2016

Entity Linking with Convolutional Neural Network

Xu, Shunyi

UNIVERSITY OF CALGARY

Entity Linking with Convolutional Neural Network

by

Shunyi Xu

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER, 2016

Abstract

Entities are real world objects such as persons, places, or events that appear in natural language text such as web pages, news, and journals. *Entity Linking*, a nascent field in Natural Language Processing, is the task of linking entities in text to their referent entries in a *Knowledge Base* (KB), which is a repository of information such as Wikipedia. There's a huge application of *entity linking* in *automatic knowledge base population*, prevention of identity crimes, etc. It can also provide background information about unfamiliar concepts during document reading, rendering a smooth and joyful reading experience without frequent "context switch".

This thesis taps into the power of convolutional neural network, and proposes an architecture that makes use of deep learning layers, convolution, max pooling, and fully-connected neurons with dropout to approach the problem of entity linking. Based on a pre-trained *word2vec* word embedding and another ad-hoc trained layer of word representation, we were able to outperform previous state-of-art models, which handcrafted a large number of features, by a modest margin.

Visualization of the neural network is also provided in order to understand what happens under the hood. Our experiment showed that it clearly captured the desired features, indicating the efficacy of neural network in dealing with *entity linking*.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Reda Alhajj, for his unwavering support and kind suggestions to my graduate research. His patience and inspiration has been a beacon to guide me during the ups and downs of the project.

I would like to extend my appreciation to all my friends in the database group. Thanks to Alper, who flipped my switch on in deep learning. Thanks to Zohbi, who engage me in several sensible discussions about cultures and languages. Thanks to Abed, we've had an interesting experience TAing python. Thanks to Gabi, Ibrahim, Salim, Omar, Kostas, and Sarhan who have made this lab lively and laid generous support in coordinating group events and department-wide activities. Thanks to Tamer, who treated us to a great feast in the group briefing. Thanks to Ayessha for preparing plentiful materials in the database class, and being a great TA in pointing us in the right direction. Last but not least, I would like to especially thank Manmeet for proofreading the thesis. New to the lab, you have brought so much liveliness to our ambiance, and I hope you would realize your dream of auto-translating novels in the near future.

I am also indebted to my friends in the other lab. Thanks to Linquan for being a good senior buddy (shixiong), and may your sweet romance with Xueying blossom beautifully in the future. Thanks to Yuhui for co-working on the deep learning model in CPSC 671. Thanks to Xunrui, Sijia, Wei, Ruiting, Yang Liu, Yao Zhao for all the good days we've been together. I wish the best for your future studies and career.

Finally, I would like to thank my family for their support through my entire life. This thesis is dedicated to you.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
KB	Knowledge Base
NLP	Natural Language Processing
NGD	Normalized Google Distance
CRF	Conditional Random Field
ILP	Integer Linear Programming
CNN	Convolutional Neural Network
w2v	Word2Vec
IQR	Interquatile Range
LDC	Linguistic Data Consortium

Chapter 1

INTRODUCTION

With the penetration of World Wide Web in our daily life, there is a drastic change in the way people interact with information and acquire knowledge. *Knowledge Base* (KB), which is a repository of information, is a common source people turn to when there is a need to look up terms or learn facts. *Wikipedia* is a well known site serving this purpose. Since its creation in 2001, it has collected 5,223,573 fact entries, and provides reference for hundreds of thousands of visitors from around the globe everyday. *DBpedia* [28], which is another site pivoting on similar ideas, is a refinement based on Wikipedia and other information sources. It contains 4.58 million knowledge entries, and is able to handle more sophisticated questions from users. *Freebase* [2], which is a practical and scalable tuple database used to structure general human knowledge, serves also as a prominent KB. Thus far, it contains more than 125 million tuples of more than 4000 varying types.

Many of these knowledge bases are built by crowdsourcing, which requires heavy human involvement. The laborious, costly, and tedious process of information/fact extraction and labeling significantly limits the coverage of knowledge bases. It would be more efficient if we could automatically populate the knowledge base using facts/information that already exists out there in the web. Thus came the *Automatic Knowledge Base Population*, which is a nascent and hot field in Natural Language Processing (NLP) recently [22] [10] [1] [29] [8]. A lot of KBs sprouted out under this principle. Typical examples are YAGO [49], NELL [4], and Reverb [12].

Automatic Knowledge Base Population is challenging due to the inherent deficiency of current web layout. The web is mostly unstructured, with no unified approach to automate the process of accessing the data/fact/information therein. This calls for the need of appropriate *information retrieval* technique, which targets to retrieve the relevant information from unstructured sources.

Information retrieval is often entity-centric. Entities are real world objects such as a person, a

place, an organization, etc. Given any document, we are generally concerned about such questions as what words in the text represent entities, what the relation among them is and what real world objects they actually refer to. This brings up three important and closely-related tasks that can form a processing pipeline if necessary.

- **Named Entity Recognition:** Named Entity Recognition, or NER in short, identifies spans of words in text that refer to some entity, and figures out what type it is. Common types include *People*, *Location* and *Organizations*. In a special domain, named entities may also refer to terminologies exclusive to that area. A typical example would be *Genes* in bio-informatics.

- **Relation Extraction:** Given two identified tokens representing entities, relation extraction aims to figure out the relation between them. For instance, given two person names about education, identify who is the teacher and who is the student. Relation extraction can be closed-domain or open-domain. In closed-domain, a pre-known set of constraints such as fixed type of relation are imposed, while in open-domain, there's no such restriction.

- **Entity Linking:** Both NER and relation extraction deal with *type*, i.e., is this entity of type *person* or are these two entities associated with relation type *couple*. Entity linking answers questions directly about instance, i.e., is this entity Bill Gates or Steve Jobs. That said, entity linking tries to understand the candidate words, and maps it to its real world object representative, which is usually an entry in the knowledge base.

Entity linking is a critical last step in automatic knowledge base population. It aligns what we find in the document directly with the knowledge base. In addition to many of its inherent difficulties such as scale of data, this task is mostly complicated by the ambiguities and varieties of language. For example, the *"Clinton"*, in the news *"Speaking at a community college in Reno,*

Nevada, Clinton said Trump had brought a racist fringe into the mainstream in a way that no other major party nominee had in American history", is clearly a political figure that co-occurs with *Donald Trump* quite often. However, it's not clear whether this Clinton refers to *Bill Clinton* or *Hillary Clinton*. Further context is needed to reveal the true Clinton in this case.

In some situations, the same entity could go by different names. For example, in the text *"There's no place quite like New York. The Big Apple is the top travel destination in the U.S."* , both *"New York"* and *"Big Apple"* refer to *New York City*. Nicknames, in this case, also need to be resolved to the right entity.

Since the ability to disambiguate a polysemous name or infer whether two orthographically different mentions are the same entity is involved, the task of *entity linking* sometimes goes under the name *entity disambiguation*. In the remaining parts of this thesis, we will use these two terms interchangeably.

