ & + \sum_{B \rightarrow CA \in R} \sum_{k=1}^{i-1} P(B \rightarrow CA) \beta_{B,k,j} \alpha_{C,k,i-1} \end{aligned}$$

Probability of a parse tree for  $w$  with a node labeled  $A$   $p_{A,i,j}$  that spans  $w_i, \dots, w_j$  is  $\alpha_{A,i,j} \beta_{A,i,j}$ . Therefore,  $P(w)$  is  $\alpha_{S,1,W}$  because  $\beta_{S,1,W} = 1$

## 2.7 Viterbi Algorithm

Viterbi algorithm [Sammut and Webb, 2010] is used to find the most probable parse tree from the computed tables of inside-outside probabilities  $(\alpha, \beta)$  for an input sentence  $w$  and PCFG  $\mathbf{P} \langle N, W, R(N), \theta \rangle$ . The entity  $V_{A,i,l}$  is defined as a tuple  $\langle p, A \rightarrow BC, l_1 \rangle$  where  $A \rightarrow BC$  is the rule that  $A$  should expand to obtain the parse tree with maximum probability that yields  $w_i, \dots, w_{i+l-1}$ , index  $l_1$  is where the non-terminal  $C$  starts and  $p$  is the probability of this subtree. The algorithm starts bottom up and for each rule expanding  $A$  ( $A \rightarrow BC$ ), rule that maximizes  $p_{A \rightarrow BC} * p_{B,i,l_1-1} * p_{C,l_1,j}$  for all  $l_1$  between  $i$  and  $j$ , as explained in the following pseudo code.

## 2.8 Adaptor Grammar

An adaptor grammar [Johnson et al., 2007] augments the probabilities rules of PCFG by including an additional adaptor component  $\mathbf{C}$ , which can be used to modify the independence assumption made by the PCFG. An adaptor grammar is denoted by a tuple,  $\langle N, W, R(N), \theta, D, \mathbf{C} \rangle$  where  $\langle N, W, R(N), \theta \rangle$  forms a PCFG,  $D$  is the set of adapted non-terminals and  $\mathbf{C}_A$  is an adaptor operating on the PCFG for each non-terminal  $A$  in  $D$ . In an adaptor grammar, each symbol  $A \in N$  is associated with two distributions over  $T_A$ ,  $G_A$  and  $H_A$ , where  $\mathbf{C}$  maps  $G_A$  to  $H_A$ . Therefore,  $G_A$  and  $H_A$  are defined as follows:

$$\begin{aligned} G_A &= \sum_{A \rightarrow \beta} \theta_{A \rightarrow \beta} \text{TREEDIST}_A(H_{B_1}, \dots, H_{B_n}) \\ H_A &\sim \mathbf{C}_A(G_A) \end{aligned}$$

The introduction of  $\mathbf{C}$  and  $H_A$  modifies the independence assumptions made by the PCFG. If  $\mathbf{C}$  is an identity function, then the adaptor grammar will be the same as the PCFG.

### 2.8.1 Pitman Yor Adaptor Grammar

All the earlier work on Adaptor Grammar provide modifications to one specific variety of Adaptor Grammar called Pitman Yor Adaptor Grammar (PYAG). These approaches vary in the inference method

---

**Algorithm 2** Viterbi algorithm

---

```
1: procedure VITERBI( $\alpha, \beta, \mathbf{P}$ )
2:   Initialize  $V_{A,i,1} = \langle p_{A \rightarrow w_i}, A \rightarrow w_i, - \rangle \forall i \in 1, \dots, W$ 
3:   for  $l \in 2, \dots, W$  and  $i \in 1, \dots, n - l$  do
4:     for  $A \rightarrow BC \in R_A$  and  $l_1 \in 1, \dots, l$  do
5:       if  $V_{B,i,l_i} \neq \Phi$  and  $V_{C,i+l_1,l-l_1} \neq \Phi$  then
6:          $p_{new} = p_{A \rightarrow BC} * V_{B,i,l_i}[0] * V_{C,i+l_1,l-l_1}[0]$ 
7:         if  $V_{A,i,l} == \Phi$  or  $V_{A,i,l}[0] < p_{new}$  then
8:            $V_{A,i,l} = \langle p_{new}, A \rightarrow BC, l_1 \rangle$ 
9:       for  $A \rightarrow B$  do
10:        if  $V_{B,i,l} \neq \Phi$  then
11:           $p_{new} = p_{A \rightarrow BC} * V_{B,i,l}[0]$ 
12:          if  $V_{A,i,l} == \Phi$  or  $V_{A,i,l}[0] < p_{new}$  then
13:             $V_{A,i,l} = \langle p_{new}, A \rightarrow B, - \rangle$ 
14:   return buildTree(S, 1, W)
15: function BUILDTREE(A, i, l)
16:   Initialize root= $\Phi$ 
17:   if  $V_{[A,i,l]} \neq \Phi$  then
18:     if  $V_{A,i,l}[1] == A \rightarrow BC$  then
19:       root.addChild(buildTree(B, i,  $V_{A,i,l}[2]$ ))
20:       root.addChild(buildTree(C,  $i + V_{A,i,l}[2]$ ,  $l - l_1$ ))
21:     if  $V_{A,i,l}[1] == A \rightarrow B$  then
22:       root.addChild(buildTree(B, i, l))
23:   return root
```

---

used to estimate the posterior of Pitman Yor adaptor process. PYAG caches the previously generated parse trees and increase the probability of expanding to these trees as the frequency increases. PYAG expands the parse tree from the start non-terminal and expands it to a previously generated parse tree  $k$  with probability  $\frac{n_k - a}{n + b}$  and to a new tree with probability  $\frac{ma + b}{n + b}$  where  $n_k$  is the number of times the the non-terminal expands to  $k$ ,  $n$  is the total number of times the non-terminal  $A$  is expanded,  $m$  is the number of distinct trees that  $A$  has been expanded to, and  $a$  and  $b$  are real valued parameters such that  $a \in [0, 1]$  and  $b \geq 0$ . This specification of the PYAG follows a “rich gets richer” dynamic where the probability of expanding to a tree is weighed by the frequency of the tree. Adaptor for adaptor non-terminal  $A$ , given by  $C_A$  is associated with  $a_A, b_A, \mathbf{x}_A$  and  $n_A$ .  $a_A, b_A$  are the real valued parameters described above,  $x_A$  is the sequence of previously generated subtrees with  $A$  as the root and  $n_A$  is the list of the frequencies of the subtrees in  $x_A$  such that the  $i^{th}$  element in  $n_A$  refers to the frequency of the  $i^{th}$  in tree  $x_A$ .

PYAG also defines  $u$  as a pair  $(t, l)$  where  $t$  is a parse tree such that  $t \in x_S$  and the function  $l(\cdot)$  represents an index function.  $l(q)$  gives the index of the entry in  $\mathbf{x}_A$  for subtree  $t'$  of  $t$  rooted at  $q$ , i.e, such that  $x_{A_l(q)} = t'$ . The sequence  $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n$  can be decoded to produce the adaptor states at each step after  $\mathbf{u}$  is generated. PYAG uses MCMC inference to sample from the posterior over the analyses  $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n$  given the text strings  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$  where  $\mathbf{u}_i$  is the analysis for a training sentence  $\mathbf{x}_i$ . PYAG assumes the Dirichlet prior for the PCFG production probabilities  $(\theta)$ , where  $\alpha_{A \rightarrow \beta}$  is the Dirichlet parameter for the rule  $A \rightarrow \beta$ .

$$P(\theta|\alpha) = \prod_{A \in \mathcal{N}} \frac{1}{B(\alpha_A)} \prod_{A \rightarrow \beta \in R_A} \theta_{A \rightarrow \beta}^{\alpha_{A \rightarrow \beta} - 1} \quad \alpha_A \text{ is the subsequence of } \alpha \text{ indexed by } R_A$$

$$B(\alpha_A) = \frac{\prod_{A \rightarrow \beta \in R_A} \tau(\alpha_{A \rightarrow \beta})}{\tau(\sum_{A \rightarrow \beta \in R_A} \alpha_{A \rightarrow \beta})} \quad \tau(x) \text{ is the generalized factorial function}$$

The joint probability of  $\mathbf{u}$  with the Dirichlet and the non-terminal specific Pitman-Yor parameters  $(\mathbf{a}$  and  $\mathbf{b})$  for the PYAG is given by

$$P(u|a, b, \alpha) = \prod_{A \in \mathcal{N}} \frac{B(\alpha_A + \mathbf{f}_A(\mathbf{x}_A))}{B(\alpha_A)} PY(\mathbf{n}_A(\mathbf{u})|\mathbf{a}, \mathbf{b})$$

where  $f_{A \rightarrow \beta}(\mathbf{x}_A)$  is the number of times the production  $A \rightarrow \beta$  expands to a root node of a tree in  $\mathbf{x}_A$ , and  $\mathbf{f}_A(\mathbf{x}_A)$  is the sequence of frequency of rule  $r$  in  $R_A$  expands to a root node of a tree in  $\mathbf{x}_A$ . The posterior distribution over all the analyses  $u$  is obtained by normalizing  $P(u|a, b, \alpha)$  over all the possible analyses of  $u$ . Since computing  $P(u|a, b, \alpha)$  over all possible  $u$  is intractable, the posterior approximation methods discussed in the earlier methods can be used to obtain these parameters. Johnson et al. [2007] use Metropolis-Hastings sampling method to perform MCMC inference to approximate the posterior. Efficiency of MCMC inference often depends on the number of samples and the probability of exploration making it slow in practice. Cohen et al. [2010] use variational inference to approximate this posterior. Variational inference requires that the counts of expanding of different rules to be calculated to maximize

the Evidence Lower Bound (ELBO). Zhai et al. [2014] introduce an online training algorithm which performs variational inference on a smaller batch and adds sampling within the batch processing to estimate the ELBO.

## 2.9 Deep Learning

Deep learning is a subset of machine learning where the probability distribution is modeled as a multi-layered neural networks. Neural network is a network of neurons characterized by weights and biases. Each neuron takes an input and operates using weights and biases to produce an output. Most often, this output is passed through a non-linear activation function because without the activation function, the family of distributions that can be modeled is restricted to linear models. Neural network architectures vary in how the connections are made between neurons. Two such architectures are discussed in the following sections.

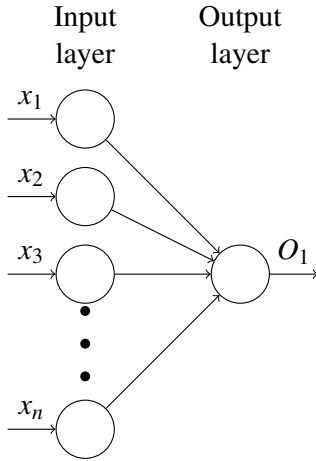
### 2.9.1 Feed Forward Networks

Feed forward network, also known as multi-layer perceptron is a sequence of single layer perceptrons where the output of each layer is passed as an input to the next layer. Single layer perceptron is associated with weights  $w_1, w_2, \dots, w_m$  and bias  $b$ , takes a vector of size  $m$  as input and outputs  $z = \sum_{i=1}^m w_i * x_i + b$ . This output signal is amplified or attenuated using a non-linear activation function. A multi-layer perceptron is a series of perceptrons where each layer takes the last layer's output as input and produces output for the next layer. The difference of the output of the last layer and the expected output is computed as the loss and this loss is propagated backwards through the layers according to an optimizer function to update the weights. Feed forward networks are named so because the input signal travels forward through the layers and the algorithm to update weights is called back propagation algorithm because the weights are updated backwards starting from the output layer.

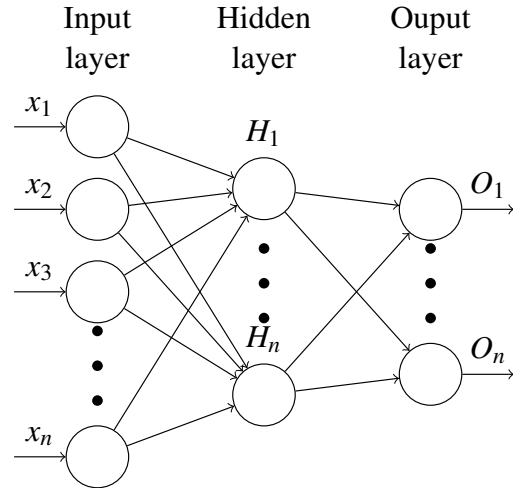
### 2.9.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a variety of neural networks that work well with sequential input like time series, text etc. Recurrent neural networks are similar to feed forward networks except that RNNs have the same weights for every layer for theoretically any length of the input sequence. The following image Figure 2.3 gives an idea of how an RNN works.

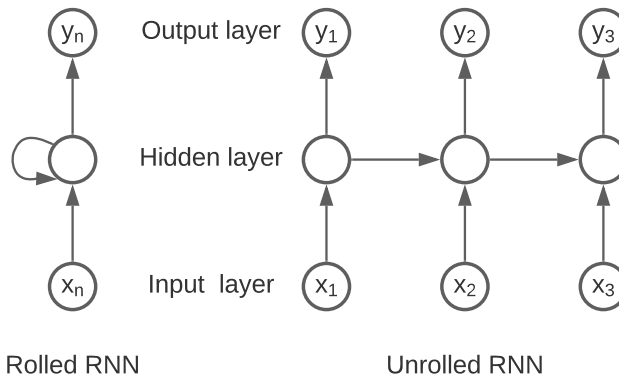
RNN makes use of back propagation through time (BPTT) to update the weights and optimize on the loss function. BPTT adds the loss at each time step contrary to the feed forward networks which only optimizes on the loss at the last step. While RNN promises a method to tackle sequential input, RNNs are not effective for input of longer lengths. RNNs suffer from two important issues: vanishing gradient and exploding gradient. For longer lengths, if the gradient is small, the gradient keeps getting smaller as the error propagates backward. If the initial gradients are large to start with, then the gradient explodes. Variations of RNNs like gated RNNs, long short term memory (LSTM) etc., are proposed to mitigate the vanishing and exploding gradient problems. LSTM uses three gates - input, output and forget gates - to



**Figure 2.1** Single layer Perceptron



**Figure 2.2** Multi-layer Perceptron



**Figure 2.3** Recurrent neural network

control information flow between the hidden states and cell states. Gated Recurrent Units (GRU) use two gates instead - reset and update gate. GRU alleviates the vanishing and exploding gradient problems while being computationally inexpensive compared to LSTM.

### 2.9.3 Word Embeddings

Word embeddings are learned and distributed representations of words, which are the building blocks of text. Word embeddings encode semantic and syntactic regularities about the words based on how they are learned. If the words are properly and sufficiently trained, the dot product of the word embeddings of similar words is higher. In addition, dense and low-dimensional word embeddings also present a computationally efficient method to use neural networks.

Word2Vec algorithm [Mikolov et al., 2013] proposes two methods to learn the word embeddings -

Skip-gram model and bag of words model. Skip-gram model learns the embeddings by predicting the context words given the current words in a window. Bag of Words model learns the embeddings by predicting the current word given the context words. The probability of predicting a word  $w_i$  given  $w_j$  is given by the dot product of the embeddings of words  $w_i$  and  $w_j$ . GloVe algorithm [Pennington et al., 2014] learns the word embeddings using the co-occurrence matrix of the words in addition to using the context.

## CHAPTER

# 3

# SEMI-SUPERVISED ENTITY RECOGNITION IN NOVEL DOMAINS

## 3.1 Research Problem

The problem of detecting named entities is defined as follows: given a sentence  $S$  with words  $w_1, \dots, w_n$ , a label  $l_i$  is assigned to each word  $w_i$  such that  $l_i$  belongs to the set of named entities  $E$ . This work provides a semi-supervised approach in a sequence of steps to learn adaptor grammar rules from an annotated dataset and apply them to a dataset with unknown labels to extract relevant entities.

## 3.2 Method

The semi-supervised approach to learn the adaptor grammar rules for a dataset with unknown labels from a different annotated dataset is done in a series of steps as explained below.

**Generate PCFG rules for annotated dataset.** For each entity  $E_i$ , the sentences containing words labeled with these entities are identified and the sentences are split into noun phrases using spaCy's noun phrase chunker [Honnibal and Montani, 2017]. The noun phrases that do not contain the words labeled with  $E_i$  are filtered out. For each such noun phrase  $NP_j$  with one or more named entities, parse tree representation is generated along with parts-of-speech (POS) tags for each word in  $NP_j$  using an existing AllenNLP POS tagger [Gardner et al., 2017]. For example, given the noun phrase, *a reservation at that fine dining*, the annotated entity labels along with POS tags for each word are used to create a parse tree representation as follows: (Phrase (DT a) (Amenity reservation) (IN at) (DT that) (Amenity fine) (Amenity dining)).

For each entity  $E_i$ , the parse trees containing the entity are fed into NLTK [Bird and Klein, 2009] generator to obtain the seed rules for generating an adaptor grammar. For example, the resulting rule for the above parse tree in PCFG is  $\text{Phrase} \rightarrow \text{DT Amenity IN DT Amenity Amenity}$ .

**Infer adaptor grammar.** The PCFG rules obtained above are processed as specified next to infer the rules for adaptor grammar. First, the similar POS tags are grouped together. For example, the POS tags VB, VBD, VBP, VBG, VBZ, and VBN are mapped to Verb; NN, NNS, NNP, NNPS to Noun; DT, JJ, JJR, and JJS to Adj; and IN to Prep. The duplicate consecutive tags of the same label or POS tag are reduced to one tag. For example, the above rule will be transformed to  $\text{Phrase} \rightarrow \text{Adj Amenity Prep Adj Amenity}$ . This processing should not affect the expressive power of the grammar because for each label and POS tag, the rules of the form  $\text{Label} \rightarrow \text{Words}$  and  $\text{Words} \rightarrow \text{Word Words}$  are added. As a result, each label can expand to a list of words instead of one word. The entity labels in these rules is changed to the word “Entity“ so that the evaluation remains consistent. Thus the further rule is further changed to  $\text{Phrase} \rightarrow \text{Adj Entity Prep Adj Entity}$ .

**Extract entities** Zhai et al. [2014] propose an online stochastic hybrid inference approach on Pitman-Yor adaptor grammar based on Pitman Yor process [Pitman and Yor, 1997] and adaptor grammar definition [Johnson et al., 2007], which is used to extract entities using the above generated rules. This approach uses Markov Chain Monte Carlo (MCMC) inference for non-parametric inference interleaved with variational inference. Without the independence assumptions made by PCFG, calculating the probability distributions is computationally expensive. Variational inference alleviates the complexity limitation of the Adaptor Grammar by directly modeling the probability distribution but requires pre-processing based on all the data to establish the support for non-parametric models. MCMC avoids this pre-processing step as it discovers the support for non-parametric models online during inference.

To extract the entities from the unannotated dataset, the sentences are split into noun phrases using spaCy’s noun phrase chunker. The generated entity specific adaptor grammar is then applied on these noun phrases to produce the parse trees. Out of all the parse trees, the parse tree with the highest probability is chosen and the words labeled with the term “Entity” are identified as entities.

### 3.3 Experiments

Two sets of evaluations is used for the approach. First, an internal evaluation is performed by obtaining the precision (P), recall (R), and F1 score (F1) for each entity type recognised by the grammar on both the datasets. Second, an external evaluation is performed where this approach is compared to the state-of-the-art baseline approaches for unsupervised NER. The entire approach is demonstrated using two datasets. First, the annotated MIT Restaurant dataset containing 7660 sentences with words labeled to eight entities. These sentences are split into 26428 noun phrases. Second, the MIT Movie dataset whose labels are not used for training purpose contains 9775 sentences with 12 entities. 34451 noun phrases are obtained from this dataset.

For both datasets, each word in each sentence is annotated with their corresponding entity labels. The labels for the restaurant dataset belong to the set B-Restaurant\_Name, I-Restaurant\_Name, B-Rating, I-

Rating, B-Amenity, I-Amenity, B-Cuisine, I-Cuisine, B-Dish, I-Dish, B-Hours, I-Hours, B-Price, I-Price, B-Location, I-Location and Other. The B- labels refer to the first word of an entity and the I- labels refer to an entity’s intermediate words. This differentiation is ignored for simplicity, and each pair of B- and I- labels are combined. The resulting set of entity labels for the restaurant dataset is Restaurant\_Name, Rating, Amenity, Price, Location, Other. Similarly, the set of entity labels for the movie dataset is Actor, Title, Character, Year, Genre, Director, Genre, Review, Ratings\_Average, Rating, Plot, Song, and Trailer.