Outside automatic knowledge base population, entity linking also sees huge applications in other areas. For example, in health science, it is useful to link documents of patients to some health repository in order to maintain personal health records. In criminology, linking criminals to a crime knowledge base may help facilitate research in crimes, prevent identity crime, and support law enforcement. [38]

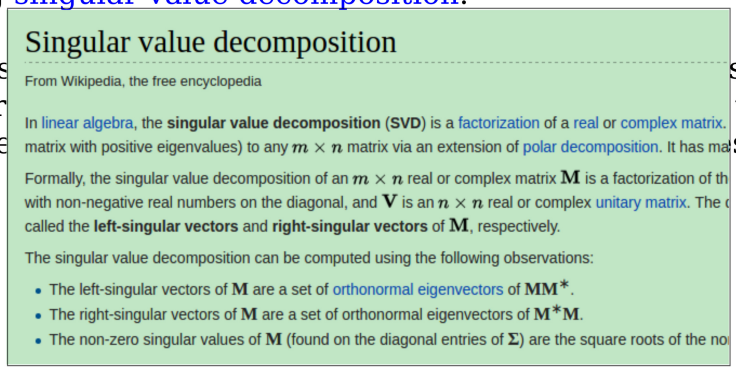
Another possible and potentially popular application of *entity linking* is to bring explanatory links to named entities in all documents. It saves readers the trouble of manually searching the Internet to look up terms and makes their reading experience more smooth and joyful.

Figure 1.1 depicts such an application. In a snippet of academic paper, all entities recognized are highlighted in blue, and if user clicks on one of them, its referent Wikipedia entry pops out. In this example, *singular value decomposition* is the entity linked. Readers can quickly grasp knowledge about it without "context switching" to other tasks. As a result, readers can remain consistently in the reading session without getting disrupted.

Data were preprocessed using statistical parametric mapping software (SPM12).50 Functional volumes were first slice-time corrected and then motion corrected.

There were no significant differences between groups/time in mean relative motion and max absolute motion (see Supplementary Information for descriptive statistics).

Processing in both EVC and ROI to voxel analyses We extracted a principal time series from the white matter (WM) and cerebrospinal fluid using singular value decomposition.



We us s from the
prepr voxel. We extracted
the re sents the time series

Figure 1.1: Document Reader Application

1.1 Problem Description and Terminology

We formalize the entity linking task and the terms involved therein as below.

Definition *Entity Mention/Surface Name:* Given a snippet of text/document, the phrases suspicious of referring to some entity is called *Entity Mention* or *Surface Name*.

For example, in the text *"Michael Jordan is a renowned fellow in machine learning"*, the phrase *"Michael Jordan"* is suspicious of referring to an entity, and is called *entity mention* or *surface name* of some person named *"Jordan"* in the real world.

Definition *Canonical Name/Ground-truth Name:* The name phrases used to refer to the real entity in the knowledge base.

Using the previous example, we know that the *surface name "Michael Jordan"* actually refers to the "Michael I. Jordan" who is a professor at University of California, Berkeley and leading

researcher in machine learning. So the *canonical name*, or *ground-truth name* in the terminology of labelled/golden dataset, is *"Michale I. Jordan"*. Usually we make whatever appears as the title entry in the knowledge base its *canonical name*. In this work, we make the entry of Wikipedia its *canonical name*. Some other examples of entities that go by the surface name of "Michael Jordan" are: "*Michale Jordan (basketball)*", *"Mike Jordan (racing driver)"*, and *"Michael B. Jordan"* (who is an American actor).

Definition *Entity Linking:* Given a document set D, for each document $d \in D$, which contains an entity mention m, create a mapping from m to its referent real world entity e in the knowledge base K.

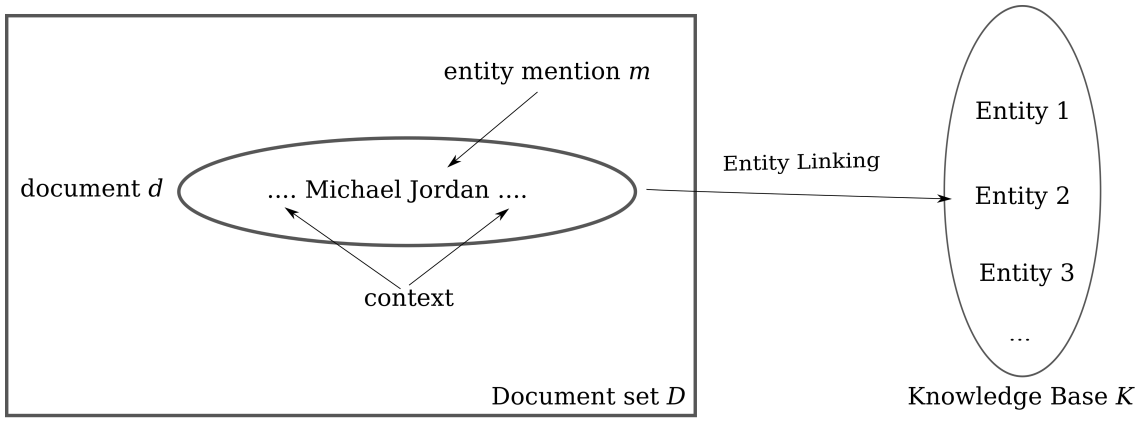


Figure 1.2: Entity Linking

Figure 1.2 depicts the complete setting of *entity linking* according to the above definition. The document set contains multiple documents, and each document comprises of the *entity mention* and *context*. The context is everything other than the *entity mention*, and it is supposed to help our model understand what the entity really is, and aid in the linking task. For each mention in every document, we create a link to its referent entity in the knowledge base.

1.2 Contribution

In recent years, many techniques have been proposed to solve entity linking problem using Wikipedia as the knowledge base. They usually explore three types of features.

The first one is *entity popularity*, which is a statistical feature. It's based on the simple assumption that the most prominent entity is usually the most likely referent entity for a given entity mention. The way they justify "prominent" is based on how often the entity uses certain surface names, and the frequency is counted by making use of the anchor text of a Wikipedia hyperlink. This is a very simple heuristic approach that may work for many cases. However, it lacks the intelligent depth and flexibility to work with cases that use more rare surface names. Regardless of the context that come along, any surface name will be linked to one entity, which is the entity that uses this name most. As a result, this approach is stranded in one fixed point, and we will see a unanimous and monotone outcome for the same entity mention.

The second feature is *context similarity*, which is a textual feature. It complements the first problem by taking context into consideration. It defines some *similarity measure* and estimate how close the text surrounding an entity mention is to the document in the knowledge base describing the entity. Though the specific *similarity measure* varies, it largely depends on exact word overlap, which is often too strict a restriction. It also fails to take the semantic similarity of words into consideration. For example, "round" and "circle" may be drastically dissimilar because they are so different literally. However, they are close in semantic sense.

The third feature is called *topical coherence*. Instead of addressing this task from a word's point of view, it abstracts the document on a topic level, and compares their similarity in terms of topic. The rationale behind this is that the referent entity in the knowledge base should describe similar topic to that of the entity mention in any unstructured text. This approach has some flavor of *semantic similarity*.

Different from all the previous approaches, we propose to use convolutional neural network to tackle the entity linking problem. We separate entity and context, embed them into a high

dimension continuous space, and use convolutional neural network to automatically pick up the features in the training example. We combine the features generated and use a linear classifier to assign them to the relevant entity.