The semi-supervised approach is compared to both unsupervised and supervised approaches. The following unsupervised entity extraction algorithms are used as baselines to evaluate the current approach. **(1) MOD-ENT** Elsner et al. [2009] proposed a method to identify entities in noun phrases by assuming a Modifier|Named\_Entity|Preposition|Pronoun order in the noun phrases. Moreover, their grammar categorizes pronouns into classes like location and person. For example, the location class includes pronouns like where and there and the person class includes pronouns like who, they, and so on. **(2) BRAND-PROD** Zhai et al. [2016] proposed a method to identify brands and products from query strings. They assume the phrases can be reduced to BRAND or PRODUCT and BRAND PRODUCT, where each of BRAND and PRODUCT is further reduced to a string of words.

The following supervised approaches for entity recognition are used. **(1) MOVIE** This baseline trains an adaptor grammar directly on the rules obtained from the training subset of MIT Movie Dataset to extract entities. The approach first develops a PCFG from the training phrases and enhances this PCFG into an adaptor grammar. **(2) LSTM-CRF** Liu et al. [2018] and Vinodababu [2019] implement a model that co-trains for language modeling along with Long Short Term Memory (LSTM) followed by a Conditional Random Fields (CRF) layer. The CRF layer models the transitions between the entities [Huang et al., 2015]. The BiLSTM-CRF model uses word embeddings of dimension 100, which are trained along with the model. To keep the comparison fair, the model is trained on the Restaurant dataset for entities and evaluated on the Movie dataset. **(3) BERT-NER** Pretrained BERT [Devlin et al., 2019] for NER [Raj, 2019]. The model is fine-tuned on the restaurant dataset and evaluated on the movie dataset to keep the comparison fair.

### 3.4 Results and Analysis

Table 3.1 shows the results of the internal evaluation. The precision, recall, and F1 scores for each entity in the Restaurant dataset are greater than those in the Movie dataset. This is expected for several reasons. One, the adaptor grammar is learned from the Restaurant dataset. Two, the phrase in the Movie dataset could have possibly more than one entity; however, the entity-specific extraction shows results for only one kind of entity at a time. In addition, it is observed that for some entities in the Restaurant dataset, such as Hours, Location, and Price, the Movie dataset’s recall values are lower. This is possibly due to differences in the nature of entities present in both the datasets.

Table 3.2 shows the results for our external evaluation. The semi-supervised approach performs better than previous unsupervised approaches. This is due to two reasons. One, the adaptor grammar for MOD-ENT is based on a priori domain knowledge and for BRAND-PROD is based on heuristics.

**Table 3.1** Semi-supervised entity recognition - internal evaluation

Dataset	Restaurant			Movie		
	P	R	F1	P	R	F1
Name	0.45	0.71	0.55	0.34	0.42	0.38
Rating	0.42	0.49	0.45	0.38	0.42	0.40
Location	0.44	0.61	0.51	0.33	0.34	0.36
Cuisine	0.49	0.55	0.52	0.34	0.43	0.38
Dish	0.50	0.65	0.56	0.40	0.43	0.41
Amenity	0.51	0.68	0.58	0.40	0.43	0.42
Hours	0.46	0.70	0.58	0.38	0.39	0.38
Price	0.49	0.56	0.52	0.33	0.39	0.36

**Table 3.2** Semi-supervised entity recognition - external evaluation

Baseline	P	R	F1
MOVIE (S)	0.45	0.49	0.47
LSTM-CRF (S)	0.82	0.74	0.78
BERT-NER (S)	0.96	0.94	0.95
MOD-ENT (U)	0.35	0.55	0.43
BRAND-PROD (U)	0.32	0.46	0.38
<b>BOTTOMUP (Semi)</b>	<b>0.36</b>	<b>0.56</b>	<b>0.44</b>

Both cannot be generalized across domains, indicating the effectiveness in our approach. The MOD-ENT baseline performs almost as well as the proposed model, possibly because the movie dataset agrees with the ordering assumptions made by the grammar proposed by Elsner et al. [2009]. This is a reasonable assumption considering that the grammar used by Elsner et al. is proposed for search queries, and the movie dataset also contains search queries which would have similar sentence structure, albeit a different domain of queries. In addition, the grammar has explicitly marked pronouns.

Considering the supervised approaches, the MOVIE baseline performs better than the approach in terms of precision, which is expected because the rules required for the grammar are obtained directly from the dataset. The supervised approaches, LSTM-CRF, and BERT-NER perform much better than other approaches as the models are trained as a Sequence Labeling task on the Restaurant dataset and it learns both the semantic and syntactic structure of the sentences. BERT-NER performs better than LSTM-CRF possibly because the BERT architecture does not have typical limitations of LSTM like vanishing or exploding gradients. The pre-trained BERT-NER model uses attention mechanism that is more suitable to learn the complex patterns than the LSTM-CRF model.

### 3.5 Limitations and Future Work

This work proposes a semi-supervised approach based on adaptor grammar to extract named entities from domain-specific datasets. This is an extremely important problem in enterprise settings where documents

require annotated labels.

Evaluation results show that the semi-supervised approach performs better than unsupervised baseline approaches that rely on manual effort or heuristic methods in grammar creation for adaptor grammar. For the future work, the evaluation can be expanded to multiple domains. Also, there is a scope to generate the adaptor grammar with rules trained on noun phrases with multiple entities. This would address the current limitation of the approach to handle multiple entities in a single phrase.

## CHAPTER

# 4

# PARAMETRIC NEURAL ADAPTOR GRAMMAR

## 4.1 Research Problem

For this work, the parametric neural adaptor grammar is applied and evaluated on two entity extraction problems, academic concept extraction and named entity recognition, as defined below.

### 4.1.1 Academic Concept Extraction

Academic Concept Extraction is defined as identifying the academic concepts like applications, methods etc., from the titles of academic work. The academic titles usually contain the name of a technique that can be used for a specific application. These techniques and applications usually have a concept and occasionally modifiers to the concepts. The titles are split into noun phrases with words  $w_1, \dots, w_n$ . The concept extraction problem associates the entity titles  $l_i$ , to each word  $w_i$ , such that  $l_i \in \{\text{Modifier, Concept}\}$ . Krishnan et al. [2017] explored the use of Pitman Yor Adaptor Grammar with predefined grammar rules based on the domain knowledge. This work applies the same grammar rules shown in table 4.1 with the parametric neural adaptor grammar for a fair comparison.

### 4.1.2 Named Entity Recognition

For named entity recognition problem, the noun phrases from the MUC-7 dataset which contain names or locations or organisations are isolated. The MUC-7 dataset contains sentences from news articles annotated with the named entities and part of speech tags. The task aims to label each phrase with the

**Table 4.1** Grammar rules to build the adaptor grammar for academic concept extraction

Phrase	→	Modifier Concept
Phrase	→	Concept Modifier
Phrase	→	Modifier Concept Modifier
Phrase	→	Modifier
Phrase	→	Concept
Concept	→	Words
Modifier	→	Words

entity types, Person, Organisation or Location. The grammar rules presented in Elsner et al. [2009] as shown below in the table 4.2 are used for this task for an evaluation of the parametric neural adaptor grammar.

**Table 4.2** Grammar rules to build adaptor grammar for named entity recognition

Phrase	→	Person   Organisation   Location
Person	→	(NE <sup>0</sup> ) (NE <sub>1</sub> ) (NE <sub>2</sub> ) (NE <sub>3</sub> ) (NE <sub>4</sub> )
Location	→	(NE <sup>0</sup> ) (NE <sub>1</sub> ) (NE <sub>2</sub> ) (NE <sub>3</sub> ) (NE <sub>4</sub> )
Organisation	→	(NE <sup>0</sup> ) (NE <sub>1</sub> ) (NE <sub>2</sub> ) (NE <sub>3</sub> ) (NE <sub>4</sub> )
NE <sup>x</sup>	→	Words

## 4.2 PUER: Parametric Neural Adaptor Grammar

This work introduces a neural variant of adaptor grammar, PUER, short for Parametric Unsupervised Entity Recognition. PUER performs unsupervised entity recognition of the words in noun phrases in three steps. In the first step, the posterior probability of generating the noun phrase is approximated using an architecture built on recurrent neural networks (RNN) and the inside-outside probabilities. The posterior probability is maximized by learning the parameters of the adaptor grammar. Once these parameters are learnt, the test sentences are decoded by approximating the posterior probabilities for each parse tree and using Viterbi algorithm to find the parse tree with maximum probability.

PUER is a neural variant of the adaptor grammar. PUER follows a recurrent neural network (RNN) based auto-encoder structure [Jang et al., 2019], with the encoder constructed using gated recurrent units (GRU) and the decoder constructed using RNNs. The adaptor grammar is encoded in a neural representation using  $\alpha$  the inside probabilities of a non-terminal  $A$  [Baker, 1975].

PUER is defined as  $\langle N, W, S, P, R(N), D, C \rangle$  where  $N$  is the set of non-terminals,  $W$  is the set of terminals,  $S$  is the start state such that  $S \in N$ ,  $P$  refers to pre-terminals, the subset of the non-terminals which expand to terminals.  $R(N)$  is the set of rules that parametric neural adaptor grammar uses,  $D$  is the set of adapted non-terminals and  $C_A$  is the adaptor function for the adapted non-terminal  $A$ . The grammar rules used for parametric neural adaptor grammar are converted to Chomsky normal form, to convert

the rules to either of the forms,  $A \rightarrow BC$  or  $A \rightarrow w$  where  $w$  is a terminal, as shown in tables 4.3 and 4.4. Adopting the CNF form restricts the number of non-terminals to be processed for each rule and reduces the complexity of the probability estimation.

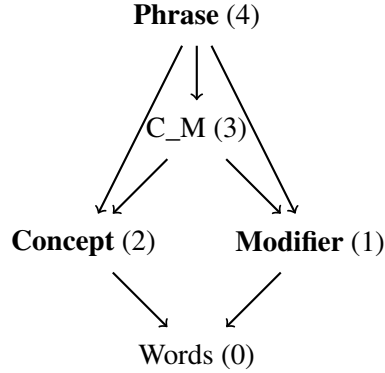
**Table 4.3** Grammar rules for concept extraction task in Chomsky form

Phrase	→	Modifier_Concept
Phrase	→	Concept Modifier
Phrase	→	Modifier_Concept Modifier
Phrase	→	Modifier
Modifier_Concept	→	Modifier Concept
Modifier	→	Words
Concept	→	Words

**Table 4.4** Grammar rules for named entity recognition task in Chomsky form

Phrase	→	Person
Phrase	→	Location
Phrase	→	Organization
NE1	→	NE0
NE2	→	NE0 NE1
NE3	→	NE0 NE2
NE4	→	NE0 NE3
NE5	→	NE0 NE4
Person	→	NE1
Person	→	NE2
Person	→	NE3
Person	→	NE4
Person	→	NE5
Location	→	NE1
Location	→	NE2
Location	→	NE3
Location	→	NE4
Location	→	NE5
Organization	→	NE1
Organization	→	NE2
Organization	→	NE3
Organization	→	NE4
Organization	→	NE5
NE0	→	Words

Each of the terminals and the non-terminals is associated with an embedding vector of dimension



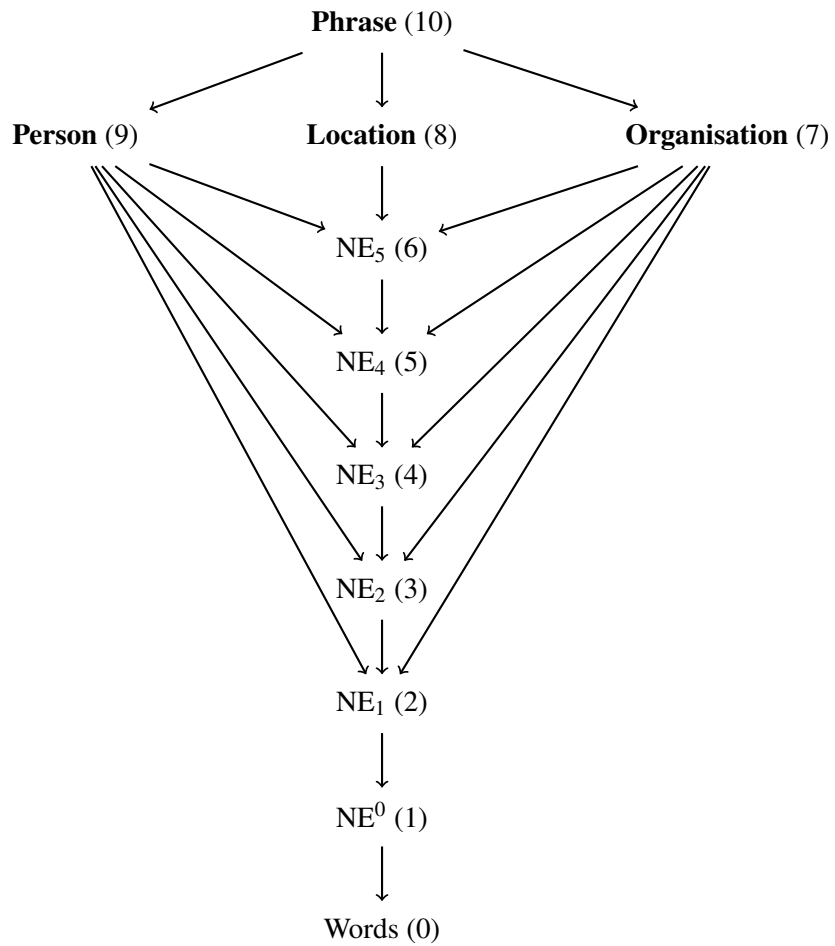
**Figure 4.1** Dependency graph in non-terminals for concept extraction grammar. The index for each non-terminal in the topological sort is given the number in the brackets next to them

*d.* Non-terminals (N) for the academic concept extraction forms the set {Phrase, Modifier, Concept, C\_M, Words} and the adapted non-terminals form {Phrase, Modifier, Concept}. For the named entity recognition task, the set of non-terminals is {Phrase, NE<sup>0</sup>, NE<sup>1</sup>, NE<sup>2</sup>, NE<sup>3</sup>, NE<sup>4</sup>, NE<sup>5</sup>, Person, Location, Organisation} and the set of adapted non-terminals is {Phrase, Location, Organisation, Person}. All the words in the vocabulary form the set of terminals for both the tasks. PUER uses only one non-terminal Words which can expand to an arbitrary length of terminals. For a phrase of length  $l$  with tokens  $w_1, w_2, \dots, w_l$ ,  $\alpha(A, i, j)$  represents the probability of generating the part of the phrase  $w_i, \dots, w_j$  with parse trees rooted at A.  $\alpha$  is a  $N \times l \times l \times d$  dimensional matrices where N is the number of non-terminals and  $l$  is the length of the phrase.

The embedding vectors for the non-terminals are randomly initialized and the embedding vectors for the terminals are the pretrained word embeddings, such that  $e_{w_i}$  represents the word embedding vector of the word  $w_i$ . These parameters are learned along with the model parameters. The parametric neural adaptor grammar is used as an encoder to generate a hidden representation of the phrase and the hidden representation is used to decode the original phrase. The model follows an auto-encoder structure where the encoder is a parametric neural adaptor grammar and a regular RNN decoder.

The non-terminals in the adaptor grammar rules are sorted in the topological order as per the dependencies. This order maintains the recursion that is to be followed in computation of  $\alpha$ , such that for each rule  $A \rightarrow BC$ ,  $\alpha(B, i, j)$  and  $\alpha(C, i, j)$  are computed before  $\alpha(A, i, j)$ . The recursion to the same non-terminal (like in the rule for “Words”) is handled by assuming that the non-terminal on the left expands to the same non-terminal on the right only if the span of the right non-terminal is less than or equal to the span of the left non-terminal. The dependency graph for the non-terminals along with a possible topological ordering of the non-terminals is shown in tables 4.1 and 4.2.

The matrices  $\alpha$  are calculated recursively, bottom up starting from the terminals according to the



**Figure 4.2** Dependency graph in non-terminals for entity recognition grammar. The index for each non-terminal in the topological sort is given the number in the brackets next to them. Few arrows along the central vertical lines are omitted for interpretability.

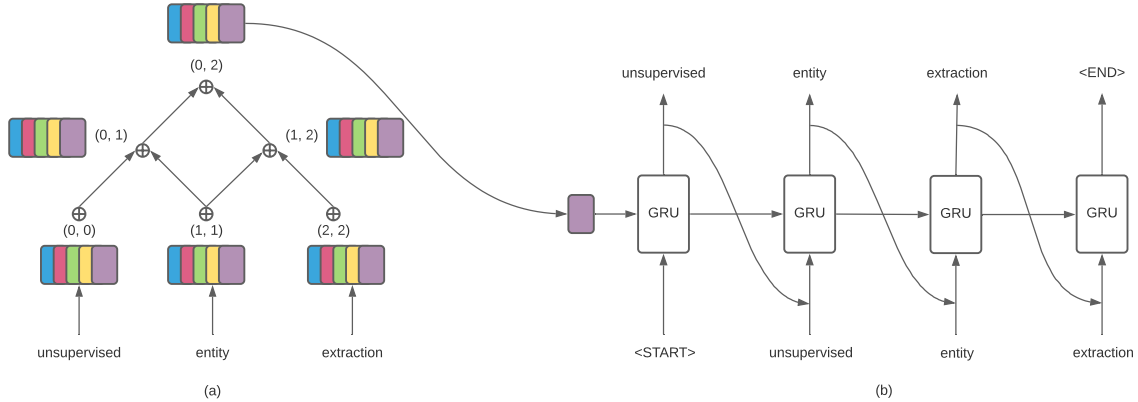
following equations.

$$\alpha(P, i, i) = f_P(e_P, e_{w_i}) \quad (4.1)$$

$$\alpha(P, i, j) = f_P(e_P, f_P(e_{w_i}, \alpha(e_P, i + 1, j))) \quad (4.2)$$

$$\alpha(A, i, j) = \sum_{k=i+1}^j f_A(e_A, f_A(\alpha(e_B, i, k - 1), \alpha(e_C, k, j))) \quad (4.3)$$