What sets us apart from all of the above feature-oriented approaches is that the neural network can figure out the key features in the training sample, and step on the course it deems appropriate without human intervention. This saves us a lot of trouble in handcrafting the features, and deciding which are useful and which need to be discarded. It also removes subjectivity from determining the fate of features, which is not always correct.

In addition, the convolutional neural network can implicitly calculate the statistical feature of the samples. Terms that appear more often will assign heavier weights to their connected neurons, and as we feed tens of thousands of training samples, the network implicitly picks up the statistical counts for us.

Our neural network can also address the exact-word-overlap problem in the *context similarity* approach. Since we used word embedding, it will map words with similar semantic to closer spatial positions. Hence, instead of working on literate/textual similarity, we are dealing with semantic similarity.

The neural network is also able to implicitly figure out the topic. Since words that appear more often in certain entity will be seen by the neural network, they can be interpreted as the topical words of the entity. For example, the neural network might see *Barack Obama* co-occur with *US president*, *Black*, and *Democrats*. These can be seen as the topical words for the entity *Barack Obama* and any surface name whose context contains these words is likely to be a well-defined evidence to refer to *Barack Obama*.

To some degree, our model could be seen as a superset of these three features, because it automatically incorporates them, and is not restrained from discovering more. We can rely on the neural network to discover features for us, and leave it to decide which are important and which are not.

Besides, we constructed a training dataset based on a commonly used test dataset. Previous test data sets are mostly feature based, and are used with unsupervised approaches. There is a lack of an appropriate training set in mind. Since training is an inevitable part in neural network, we constructed a set of training data by crawling Google news. This training dataset could be useful for future research under similar scope.

1.3 Thesis Structure

The rest of the thesis is structured as following. Chapter 2 provides background and related work on entity linking. Chapter 3 details the model we propose to use and elaborates on the convolutional architecture we bring forth. Chapter 4 describes the dataset for experimentation and some pre-processing mechanisms. Chapter 5 illustrates our experiment results, and introduces some visualization primitives to better understand the neural network. Last but not least, we conclude in chapter 6 with summaries and future work.

Chapter 2

RELATED WORK

We survey some of the related literature in this chapter and present them in the following sections of *Entity Linking with Handcrafted Features* and *Neural Network*.

2.1 Entity Linking with Handcrafted Features

We categorize the work into the following subsections based on the methodologies they use, and each of these subsections will showcase a separate aspect of the entity linking research. Please be advised that the categorization is not necessarily disjoint, so some papers may actually have used a plethora of methodologies.

2.1.1 Initial Effort and Textual Similarity

Wikipedia was first used for entity linking in the work of Bunescu and Pasca [3] in 2006. They trained a *Support Vector Machine (SVM)* to exploit the lexical contents of Wikipedia pages. They made direct textual comparison between the query document and the Wikipedia page with cosine similarity. However since they needed to train a separate SVM model for each entity mention, their proposal could only be carried out in a very limited scale.

Subsequently, in the work of "Wikify: linking documents to encyclopedic knowledge" [33], Mihalcea and Csomai combined keyword extraction and entity disambiguation into one system. They experimented with two approaches in the disambiguation part, one was to compare the lexical overlap between the candidate Wikipedia entries and the query document, and the other was to train a Naive Bayes classifier for each entity mention. Again they suffered the problem faced in [3] because it requires one classifier per mention.

2.1.2 Semantic Similarity

All above methods focus on textual closeness between the query document and knowledge base entries. They are a) very strict with words overlap and b) fail to measure the semantic similarity between them. For the latter point, there exists abundant of information in the Wikipedia that was often neglected by previous methods.

In view of this, Cucerzan [9] presented a large-scale system for entity disambiguation by incorporating more non-literate features. They employed a vector based comparison model using the disambiguation pages, redirects, and categories available on Wikipedia.

Milne and Witten [34] made use of the anchor text in links, and used a self-defined *commonness* and *relatedness* metric [55] [32] to measure the semantic similarity between entity mention and entity entry in the Wikipedia. They also combined *entity linking* with *entity recognition*. Different from traditional linking cult, which identified the entities first before linking them to the KB, they linked the entities first and used the linked probability to decide if it's good enough to get linked anyway.

Ratinov *et al.* [39] proposed a *local* and *global* approaches to address entity linking problem. In their phrases, *local* referred to the textual similarity between entity mention (along with its context) and the Wikipedia pages and each mention was linked independently. On the other side, *global* was a set of features that were based on the entire document and all mentions were disambiguated concurrently in order to produce coherent results. For example, if a mention of *Michael Jordan* refers to the machine learning professor in the document, then the mention of *Monte Carlo* is expected to refer to the statistical technique instead of the location. Features used in *global* approach are *Normalized Google Distance (NGD)* and *Pointwise Mutual Information* [55]. They exploited the incoming and outgoing link structures of Wikipedia, and it was a refinement of previous works.

2.1.3 Presented as Graph Problem

Han and Zhao [17] proposed *social relatedness* to represent the similarity of entities in the sense of a potential social network. For instance, in the Wikipedia article of *Michael I. Jordan* (recall from our introduction section that this is the machine learning professor instead of the basketball superstar), it contains a lot of key words related to other machine learning big name such as *Andrew Ng and David Blei*. This indicates that the *Jordan* in any text co-occurring with *Andrew Ng* or *David Blei* is more likely to be referring to the machine learning *Michale I. Jordan*. Aware of this concept of similarity, Han and Zhao built a *concept graph* to represent closeness between entities keywords. The *concept graph* features a set of nodes which represent entities in the Wikipedia repository. The edges between node represent relatedness between concepts. The smaller the weight, more related the neighboring entities are.

Along the line of presenting the problem as a graph problem comes the work of Han *et al.* [16], which incorporated more nodes into the graph. They defined a *Reference Graph*, which also incorporated mentions explicitly in the graph.

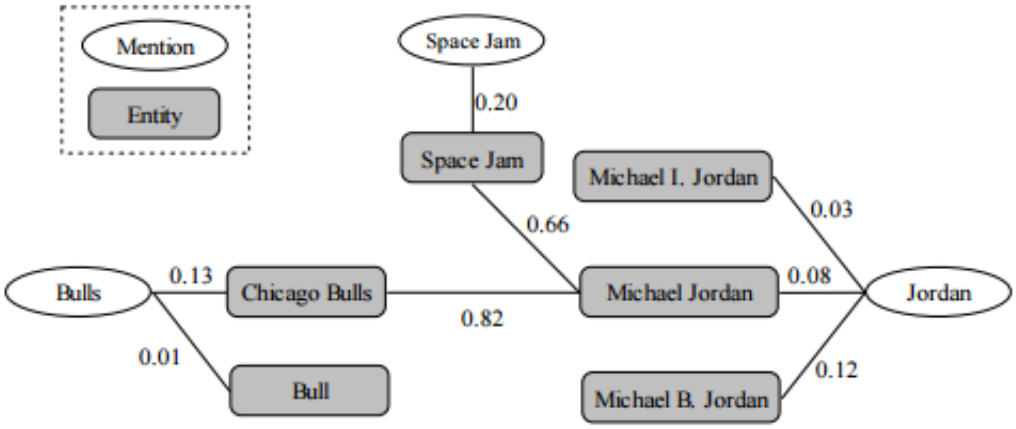


Figure 2.1: Referent Graph

As shown in Figure 2.1 [16], the entity mentions are denoted in white oval boxes and the canonical entities are shown in grey rounded rectangular boxes. There are two kinds of edges. The edges between grey boxes are *compatible* index. They show how close the referent entities are

in real world. The edges between white box and grey box are *semantic-related* index between a mention and a canonical entity. Throughout the process, the evidence score of a mention referring to certain entity is calculated and in the end, the entity that has the highest evidence score multiplied with some compatible score is the mention's true canonical entity. The graph's weight is iteratively updated, with an initial weight assigned according to the *prior* importance score to each name mention based on *tf-idf* index, which serve as the bootstrap evidence. The evidence is further propagated through the two kinds of edges in different ways.

Inspired by a similar idea, Hoffart *et al.* [19] unifies previous approaches into one comprehensive framework. It combines the prior probability of an entity being mentioned, the similarity between context and entity, and coherence among candidate entities for all mentions together. Different from some of its previous work, it used a knowledge base derived from Wikipedia and YAGO.

2.1.4 Domains Other than Web Document

Using similar taste of graph, Shen *et al.* [43] extended the general domain of text, which is usually in the form of web document, into the world of Tweets. In this special domain, text is notoriously known for being noisy, short, and informal. Besides, since previous methods largely relied on the context around entity mention and topical coherence between entities in the documents, the tweet entity linking task becomes extremely challenging due to insufficient context information in the tweet. In order to combat this issue, the author assumed that each user has an underlying topic interest distribution over various named entities, and integrated intra-tweet local information and inter-tweet user interest into a graph. Throughout the process, an *interest score*, a weight of node, is calculated and propagated. In the end, whatever entity that enjoys the highest interest score is the inferred entity for a given entity mention.

In addition to that, Shen *et al.* [42] extended the domain into heterogeneous information network, which is a generalization of documents that lack features specific to Wikipedia related

knowledge bases such as social media networks and bibliographic networks. They proposed a probabilistic model to approach the entity linking task under such setting. Particularly, it featured an entity popularity model that measures how popular an entity is, and an entity object model that captures the distribution of multi-type objects appearing as context of the entity, which is generated using some *meta-path constrained random walks* over the network. They used the *expectation-maximization (EM)* algorithm to automatically learn the weights for each *meta-path*, which requires no training data.

2.1.5 Entity Recognition and Entity Linking in One Shot

There are also works that combine *entity recognition* and *entity linking* together. Different from previous "combination" such as in [33], which is merely a pipeline that achieves these two tasks separately, Sil and Yates [44] proposed a probability model that completed these two tasks at the same time. Besides, it also used a combination of Freebase and Wikipedia as their knowledge base, because they provide richer type information. Since it performed joint entity recognition and entity linking, the canonical problem of entity linking was perturbed to a slightly different problem as such: Given any document d, identify a set of tuple (b,e) where b is some candidate mention in the document and e is the referent entity in the knowledge base. They decomposed the document into small *connected components*, where each *connected component* is a set of candidate entity mentions that are within three words apart from each other, a simple heuristic adopted by them in fit with their dataset. They argued that, candidate mentions that are close can be analysed collectively because their features can enhance each other's recognition and linking accuracy. They then used maximum-entropy model to estimate the posterior probability of a candidate mention b given the document d and a connected component cc of d, aka $p(b|d,cc)$. The maximum-entropy model will employ a set of feature functions (such as those features we covered in the introduction chapter), and each feature is associated with some weight. They took the L2-regularized conditional log likelihood (i.e., sum of log of all such $p(b|d,cc)$ plus L2 weight sum) as the objective function to

maximize in training, in order to learn weights.

Guo *et al.* [48] also considered joint mention detection and entity disambiguation, but their technique were suited for microblog texts, which are short, and able to afford more computationally expensive approaches. They used a structural SVM algorithm [40] [52] [6], which is a generalized SVM algorithm that can output structured labels. In addition, they used Wikipedia as the knowledge base.

In a more bold attempt, Durrett and Klein [11] proposed a joint model to address *coreference resolution*, *entity recognition* and *entity linking* in one joint model. *Coreference resolution* [46] [35] [37] is a task to identify cluster of entity mentions in a document referring to the same entity. This is based on the observation that these three tasks are usually closely related.

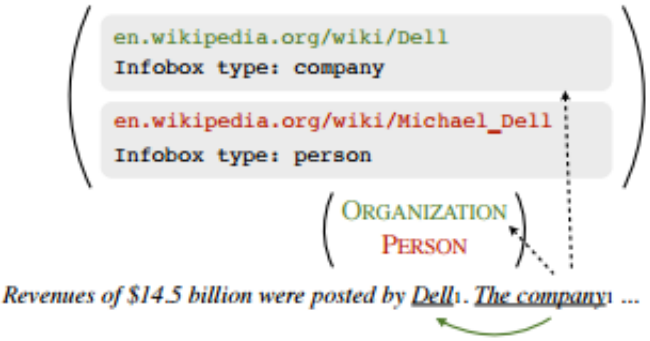


Figure 2.2: Coreference help resolve ambiguous cases of semantic types and entity links

For instance, in the Figure 2.2, *The company* co-refers to *Dell*, and we know *The company* is clearly describing a company, therefore *Dell* is of type *Organization* (in *entity recognition's type terminology*) and the linked entity should be Dell, the company, rather than Dell, the person. They proposed a structured *Conditional Random Field (CRF)* to solve this problem. CRF is a statistical technique introduced in [26] and is widely adopted in applications of [13] [51] [54]. It uses "neighboring" samples to predict a label for a single sample, so they can produce consistent outcome.

Similarly, Cheng and Roth [7] unified relation extraction with entity linking. They argued that

the relation between entities can provide richer information to augment the entity linking task. As a result, they formed an *Integer Linear Programming (ILP)* that incorporates relation inference and entity linking in one problem.

2.2 Neural Network

Neural Network is a classic technique used in machine learning and pattern recognition [18] [41] [20] [31] [5].