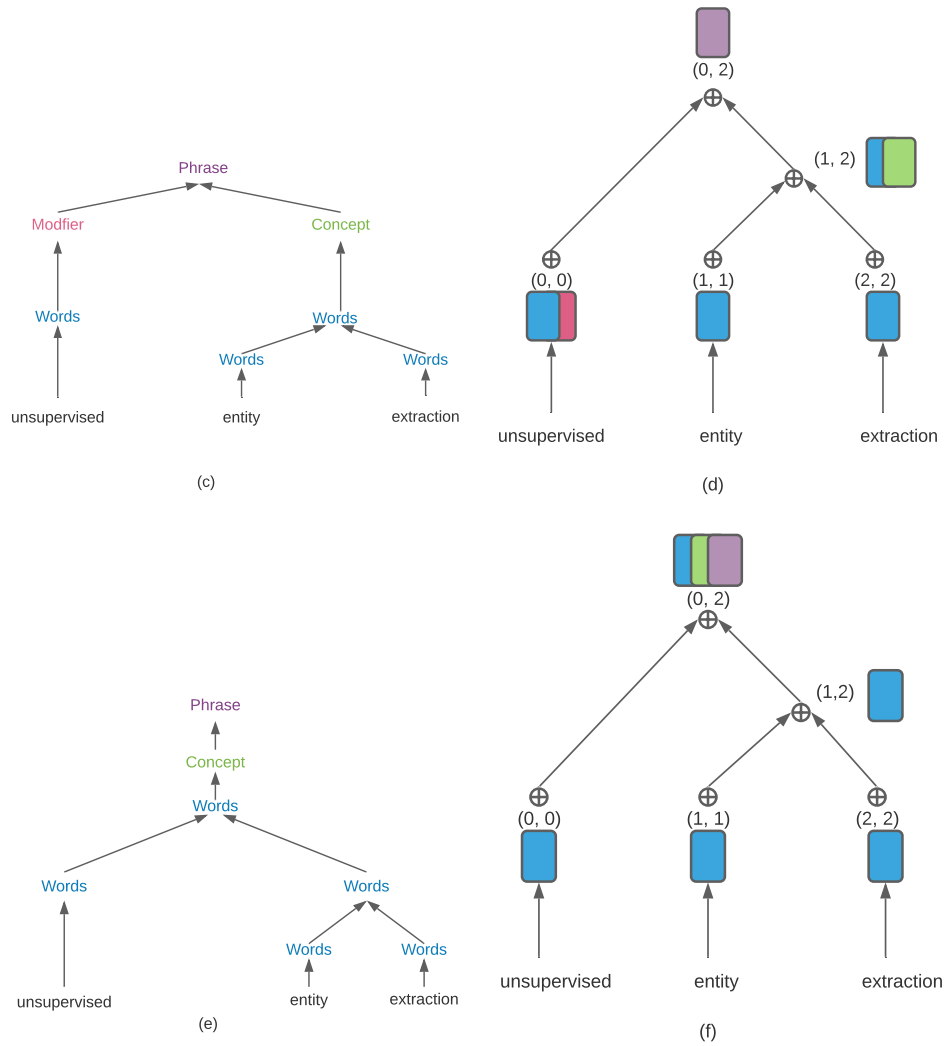
Here  $f$  is the RNN architecture used and  $P$  is the pre-terminal.  $\alpha$  computation is started from the pre-terminal  $P$  and move along the topological sort order. For a pre-terminal  $P$  and a span length of 1,  $\alpha(P, i, i)$  is a function of the word  $w_i$  and the embedding of the pre-terminal  $e_P$ . For the span length greater than 1,  $\alpha(P, i, j)$  is computed recursively as a function of the embedding of  $P$ , embedding of the word  $w_i$ ,  $e_P$ , and  $\alpha(e_P, i + 1, j)$ . For the other non-terminals like  $A$ , the contributions from all the rules  $A$  expands to recursively are added. The instances of RNNs are used, once for adapted non-terminals and the other for non-adapted non-terminals.  $f_A$  in the above equation refers to the RNN for adapted non-terminals if  $A$  is adapted. If  $A$  is non-adapted,  $f_A$  refers to the RNN for non-adapted non-terminals. The algorithm is presented in Algorithm-3



**Figure 4.3** Parametric neural adaptor grammar. Figure (a) shows the encoder and the progress of  $\alpha$  values for the concept extraction problem on the phrase “unsupervised entity extraction”. The non-terminals are represented by different colors in the topological order: blue - Words, pink - Modifier, green - Concept, yellow - C\_M, purple - Phrase. Given a start and end index, the  $\alpha$  values are calculated in this order (from left to right). Figure (b) shows the decoder that produces the same output as the input phrase.

## 4.2.1 Training

The  $\alpha$  values are calculated recursively according to Algorithm 3 based on Equations 4.1, 4.2, and 4.3.  $\alpha(S, 1, n)$  is then passed through a feed-forward neural network  $f_1$  and a sigmoid layer, which yields the predicted probability of the phrase  $w_1, \dots, w_n$ .  $\alpha(A, i, j)$  works as a proxy for the probability of



**Figure 4.4** Parametric neural adaptor grammar. Figures (c) and (e) show the two of the possible parse trees for the phrase “unsupervised entity extraction”. Figures (d) and (e) show the computations of  $\alpha$  affected by the parse trees in (c) and (f) respectively

---

**Algorithm 3** Parametric neural adaptor grammar - inside probabilities

---

```
1: procedure ALPHA
2:   Initialize  $N \times n \times n$  dimensional matrix  $\alpha$ 
3:   for  $l \in 1, \dots, n$  do
4:     for  $i \in 0, \dots, n - 1$  do
5:       for  $A \in N$  in the order of topological sort do
6:         if  $A == P$  and  $l == 1$  then
7:           if  $A$  is adapted then
8:              $\alpha(A, i, i) = f_{AD}(e_A, e_{w_i})$ 
9:           else
10:             $\alpha(A, i, i) = f_{NA}(e_A, e_{w_i})$ 
11:          else if  $A == P$  then
12:            if  $A$  is adapted then
13:               $\alpha(A, i, j) = f_{AD}(e_A, f_{AD}(e_{w_i}, \alpha(e_A, i + 1, j)))$ 
14:            else
15:               $\alpha(A, i, j) = f_{NA}(e_A, f_{NA}(e_{w_i}, \alpha(e_A, i + 1, j)))$ 
16:          else
17:            for  $R_{A \rightarrow BC} \in R_A$  do
18:              if  $A$  is adapted then
19:                 $\alpha(A, i, j) = \sum_{k=i+1}^j f_{AD}(e_A, f_{AD}(\alpha(e_B, i, k - 1), \alpha(w_C, k, j)))$ 
20:              else
21:                 $\alpha(A, i, j) = \sum_{k=i+1}^j f_{NA}(e_A, f_{NA}(\alpha(e_B, i, k - 1), \alpha(e_C, k, j)))$ 
22:   return  $\alpha$ 
```

---

generating the phrase  $w_i, \dots, w_j$  from the terminal  $A$ . This computation is needed so that we can perform a Viterbi decoding to find the parse tree with the highest probability. Also,  $\alpha(S, 1, n)$  value is fed into the RNN decoder as its hidden state. The loss as shown in Equation 4.4 is computed as the sum of two factors, namely, (1) the predicted probability of the phrase [Kim et al., 2019] subtracted from 1 and (2) the reconstruction loss [Drozdo et al., 2019] of the decoder:

$$\mathcal{L} = (1 - \text{sigmoid}(f_1(\alpha(S, i, n)))) - \sum_{i=1}^n \log(p_i(w_i)) \quad (4.4)$$

where  $p_i(w_i)$  is the probability of generating  $w_i$  in the  $i^{\text{th}}$  step of the decoding process. This loss is minimized by an Adam optimizer, as explained in Section 4.3.

## 4.2.2 Inference

The inference is achieved by retrieving the parse tree with the maximum predicted probability computed with the Viterbi algorithm. Starting from  $\alpha(S, 1, n)$ , the Viterbi algorithm recursively finds the rule that each non-terminal  $A$  should expand to from  $\alpha(A, i, j)$  to generate the parse tree of the highest probability. The predicted probability of the parse tree rooted at  $A$  covering the span  $w_i, \dots, w_j$  is calculated as follows.

$$p(A, i, j) = \text{sigmoid}(f_1(\alpha(A, i, j))) \quad (4.5)$$

## 4.3 Experiments

### 4.3.1 Datasets

All the datasets used in this work are restricted to English language. The experiments are conducted by focusing only on the entries from English. DBLP [Kariv and Pollock, 2018; Krishnan et al., 2017] and ACL [Radev et al., 2013; Team, 2020] datasets are used for the academic concept extraction task. ACL and DBLP datasets contains the metadata for 63736 and 8211940 papers respectively. The titles of these papers are preprocessed to include only the titles containing the gold standard concepts as proposed by Krishnan et al. [2017]. 15078 noun phrases from the ACL dataset and 526303 noun phrases from the DBLP dataset containing the concepts are extracted from these titles and used for the training purposes. For both the datasets, the training, validation and the testing datasets are split in the 80-10-10 ratio.

MUC-7 dataset<sup>1</sup>, as suggested by Elsner et al. [2009] is adopted for the named-entity recognition task. The dataset contains sentences from news articles annotated with their entity titles and parts-of-speech (POS) tags. 29110 noun phrases containing person, location and organisation tags along with their annotated labels are extracted with spaCy noun phrase chunker [Honnibal and Montani, 2017]. The noun

<sup>1</sup><https://catalog.ldc.upenn.edu/LDC2001T02>

phrases are split into training, evaluation and testing datasets with a 80-10-10 ratio. The average length of phrases in the ACL, DBLP, and the MUC datasets are 6.2, 5.8, and 2.6, respectively.

These tasks and datasets are specifically chosen because the earlier approaches of adaptor grammar have been applied and evaluated for entity recognition on these datasets. The same grammar rules proposed in these approaches are used for a fair comparison of performances of Pitman-Yor adaptor grammar and the parametric neural adaptor grammar approaches.

### 4.3.2 Baselines

The following baselines are used to compare the performance of the parametric neural adaptor grammar with other adaptor grammar and other neural grammar based approaches. The confusion matrix for the concepts is obtained and the precision, recall and F1 score of the methods are compared against the parametric adaptor grammar for each label and the whole dataset.

**Adaptor Grammar with Variational inference (AG+VI).** A traditional Pitman-Yor adaptor grammar [Cohen et al., 2010] that uses expectation maximization (EM) to approximate the posterior and maximized using variational inference. This method is one of the early inference techniques used with Adaptor Grammar.

**Adaptor Grammar with Hybrid inference (AG+HI).** The method proposed in Zhai et al. [2014] is different from the earlier AG+VI method in the inference algorithm used. It uses a combination of the variational inference with MCMC sampling.

**Neural PCFG.** Kim et al. [2019] learns the probabilities of PCFG from sentences using distributed representations in embedding space.

**Compound PCFG.** Kim et al. [2019] is an extension to the above Neural PCFG approach and introduces a per-sentence latent vector.

### 4.3.3 Time and Space Complexity

For each span, the model calculates the contribution of each rule for each partition. Therefore, if the rule is of the form  $A \rightarrow BC$ , the calculation of contribution of each partition takes  $O(n)$  time in the worst case, where  $n$  is the length of the input in words, and each span  $i, \dots, j$  takes a time of  $O(mn)$ , where  $m$  is the number of rules. Since the number of spans is  $n^2$ , the overall complexity of calculating the  $\alpha$  values is  $O(mn^3)$ . The decoder to the autoencoder model takes  $O(n)$  time because each output token is sequentially produced. Once the  $\alpha$  values are calculated, the highest probability parse tree should be obtained by Viterbi algorithm. For a particular non-terminal and a span, the algorithm takes  $O(mn)$  time to find the subtree with the highest probability. Since this calculation is performed for each expanded node, the Viterbi algorithm takes  $O(n^2)$  time. Note that the Viterbi algorithm is not performed during training. In both training and running, the calculation of the  $\alpha$  values takes the most time. The complexity of the algorithm for the overall algorithm is  $O(mn^3)$ , which explains the long training and run time taken for the model despite its fewer model parameters.