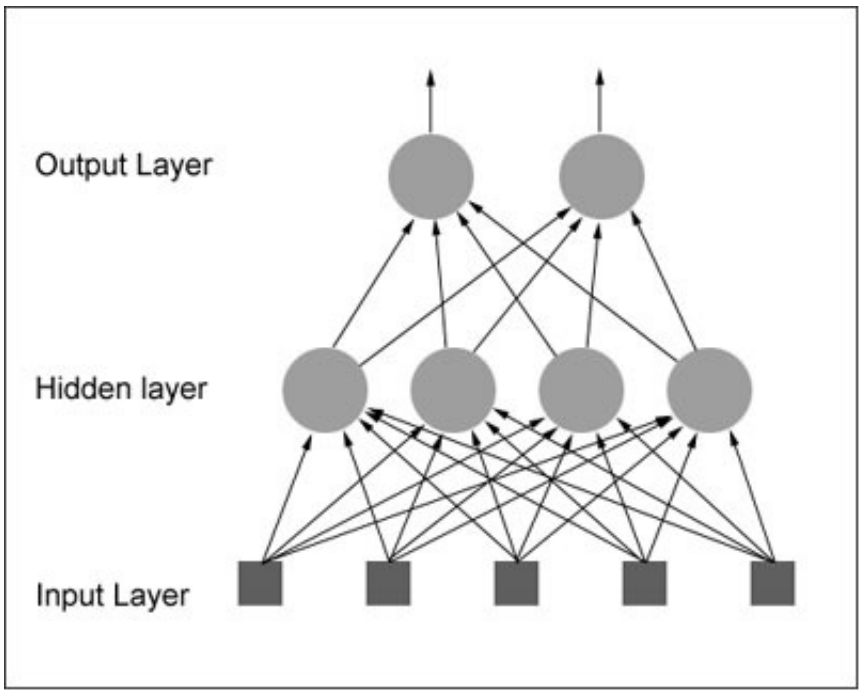


Figure 2.3: Artificial Neural Network

As demonstrated in Figure 2.3, the *neural network* features an input layer, a hidden layer of neurons, and an output layer. Neurons in the hidden layer are fully connected with all input layer signals and output layer neurons, where each connection features a weight that captures the importance of this connection. Each neuron's output is based on the weighted sum of its connected input, following some activation function, which introduces nonlinearity to the entire network. In the training, calculations propagate from input layer to hidden layer and to the output layer, which

is called *forward passing*, and after some cost function is calculated and the gradients of variables are known, the gradients are passed back in the reverse direction, which is called *back propagation*. Thus, we can adjust the weights of the neural network to optimize some user defined objective function.

Convolutional Neural Network (CNN) is a variant of the classical *neural network*. Different from traditional *neural network*, CNN uses neurons only to observe a subset of its input volume, which means it's not fully connected. As a result, the number of parameters is largely reduced. We can fuse multiple layers together in CNN, with bottom layers discovering primitive patterns and top layers constructing sophisticated patterns. CNN has seen tremendous success in computer vision [25] [24] [27] [23].

There is currently not much work addressing entity linking using convolutional neural network. We are only aware of three such pieces. Sun *et al.* [50] proposed using CNN to encode both the query document and positions therein to a feature vector, and compare it with Wikipedia entries, encoded by another CNN on its title and type, using cosine similarity. Francis-Landau *et al.* [14] proposed to encode entity mention, context, and document level information to a vector using CNN. They then mapped Wikipedia articles to vectors using a different CNN to encode their titles and main contents. They compared the mention and candidate Wiki pages using cosine similarity. Huang *et al.* [21] combined CNN and Knowledge graph which is a refinement of previous pure graph based presentation.

Our work differs from [50] and [14] in that we don't use cosine similarity and we treat it more as a classification problem rather than a ranking problem. Besides, we are able to construct more layers than theirs. Our work differs from [21] in that they formulated the problem more as a graph problem, while we treat it as a classification problem that can be solved with CNN.

Chapter 3

MODEL AND ARCHITECTURE

3.1 Word Embedding

Word Embedding [30] [15] [36] is a way of distributed representation of words in continuous high dimension space. Traditionally, words are treated as discrete atomic symbols. The *one-hot vector* model, which once gained traction, is a typical representative. In the *one-hot vector*, each word is denoted in a $\mathbb{Z}^{|V|}$ binary vector, where $|V|$ is the size of vocabulary. 1 only appears at the index of the word in some sorted vocabulary while the remaining positions are filled with 0. For example, the word *cat*, *dog*, *zone* will be denoted as following in some made-up vocabulary, where *cat*, *dog*, *zone* is the first, third, and last word of that vocabulary after sorting.

$$encode(\text{cat}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad encode(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \qquad encode(\text{zone}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \tag{3.1}$$

However, these encodings are arbitrary, providing no useful information that reveals their relation.

$$encode(cat)^T \cdot encode(dog) = encode(cat)^T \cdot encode(zone) = 0 \tag{3.2}$$

For example, in Equation 3.2, the inner product of cat and dog, and that of cat and zone are both zero, which means they are equivalent from a mathematical perspective. However, cat and dog share similar meanings, and from a linguistic point of view, their dot product, also known as cosine

similarity, should be smaller than that of cat and zone. This is a property that cannot be guaranteed under *one-hot vector* model. Besides, discrete data symbols can also lead to data sparsity problem, which means we will need more data to train the model.

Therefore, we need a better word representation model that can express the similarity of words. *Word embedding* is such a model so that similar words could be mapped to close positions in high dimension space. A reasonable word embedding model could perform the mapping of above three words into the following vectors.

$$encode(\text{cat}) = \begin{bmatrix} 0.3 \\ -0.2 \\ 1.4 \\ \vdots \\ 2.33 \end{bmatrix}, \qquad encode(\text{dog}) = \begin{bmatrix} 0.4 \\ -0.1 \\ 1.23 \\ \vdots \\ 3.1 \end{bmatrix}, \qquad encode(\text{zone}) = \begin{bmatrix} 0.3 \\ 5 \\ -2.5 \\ \vdots \\ -12 \end{bmatrix} \tag{3.3}$$

This time, each coordinate of *encode(dog)* is a slight offset from *encode(cat)*, while that of *encode(zone)* is vastly different from *encode(cat)*. One can reasonably deduce that *cat* and *dog* are more closely related than *cat* and *zone*.

Turian et al [53] presented a illustration of embedded words after dimension reduction in the 2D space. As shown in the Figure 3.1, similar words such as numbers (*three*, *four*) are close to each other, forming some kind of a number region, while title terms such as *president* and *executive* share a region of themselves on the right side.

Word embedding is usually obtained with neural network training. Two models are particularly favoured in practice. *CBOW* (Continuous Bag of Words), is known for generating the middle word given its left and right context, while *Skip-gram* features predicting left and right context given the middle word. Figure 3.2 presents the neural network used in training word embedding under *Skip-gram* architecture, because it reportedly does a better job for infrequent words than *CBOW* [1].

[1] https://code.google.com/archive/p/word2vec/

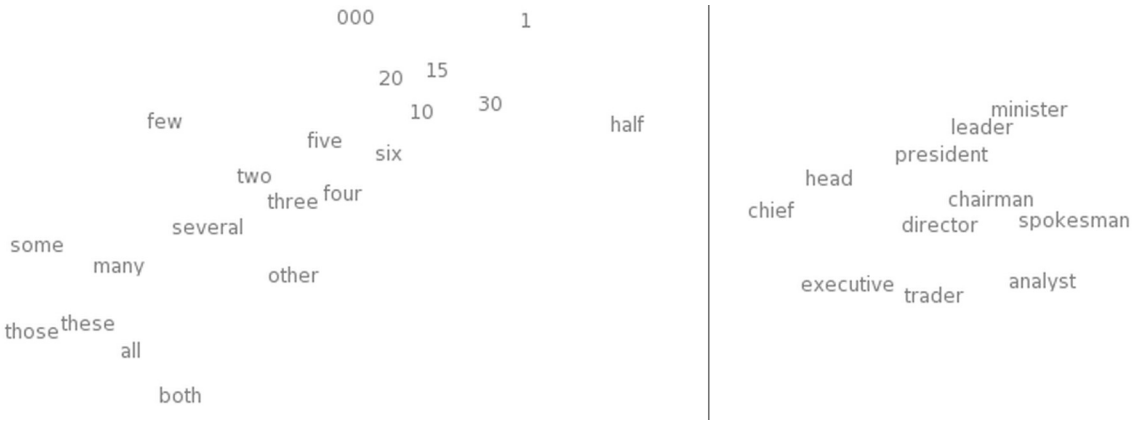


Figure 3.1: Word Embedding

In the figure 3.2, W and W' are two embedding matrices to be learned, W is the input matrix, and W' is the output matrix, which is used in our next layer. Given a training document of words sequence $w^{i-C}, w^{i-C+1}, \cdots, w^{i-1}, w^i, w^{i+1}, \cdots, w^{i+C-1}, w^{i+C}$ of a left/right context window C, it seeks to maximize the log probability of context words given the center word w^i, under strong conditional independence assumption (i.e., context words are completely conditional independent on center word).

The input x_i is word w^i expressed in *one-hot* vector mode, and the output $y_1, y_2, \cdots, y_C$ is interpreted as the probability distribution among the vocabulary at each context position (for simplicity, let's just discuss about the right context for now because the left context is symmetric). In the training, we get the word embedding u^i for word w^i via the input embedding matrix W. We have $u^i = Wx$. This will set hidden layer $h = u^i$. We then get the score for each of the C context words $v^{i-C}, \cdots, v^{i-1}, v^{i+1}, \cdots, v^{i-C}$ using $v = W'h$. We treat this score as unnormalized log probabilities, which means we need to apply softmax function to get the real probability. Hence the $y = softmax(v)$ is used to get the probability. Ideally, we want this y to be like the true probability where each y is the one hot vector of the actual context words.

That said, the objective function is defined as below, where we can train to get the embedding

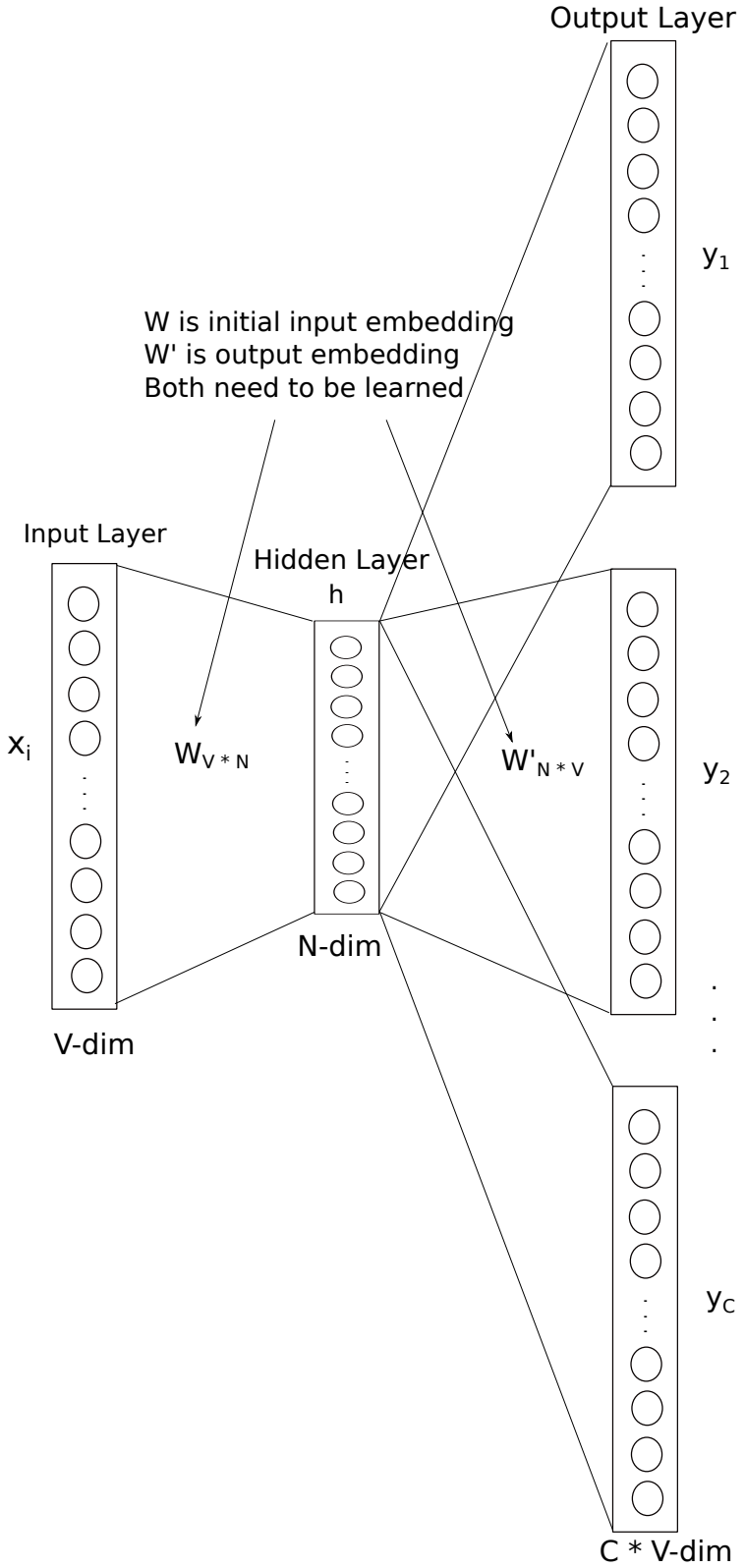


Figure 3.2: Skip Gram Architecture

W and W' via stochastic gradient descent.

$$\begin{aligned}
\text{maximize } J &= \log P(w^{i-C}, w^{i-C+1}, \cdots, w^{i-1}, w^{i+1}, \cdots, w^{i+C-1}, w^{i+C} | w^i) \\
&= \log \prod_{j=0, j\neq C}^{2C} P(w^{i-C+j} | w^i) \\
&= \log \prod_{j=0, j\neq C}^{2C} P(v^{i-C+j} | u^i) \\
&= \log \prod_{j=0, j\neq C}^{2C} \frac{\exp\left((v^{i-C+j})^T h\right)}{\sum_{k=1}^{|V|} \exp\left((v^k)^T h\right)}
\end{aligned}$$

The second equality is based on strong conditional independence assumption, the third equality is due to the embedding of words, and the last equality is the softmax function.

After the embedding matrix W' is obtained, we retrieve it to be our first channel of embedding table. We denote it as our W_1, channel 1 lookup table. Given any word w_n in the document, $W_1(w_n)$ maps the word to $\mathbb{R}^d$, d dimensional space. This serves as our baseline embedding approach. We could also add another channel of word embedding by using lookup table W_2. W_2 could be learned along the way in our entity linking task. Different from W_1, which is solely learned in a language model setting and applied to generic articles, we can mandate W_2 to be task specific, its parameters fine tuned to our entity linking task. One common practice might be to pre-train W_1 using the architecture described in Figure 3.2 and keep its parameters fixed during the entity linking training, while randomly initializing W_2 and adjusting its parameters on the fly during training. This way, our word embedding not only provides generic semantic word relation known from channel 1, but also offers task specific word relation from channel 2.