#### 4.3.4 Experimental Design

For this work, two RNN-based architectures are considered: 1) gated recurrent units (GRU) and 2) long-short term memory (LSTM). The hidden size of the RNN used is the same as the embedding size. The final probability layer is a feed forward layer with the size of the input layer as the embedding size and the output size of 1. The GRU model takes 8 hours for ACL dataset, 42 hours for DBLP dataset and 15 hours for the MUC dataset with a single-machine 8GB RAM and 4 core Nvidia GeForce RTX 2080 GPU. The LSTM model takes 9 hours for ACL, 54 hours for DBLP dataset and 18 hours for MUC-7 dataset.

#### 4.3.5 Hyper-parameters

Word embeddings of smaller dimension, 50 ( $d$ ) is used to reduce the computational complexity. The model of which the results are presented is trained with a batch size of 16 with 20 epochs. An Adam optimizer with a learning rate of 0.0001 is used with  $\beta_1 = 0.75$  and  $\beta_2 = 0.999$ . The hyper-parameter search is conducted with multiple combinations of dimension size and the learning rate. A grid analysis with dimension sizes of 50, 100 and 300, and learning rates of 0.005, 0.003, 0.001, 0.03 and 0.05. The results are shown for the best performing combination which is a dimension size of 50 and the learning rate of 0.0001. Higher learning rates showed erratic loss in the training and validation loss and lower learning rates were too slow to train and did not reach the expected loss that was shown by the learning rate of 0.0001. The models with embeddings of higher dimension had a greater computational complexity and was over-fitting the data.

Two hyperparameters, the dimension size for embeddings (same as the hidden size of the RNN component) and the learning rate are tuned for experiments. In each experiment, the model is trained for 20 epochs based on the observation that the model converges at around the 18<sup>th</sup> epoch. A batch size of 16 is chosen based on the experiments. Adam optimizer with  $\beta_1 = 0.75$  and  $\beta_2 = 0.999$  is used as an optimizer with different learning rates. Learning rate is set as proposed by Drozdov et al. [2019] and adjusted by reducing when the validation loss increased. A grid analysis with dimension sizes of 50, 100 and 300 and learning rates of 0.005, 0.003, 0.001, 0.03 and 0.05 is performed. The experiments show that the model with a dimension of 50 and learning rate of 0.001 achieves the best performance. Embeddings of larger dimensions yield greater computational complexity and cause the model to overfit. Higher learning rates cause erratic loss in the training and validation, whereas lower ones slow down the training process and fail to achieve the expected loss.

#### 4.3.6 Evaluation Measure

The standard metrics defined for the entity recognition tasks are precision, recall and f1 score. For each label defined, confusion matrix for the test data is calculated in terms of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Label-wise precision, recall and F1 score

are calculated according to the following formulae.

$$\begin{aligned} \text{Precision } (P_{\text{label}}) &= \frac{\text{TP}}{\text{TP}+\text{FP}} \\ \text{Recall } (R_{\text{label}}) &= \frac{\text{TP}}{\text{TP}+\text{FN}} \\ \text{F1 Score } (F1_{\text{label}}) &= \frac{2P_{\text{label}}R_{\text{label}}}{P_{\text{label}} + R_{\text{label}}} \end{aligned}$$

For each dataset, the consolidated precision, recall and F1 score is computed by the weighted average of precision, recall and F1 score, weighed by the support for each label ( $n_{\text{label}}$ ), the number of times words annotated the label appear in the test set.

$$\begin{aligned} \text{Precision } (P) &= \frac{\sum_{\text{label}} n_{\text{label}} * P_{\text{label}}}{\sum_{\text{label}} n_{\text{label}}} \\ \text{Recall } (R) &= \frac{\sum_{\text{label}} n_{\text{label}} * R_{\text{label}}}{\sum_{\text{label}} n_{\text{label}}} \\ \text{F1 Score } (F1) &= \frac{2PR}{P+R} \end{aligned}$$

### 4.3.7 Number of Parameters

The model with a dimension size of 50 contains 390,531 and 390,831 trainable parameters including the parameters for the embeddings vectors for concept extraction and named entity recognition tasks, respectively. The difference in the number of parameters is because of the difference in the numbers of non-terminals present in both the tasks. The concept extraction task considers only five non-terminals whereas the named entity recognition task involves eleven non-terminals.

## 4.4 Results and Analysis

The table 4.5 shows the consolidated Precision, Recall and F1 scores for all the labels identified using PUER compared against the other grammar induction baselines. PUER has outperformed all the other baselines by a significant margin. It is worth noting that this method provides a way of injecting domain knowledge into the model unlike the other neural based approaches like Neural and Compound PCFG [Kim et al., 2019]. PUER outperforms even the other adaptor grammar methods with the same domain information. PUER learns the probabilities of the different parse trees more accurately, compared to the other methods for the datasets, because it takes advantage of the semantic information of word embeddings and the hidden representations. Table 4.5 shows the the performance is very comparable for LSTM and GRU. Table 4.6 shows the consolidated performance of PUER LSTM and GRU on validation data. Since, the performance is comparable, length of phrases is shorter and GRU is faster, PUER-GRU

**Table 4.5** Parametric neural adaptor grammar - evaluation against baselines on test set

Dataset	Concept Extraction						Entity Recognition		
	ACL			DBLP			MUC		
Method	P	R	F1	P	R	F1	P	R	F1
AG + VI Cohen et al. [2010]	64.5	69.2	66.7	71.6	72.9	72.2	48.6	43.2	45.7
AG + HI Zhai et al. [2014]	80.6	73.1	76.7	69.9	73.9	71.8	52.4	56.9	54.5
N-PCFG Kim et al. [2019]	68.7	69.1	68.9	64.9	68.3	66.5	55.7	54.8	55.2
C-PCFG Kim et al. [2019]	72.6	69.8	71.2	68.9	72.5	70.6	58.3	60.4	59.3
<b>PUER-GRU</b>	<b>85.4</b>	82.9	<b>84.1</b>	76.9	78.4	77.6	<b>65.8</b>	64.2	64.9
<b>PUER-LSTM</b>	82.6	<b>83.1</b>	82.8	<b>77.8</b>	<b>78.6</b>	<b>78.1</b>	<b>65.8</b>	<b>64.9</b>	<b>65.3</b>

**Table 4.6** Parametric neural adaptor grammar - evaluation against baselines on validation set

Dataset	Concept Extraction						Entity Recognition		
	ACL			DBLP			MUC		
Method	P	R	F1	P	R	F1	P	R	F1
AG + VI Cohen et al. [2010]	69.8	67.6	68.7	72.8	73.2	73.0	49.3	45.2	47.2
AG + HI Zhai et al. [2014]	76.9	75.3	76.1	70.6	74.3	72.4	54.9	57.6	56.2
N-PCFG Kim et al. [2019]	67.2	69.8	68.5	65.7	67.4	66.5	56.4	54.1	55.2
C-PCFG Kim et al. [2019]	74.1	70.4	72.2	70.5	74.1	72.2	60.7	61.8	61.2
<b>PUER-GRU</b>	<b>86.2</b>	<b>84.2</b>	<b>85.2</b>	<b>78.5</b>	<b>79.3</b>	<b>78.9</b>	64.4	63.7	64.0
<b>PUER-LSTM</b>	83.4	84.1	83.9	77.4	76.8	77.1	<b>65.6</b>	<b>64.2</b>	<b>64.9</b>

is deemed as the main model.

PUER yields better results for the ACL dataset over the DBLP dataset, for the concept extraction task. The difference might be due to the fact that the DBLP dataset is more diverse as it contains titles of academic work from all the areas of Computer Science as opposed to ACL which is restricted to Computational Linguistics and is therefore less sufficiently represented by PUER. PUER notably outperforms the baseline methods on the entity recognition tasks. Similar performance comparisons of the results can be observed on the validation datasets as well in the table 4.6. However, PUER performs relatively poor compared to the concept extraction task, possibly due to the following reasons:

- The number of labels is higher than that of the concept extraction task
- The noun phrases in the MUC dataset are often proper nouns, as they are names of persons, locations, or organizations, which may not be fully represented by the word embeddings

Tables 4.8 and 4.7 show the label-wise performance of PUER compared to the other baselines for concept and modifier labels of the academic concept extraction task respectively. From the tables it can be seen that the performance for the modifier label is better than that of the concept label on both ACL and the DBLP datasets. Also, the effect of the pretrained word embeddings is higher on the entities of the modifier label compared to the ACL and the DBLP datasets. The table 4.11 shows the performance of the model on entity recognition on MUC dataset for each of the three labels: person, organisation and location. The performance of the the model is better on the organisation compared to the person and

**Table 4.7** Internal Evaluation: Academic Concept Extraction: Modifier - Test

<b>Dataset</b>	<b>ACL</b>			<b>DBLP</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
AG + VI Cohen et al. [2010]	63.9	69.3	66.5	73.6	75.2	74.4
AG + HI Zhai et al. [2014]	81.1	71.1	75.8	71.9	75.8	73.8
N-PCFG Kim et al. [2019]	67.3	70.5	68.9	63.5	70.5	66.8
C-PCFG Kim et al. [2019]	76.3	69.4	72.7	68.6	75.1	71.7
<b>PUER-LSTM</b>	82.3	85.1	83.7	<b>80.5</b>	<b>79.4</b>	<b>80.0</b>
<b>PUER-GRU</b>	<b>83.8</b>	<b>85.5</b>	<b>84.6</b>	79.2	79.1	79.1

**Table 4.8** Internal Evaluation: Academic Concept Extraction: Concept - Test

<b>Dataset</b>	<b>ACL</b>			<b>DBLP</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
AG + VI Cohen et al. [2010]	64.9	69.1	66.9	70.3	71.4	70.8
AG + HI Zhai et al. [2014]	80.3	74.4	77.2	68.6	72.7	70.6
N-PCFG Kim et al. [2019]	69.6	68.2	68.9	65.8	66.9	66.3
C-PCFG Kim et al. [2019]	70.2	70.0	70.1	69.1	70.8	69.9
<b>PUER-LSTM</b>	82.9	80.8	81.8	74.7	77.6	76.1
<b>PUER-GRU</b>	<b>86.4</b>	<b>81.2</b>	<b>83.7</b>	<b>75.4</b>	<b>77.9</b>	<b>76.6</b>

location as shown in the table possibly because the organisation names usually include signals like Corp, Inc, etc, which makes their labeling easier. The label-wise performance of PUER on validation data is shown in tables 4.8, 4.7 and 4.11.

#### 4.4.1 Dataset sizes

Table 4.13 shows the performance of PUER-GRU on the two tasks with different fractions of datasets considered for training. For the concept extraction task, PUER yields the F-1 scores of 36.8% and 53.9% for the ACL and DBLP datasets, respectively, if using 10% of the dataset for training. Compared to the performance of PUER trained on 100% of the datasets, PUER’s performance on the ACL dataset improves considerably, possibly because the ACL dataset is much smaller in size compared to the DBLP dataset. The F-1 score of PUER on the MUC dataset increases from 31.7% with 10% of the dataset to 65.2% with 80% of the dataset, and decreases slightly to 64.9% if using 100% of the dataset. The decrease in performance is likely due to overfitting.

#### 4.4.2 Ablation Study

Table 4.14 shows the consolidated precision, recall and F1 score of the model without the RNN decoder and pretrained word embeddings, respectively. As we mentioned above, the MUC dataset contains a large number of proper nouns whose information might not be fully captured by the word embeddings. The removal of pretrained embeddings imposes a slight decrease in performance, which consolidates our analysis. On the other hand, the decrease in performance is significant on the concept extraction tasks of

**Table 4.9** Internal Evaluation: Academic Concept Extraction: Modifier - Validation

<b>Dataset</b>	<b>ACL</b>			<b>DBLP</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
AG + VI Cohen et al. [2010]	73.7	66.2	69.7	73.4	75.4	74.4
AG + HI Zhai et al. [2014]	73.2	75.0	68.6	71.6	75.0	73.2
N-PCFG Kim et al. [2019]	63.0	69.5	66.1	65.3	67.4	66.3
C-PCFG Kim et al. [2019]	77.8	68.2	72.7	70.4	76.0	73.1
<b>PUER-LSTM</b>	80.4	<b>84.7</b>	82.5	77.9	75.5	76.7
<b>PUER-GRU</b>	<b>84.8</b>	84.2	<b>84.5</b>	<b>79.7</b>	<b>79.8</b>	<b>79.7</b>

**Table 4.10** Internal Evaluation: Academic Concept Extraction: Concept - Validation

<b>Dataset</b>	<b>ACL</b>			<b>DBLP</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
AG + VI Cohen et al. [2010]	65.3	69.2	67.2	72.1	70.6	71.3
AG + HI Zhai et al. [2014]	81.2	75.6	78.3	69.4	73.5	71.4
N-PCFG Kim et al. [2019]	72.1	70.0	71.1	66.2	67.3	66.7
C-PCFG Kim et al. [2019]	69.8	72.9	71.3	70.6	71.9	71.2
<b>PUER-LSTM</b>	86.9	83.4	85.1	76.8	78.3	77.5
<b>PUER-GRU</b>	<b>87.8</b>	<b>84.2</b>	<b>86.0</b>	<b>77.1</b>	<b>78.7</b>	<b>77.9</b>

ACL and DBLP datasets possibly because the semantic information in the word embeddings is useful to label the data. The RNN decoder proves useful as an indirect supervision which forces the encoder to learn more information about the input phrases. Tables 4.16 and 4.17 show the label-wise performance of the model without RNN decoder and (pretrained) embeddings.

### 4.4.3 Qualitative Analysis

Two major classes of errors that PUER makes are identified by analysis on the test results:

- Some modifiers (possibly adjectives) are misidentified as concepts
- Some concepts (possibly nouns) are misidentified as modifiers

For example, PUER misidentified the modifier “independent” in the phrase “domain independent language modelling” as a concept. Meanwhile, the concept “information” in the phrase “natural language information retrieval,” is misidentified as a modifier. The main type of errors from the named entity recognition task come from examples with ambiguous labels. For example, the dataset contains some cases where the phrase “Japan” is labeled as Location while PUER identifies it as Organization, and other cases where “Japan” is labeled as Organization and PUER identifies it as Location. The other major class of errors is from phrases that contain words not present in the considered vocabulary.

**Table 4.11** Named entity recognition - internal evaluation - test

<b>Dataset</b>	<b>Person</b>			<b>Location</b>			<b>Organisation</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
AG + VI Cohen et al. [2010]	47.4	44.6	45.9	49.2	43.0	45.9	49.4	41.8	45.3
AG + HI Zhai et al. [2014]	52.9	56.4	54.6	51.5	57.1	54.1	52.5	57.3	54.8
N-PCFG Kim et al. [2019]	54.8	55.0	54.9	55.9	54.2	55.0	56.5	55.0	55.7
C-PCFG Kim et al. [2019]	58.1	60.6	59.3	57.8	60.1	58.9	58.9	60.4	59.6
<b>PUER-LSTM</b>	64.5	<b>64.3</b>	64.4	<b>66.1</b>	<b>64.2</b>	<b>65.1</b>	66.6	<b>66.3</b>	<b>66.4</b>
<b>PUER-GRU</b>	<b>66.3</b>	63.8	<b>65.0</b>	65.2	64.1	64.6	<b>66.8</b>	64.7	65.7

**Table 4.12** Named entity recognition - internal evaluation - validation

<b>Dataset</b>	<b>Person</b>			<b>Location</b>			<b>Organisation</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
AG + VI Cohen et al. [2010]	48.3	46.1	47.2	49.7	45.8	47.7	49.8	43.8	46.6
AG + HI Zhai et al. [2014]	55.6	56.8	56.2	53.7	57.9	55.7	55.5	58.0	56.7
N-PCFG Kim et al. [2019]	57.1	53.8	55.4	55.8	54.2	55.0	56.4	54.3	55.3
C-PCFG Kim et al. [2019]	59.9	62.4	61.1	60.1	60.7	60.4	62.0	62.4	62.2
<b>PUER-LSTM</b>	<b>66.1</b>	<b>64.4</b>	<b>65.2</b>	63.8	<b>63.9</b>	<b>63.8</b>	<b>67.0</b>	<b>64.3</b>	<b>65.6</b>
<b>PUER-GRU</b>	64.2	63.9	64.0	<b>64.3</b>	63.2	63.7	64.7	64.0	64.3

#### 4.4.4 Threats to Validity

The experiments are conducted and tested with respect to only two natural language tasks of academic concept extraction and named entity recognition. The same grammar rules that have been used by the earlier approaches for the respective tasks are used to compare. While the parametric neural adaptor grammar presents an interesting encoder approach that could work for other tasks as well, the results are not guaranteed. Further, even for these specific tasks, different datasets could yield different results. Also please note that since the data used for training and the word embeddings are used for vocabulary in English, there is no guarantee as to this method could work for the same tasks of other languages.

## 4.5 Limitations and Future Work

Adaptor Grammar with distributed representations is a promising direction on work on the grammar induction problem. Grammar induction is an important problem to understand the natural language. Using word embeddings to learn the grammar is an interesting approach that is not explored extensively. The approach proposed in this paper is a proof of concept that an adaptor grammar with distributed representations works well for grammar induction problem. Though the approach works, it has major limitations. For example, the model takes too long to train because the Inside-Outside algorithm takes long to compute. For each sentence, calculating the  $\alpha$  values takes  $O(n^3)$  time per sentence. Even for inference, the dynamic programming Viterbi algorithm takes  $O(n^2)$  time to obtain the parse tree with the highest probability. With the presented experimentation setup, the model takes 42 hours for training. Future

**Table 4.13** Performance evaluation with data increase

Data	Concept Extraction						Entity Recognition		
	ACL			DBLP			MUC		
Method	P	R	F1	P	R	F1	P	R	F1
10%	32.4	42.6	36.8	52.3	55.6	53.9	28.9	35.1	31.7
20%	46.7	43.8	45.2	54.6	57.5	56.0	45.4	54.8	49.6
40%	63.2	56.3	59.6	67.4	61.9	64.5	56.8	60.2	58.4
80%	78.9	73.4	76.0	73.7	72.8	73.2	<b>66.1</b>	<b>64.3</b>	<b>65.2</b>
100%	<b>85.4</b>	<b>82.9</b>	<b>84.1</b>	<b>76.9</b>	<b>78.4</b>	<b>77.6</b>	65.8	64.2	64.9

**Table 4.14** Ablation study - evaluation against baselines

Dataset	Concept Extraction						Entity Recognition		
	ACL			DBLP			MUC		
Method	P	R	F1	P	R	F1	P	R	F1
<b>AG+DR</b>	<b>85.4</b>	<b>82.9</b>	<b>84.1</b>	<b>76.9</b>	<b>78.4</b>	<b>77.6</b>	<b>65.8</b>	<b>64.2</b>	<b>64.9</b>
- RNN decoder	80.2	78.8	79.4	74.2	73.2	73.6	64.6	62.8	63.7
- Pretrained embeddings	81.6	79.7	80.6	72.8	74.6	73.7	64.8	63.6	64.2

work can plan on using an optimized Expectation Maximization (EM) approach so that the model runs faster. Further, the recurrent neural networks can be substituted by attention-based or pretrained models for better understanding and faster computation. Introducing a non-parametric approach for Adaptor Grammar could be another branch of work where the embedding vectors for the Adaptor Grammar come from a probability distribution, instead of being initialized randomly. The non-parametric method has the advantage of being able to work with different probability distributions and better posterior maximization techniques. One other limitation of the method is that, it does not provide any concrete algorithm to come up with the initial grammar. Exploring methods to learn the grammar rules required for adaptor grammar is also an interesting idea for further research.