In theory, we could use many number of channels as we want. Figure 3.3 illustrates a universal architecture for our embedding layer. There are n channels of embedding, where EW_1 is the embedding matrix as described above. $EW_2, \cdots, EW_n$ could be some other embedding tables. The document d is taken as input, and it is further split into two subcomponents: context d_c and entity d_e, with surface names replaced by some placeholder symbol. Each of these subcompo-

nents go through the same n embedding tables, producing n channels of encoded result for both context and entity. We denote $x_c = [x_c^1, x_c^2, ..., x_c^n]$ as the n channels encoded context vector and $x_d = [x_d^1, x_d^2, ..., x_d^n]$ as the n channels encoded entity vector. x_c and x_d are further fed into subsequent layers for entity linking.

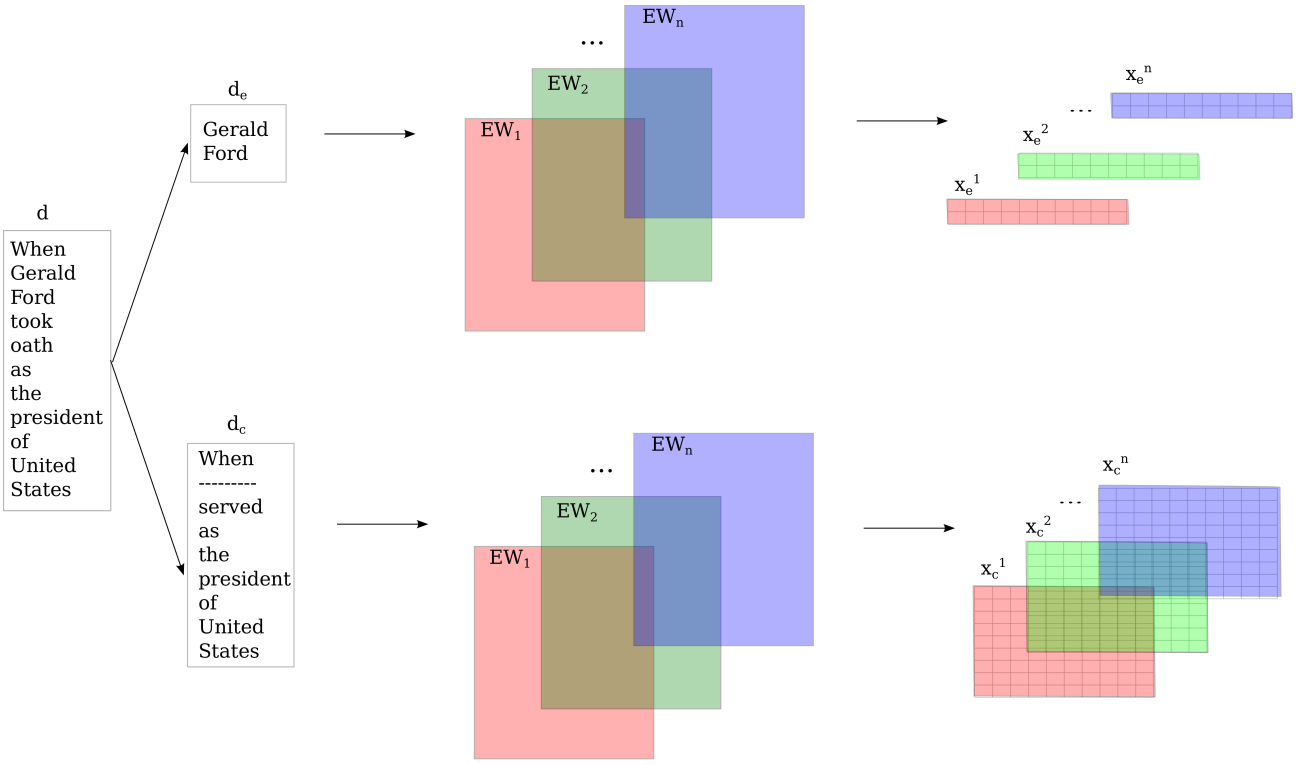


Figure 3.3: n Channel Embedding Layer

3.2 Convolution and Pooling

3.2.1 Convolution and Activation Layer

Convolution is a mathematical operation applied on two functions that produces a third, modified function. In 1-D space, the convolution h is expressed in the following in terms of a kernel function f in domain of $[1,m]$, and an input function g in domain of $[1,n]$.

$$h(i) = (f * g)(i) = \sum_{k=1}^{m} f(k) * g(i-k+m/2) \tag{3.4}$$

One way to think of convolution is to flip the kernel function, which is a fixed window of length m according to above definition, and slide it over the input. For each position of the flipped kernel, we multiply the overlapping values of the flipped kernel and the input, and add up the results.

The convolution can be easily extended to 2D space, especially in image processing. Given a kernel matrix of shape $[mr, mc]$, and an input image g of shape $[nr, nc]$. The convolution at position i, j is as following

$$h(i,j) = (f * g)(i,j) = \sum_{k_1=1}^{mr} \sum_{k_2=1}^{mc} f(k_1,k_2) * g(i-k_1+mr/2, j-k_2+mc/2) \tag{3.5}$$

As with 1D convolution, the 2D convolution can be thought of first flipping the kernel matrix horizontally and vertically, and then sliding it over the input image to get summation of product of overlapping values. And again, it is a linear combination of the areas covered in the window of length $[mr, mc]$ as defined by the kernel f.

In NLP, our input g, which is x_c or x_d obtained from the last section, contains at least 2 dimensions. The first dimension is the number of words in the text, and the second dimension is the dimension d used in word embedding. In case we use multiple channels of embedding, there is a third dimension: number of embedding channels. That calls for convolution of 3D space. Similar to 2D convolution, the 3D convolution is simply a 3D kernel matrix (flipped horizontally, vertically and depth-wise) applied to a 3D input, where the result is simply a linear combination of values covered by the sliding window as defined by the 3D kernel matrix.

Figure 3.4 illustrates an example of convolution structure in 3D space. The left light green volume is the input g (for convenience, we abuse the definition of g, which means function of input above, to generally refer to input space) with a dimension of $[g_h, g_w, g_d]$, which respectively represents the dimension along height, width, and depth axis. The blue circles are kernels f. In the terminology of neural network, it's neurons that carry out the kernel functions f. There are n neurons, which indicates that there are n different kernels $f_1, ..., f_n$. For any kernel f_i, let its 3D flipped matrix be W_i and is of shape $[f_h, f_w, f_d]$. For any region $x \in g$ covered by the kernel window such as the small green volume in the figure (x is also of shape $[f_h, f_w, f_d]$) , it computes