**Table 4.15** Ablation study - academic concept extraction - modifier

<b>Dataset</b>	<b>ACL</b>			<b>DBLP</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
<b>AG+DR</b>	83.8	85.5	84.6	79.2	79.1	79.1
- RNN decoder	79.7	79.8	79.7	74.9	74.4	74.6
- Pretrained embeddings	82.6	78.9	80.7	73.9	75.1	74.5

**Table 4.16** Ablation study - academic concept extraction - concept

<b>Dataset</b>	<b>ACL</b>			<b>DBLP</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
<b>AG+DR</b>	86.4	81.2	83.7	75.4	77.9	76.6
- RNN decoder	80.5	78.2	79.3	73.9	72.4	73.1
- Pretrained embeddings	80.9	80.1	80.5	72.1	74.3	73.2

**Table 4.17** Ablation study - named entity recognition

<b>Dataset</b>	<b>Person</b>			<b>Location</b>			<b>Organisation</b>		
<b>Method</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>	<b>P</b>	<b>R</b>	<b>F1</b>
<b>AG+DR</b>	66.3	63.8	65.0	65.2	64.1	64.6	66.8	64.7	65.7
- RNN decoder	63.9	62.4	63.1	64.8	62.0	63.4	67.0	63.8	65.4
- Pretrained embeddings	64.9	63.8	64.3	64.6	62.7	63.6	65.4	64.0	64.7

## CHAPTER

# 5

# CONCLUSION

## 5.1 Summary

This work explores adaptor grammar, a variant of probabilistic context-free grammar for unsupervised entity extraction. Supervised entity extraction has been widely studied and has reached near human performance. But supervised entity extraction needs huge annotated data for the models to train. Since, this is not very human-intensive process, there is a need for unsupervised entity extraction which does not require annotated data. Adaptor grammar is one of the grammar-based approaches that has been previously studied for unsupervised entity extraction. The previously proposed adaptor grammar suffers two major limitations. First, though requires less human intervention than annotating, defining grammar rules still demands human expertise. Second, Pitman Yor adaptor grammar requires priors to be set as it takes a Bayesian approach and requires very heavy posterior estimation. This thesis posits two main contributions to address the limitations.

First, a semi-supervised approach to extract the grammar rules of entity extraction on a novel dataset using the grammar rules presented on a similar annotated dataset. The approach makes use of the postulate similarity in sentence structures. The semi-supervised approach has shown a promising results with a simple approach to extrapolate one domain to another related domain. The semi-supervised approach perform better than unsupervised approaches relying on manual effort or heuristics for grammar creation. Second, PUER: a parametric neural adaptor grammar is proposed which makes of modern deep learning techniques like distributed representations and word embeddings in addition to inside-output approach [Baker, 1975] to estimate the probabilities of parse trees. Inside-outside approach eliminates the need for prior and posterior estimations and estimates the probability of the start symbol expanding to the whole phrase. A proof of concept is presented for that an adaptor grammar with distributed representations could

work well for unsupervised entity extraction. PUER can be efficient for unsupervised entity extraction in industry setting if the limitation that the model takes too long to train could be addressed.

## 5.2 Future Work

The semi-supervised approach for grammar creation on novel domains has the main limitation that the method needs to be evaluated and experimented on different domains. Measuring semantic relatedness for datasets helps in finding datasets more related to the novel domain to generate the grammar. The approach currently uses the rules for each entity in the home domain individually. The method can be expanded to use rules from multiple entities. PUER's main limitation is that it takes too long to train. For each phrase, the model takes  $O(n^3)$  time to calculate  $\alpha$  value and  $O(n^2)$  time for the Viterbi algorithm to obtain the parse tree with the highest probability. Designing an optimized Expectation Maximization (EM) approach instead of the inside-outside probabilities could help in reducing the time taken for inference. The recurrent neural networks (RNN) can be substituted by attention-based models and pretrained models to experiment with. The design is highly modular and each component can be replaced with different state-of-the-art approaches and experimented on. The thesis proposes a parametric approach for adaptor grammar but the work can be extended to a non-parametric approach where the embedding vectors are from a probability distribution, instead of being randomly initialised. The thesis presents an interesting direction of exploring neural adaptor grammar with a lot of potential to exploit.

## BIBLIOGRAPHY

- James Baker. The DRAGON system—An Overview. *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Acoustics, Speech, and Signal Processing*, 23(1):24–29, February 1975.
- Edward Loper Bird, Steven and Ewan Klein. *Natural Language Processing with Python.*, 2009. URL <https://nltk.org>.
- Shay B. Cohen, David M. Blei, and Noah A. Smith. Variational Inference for Adaptor Grammars. In *Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 564–572, Los Angeles, California, June 2010. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/N10-1081>.
- Michael Collins and Yoram Singer. Unsupervised Models for Named Entity Classification. In *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. Association for Computational Linguistics, 1999.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://www.aclweb.org/anthology/N19-1423>.
- Andrew Drozdov, Pat Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. Unsupervised Latent Tree Induction with Deep Inside-Outside Recursive Autoencoders. In *North American Association for Computational Linguistics*, 2019.
- Micha Elsner, Eugene Charniak, and Mark Johnson. Structured Generative Models for Unsupervised Named-Entity Clustering. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, page 164–172, USA, 2009. Association for Computational Linguistics. ISBN 9781932432411.
- Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence*, 165(1):91–134, April 2005.
- Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Allen AI Organisation*, 2017.
- Matthew Honnibal and Ines Montani. *SpaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing*, 2017. URL <https://spacy.io/>.

- Zhiheng Huang, W. Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. Technical Report abs/1508.01991, ArXiv, 2015.
- Myeongjun Jang, Seungwan Seo, and Pilsung Kang. Recurrent Neural Network-Based Semantic Variational Autoencoder for Sequence-to-Sequence Learning. *Information Sciences*, 490:59–73, 2019.
- Mark Johnson. PCFG Models of Linguistic Tree Representations. *Association for Computational Linguistics*, 24(4):613–632, December 1998. ISSN 0891-2017.
- Mark Johnson. PCFGs, Topic Models, Adaptor Grammars and Learning Topical Collocations and the Structure of Proper Names. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1148–1157, Uppsala, Sweden, 2010. Association for Computational Linguistics.
- Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. Adaptor Grammars: A Framework for Specifying Compositional Non-parametric Bayesian Models. In *Advances in Neural Information Processing Systems*, pages 641–648. MIT Press, Vancouver, 2007.
- Adam Kariv and Rufus Pollock. Bibliographic Data, 2018. URL <https://datahub.io/collections/bibliographic-data>. [Online; accessed: 2020-09-02].
- Yoon Kim, Chris Dyer, and Alexander Rush. Compound Probabilistic Context-Free Grammars for Grammar Induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1228. URL <https://www.aclweb.org/anthology/P19-1228>.
- Adit Krishnan, Aravind Sankar, Shi Zhi, and Jiawei Han. Unsupervised Concept Categorization and Extraction from Scientific Document Titles. In *Proceedings of the ACM on Conference on Information and Knowledge Management*, pages 1339–1348, Singapore, 2017. Association for Computing Machinery.
- Jingjing Liu, Panupong Pasupat, Scott Cyphers, and Jim Glass. Asgard: A Portable Architecture for Multilingual Dialogue Systems. In *Institute of Electrical and Electronics Engineers (IEEE) International Conference on Acoustics, Speech and Signal Processing*, pages 8386–8390, Vancouver, may 2013. Institute of Electrical and Electronics Engineers.
- Liyuan Liu, Jingbo Shang, Frank Xu, Xiang Ren, Huan Gui, Jian Peng, and Jiawei Han. Empower Sequence Labeling with Task-Aware Neural Language Model. In *Association for the Advancement of Artificial Intelligence*, 2018.
- Tomas Mikolov, Kai Chen, G.S Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at International Conference on Learning Representations ICLR*, 2013, 01 2013.

- Prateeti Mohapatra, Yu Deng, Abhirut Gupta, Gargi Dasgupta, Amit Paradkar, Ruchi Mahindru, Daniela Rosu, Shu Tao, and Pooja Aggarwal. Domain Knowledge Driven Key Term Extraction for IT Services. In *International Conference on Service Oriented Computing, Lecture Notes in Computer Science*, volume 11236, pages 489–504, Hangzhou, Zhejiang, China, 2018. Springer.
- Yusra Mosallam, Alaa Abi-Haidar, and Jean-Gabriel Ganascia. Unsupervised Named Entity Recognition and Disambiguation: An Application to Old French Journals. In *Advances in Data Mining. Applications and Theoretical Aspects*, pages 12–23, Cham, 2014. Association for Computational Linguistics.
- Arvind Neelakantan and Michael Collins. Learning Dictionaries for Named Entity Recognition using Minimal Supervision. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 452–461, Gothenburg, Sweden, apr 2014. Association for Computational Linguistics.
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.
- Jim Pitman and Marc Yor. The two-parameter Poisson-Dirichlet Distribution derived from a Stable Subordinator. *Annals of Probability*, 25(2):855–900, April 1997. doi: 10.1214/aop/1024404422.
- Dragomir R. Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The ACL Anthology Network Corpus. *Language Resources and Evaluation*, 47(4):919–944, December 2013.
- Kamal Raj. BERT NER, 2019. URL <https://github.com/kamalkraj/BERT-NER>.
- Claude Sammut and Geoffrey I. Webb, editors. *Viterbi Algorithm*, pages 1025–1025. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8\_878. URL [https://doi.org/10.1007/978-0-387-30164-8\\_878](https://doi.org/10.1007/978-0-387-30164-8_878).
- Jingbo Shang, Liyuan Liu, Xiaotao Gu, Xiang Ren, Teng Ren, and Jiawei Han. Learning Named Entity Tagger using Domain-Specific Dictionary. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2054–2064, Brussels, oct 2018. Association for Computational Linguistics.
- ACL Anthology Team. ACL Dataset, 2020. URL <https://www.aclweb.org/anthology/>. [Online; accessed: 2020-09-02].
- Sagar Vinodababu. A PyTorch Tutorial to Sequence Labeling, 2019. URL <https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Sequence-Labeling>.
- Ke Zhai, Jordan Boyd-Graber, and Shay B. Cohen. Online Adaptor Grammars with Hybrid Inference. *Transactions of the Association for Computational Linguistics*, 2:465–476, 2014. doi: 10.1162/tacl\_a\_00196. URL <https://www.aclweb.org/anthology/Q14-1036>.

Ke Zhai, Zornitsa Kozareva, Yuening Hu, Qi Li, and Weiwei Guo. Query to Knowledge: Unsupervised Entity Extraction from Shopping Queries using Adaptor Grammars. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '16*, page 255–264, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340694. doi: 10.1145/2911451.2911495. URL <https://doi.org/10.1145/2911451.2911495>.