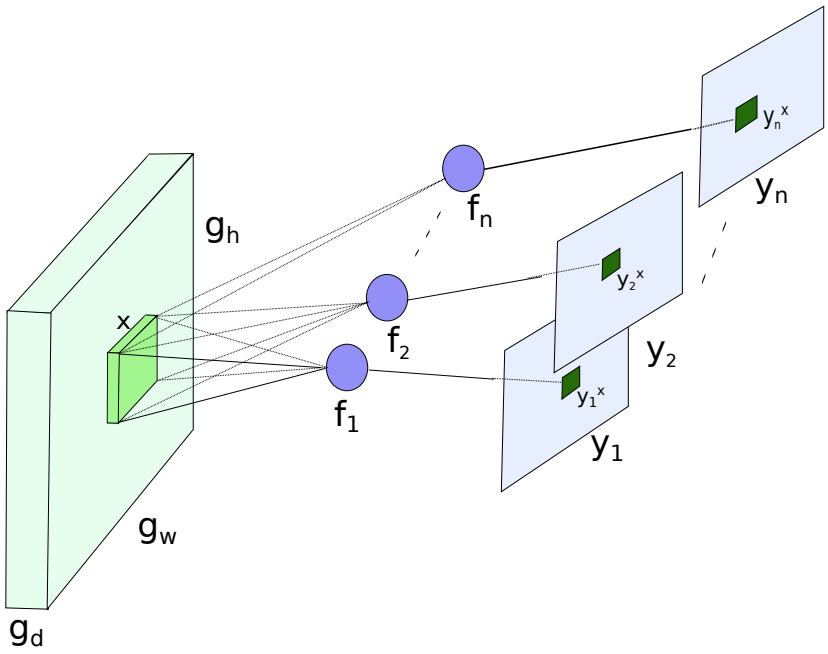


Figure 3.4: 3D Convolution

the convolution as following

$$h_i^x = W_i\, x + b_i \tag{3.6}$$

where b_i is some offset value carried by the neuron, and W_i and x perform dot product.

The output h_i^x is further applied on an activation function, which throws off value that isn't good enough to pass some threshold. In this model we use rectifier function, which simply cuts off negative h to zero. A unit performing rectifier function is known as *rectified linear unit* (*relu*). Hence, the output of the neuron is

$$y_i^x = relu(h_i^x) = max(0, h_i^x) \tag{3.7}$$

which is the dark green patches on right. Since the neuron not only performs convolution on input regions, but also discards certain values, it's essentially a filter that cuts off bad result. Hence neurons are also called filters. In the sections that ensue, we will abuse this terminology, and call neurons, kernels, filters interchangeably.

We slide the volume x across the height and width plane (or spatial plane) of input in a left-right and top-down fashion, convolve each volume with filters, and obtain what is known as the feature

map y_i for each filter f_i. The feature map is simply the conglomerate of all possible y_i^x, and it represents features as seen by filter f_i due to convolution, hence the name.

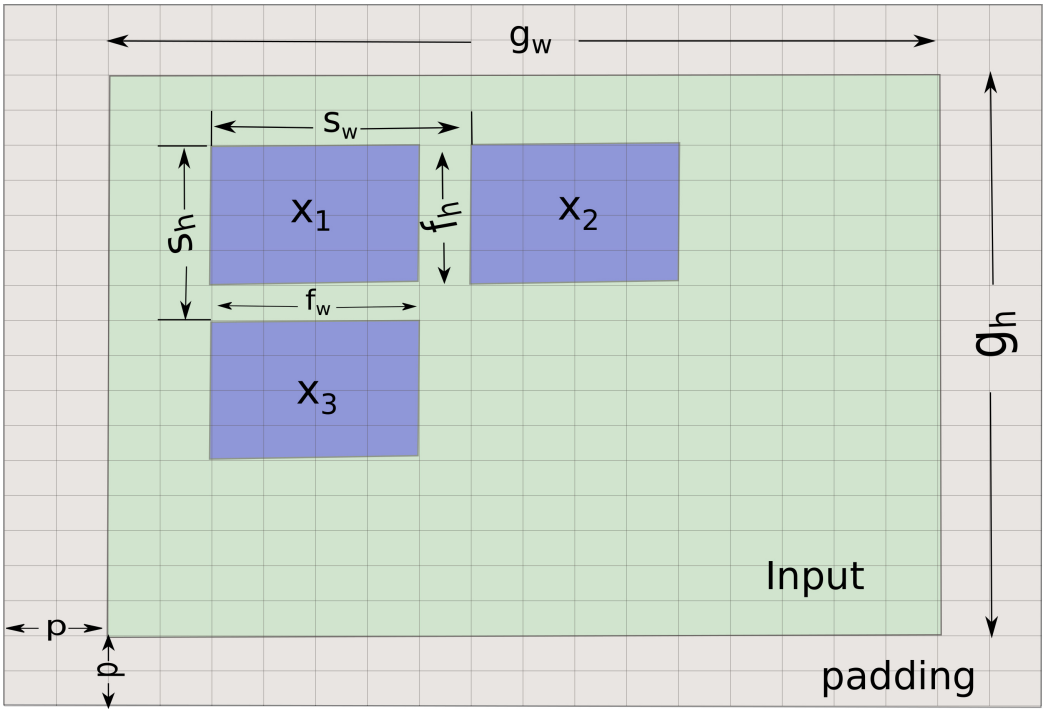


Figure 3.5: Sliding Across Input Volume

There are many ways to slide the volume x across the spatial plane, as mandated by the sliding parameters s_h and s_w, the length of one stride along height and width axis. They control the granularity level on which to perform convolution. As shown in Figure 3.5, volume x_2 is the next to-be-convolved region given a horizontal stride from x_1, and x_3 is the next to-be-convolved region given a vertical stride from x_1. If $s_w < f_w$, there is overlap of regions between two consecutive convolution horizontally. If $s_w \geq f_w$, no overlap of regions between two consecutive convolution can happen horizontally. The same is so true vertically with s_h and f_h. Generally, in complicated tasks where one doesn't want to miss any patterns, it makes sense to set both horizontal and vertical stride to be 1, so that it convolves with every neighboring region. Otherwise, one can set more lenient stride values to lessen computation burden and save training time.

There is padding (of zero) outside the input region, because we don't want to miss the convolution centered around some of the edge positions. Generally, we make horizontal padding and

vertical padding the same, denoted as p in the figure. For any filter f_i of shape f_h and f_w, and stride parameters s_h and s_w, let y_h and y_w denote the height and width of output feature map y_i obtained by convolving f_i with the entire input g with padding p. We have

$$y_h = \frac{g_h - f_h + 2p}{s_h} + 1 \tag{3.8}$$

$$y_w = \frac{g_w - f_w + 2p}{s_w} + 1 \tag{3.9}$$

In summary, the convolution (and activation) layer of n filters, takes an input of shape $[g_h, g_w, g_d]$ and produces n feature maps of shape $[y_h, y_w]$ by convolving each filter with the sub volumes of the input in a way determined by each filter's shape and stride parameters. The output feature map is further fed into the next layer, which we shall explain next.

3.2.2 Max Pooling Layer

The *Max Pooling Layer* selects the most important features seen from previous convolution layers. Another effect of this layer is to downsample the input volume spatially, because other lesser patterns are discarded.

For all the feature map y_i generated from previous convolution layer, the max pooling is parametrized by a pooling window of shape $[p_h, p_w]$, the height and width of the window, and stride value of shape (ps_h, ps_w), the stride length along height and width axis. For any region $y_i^x \in y_i$ of shape $[p_h, p_w]$, the max pooling selects the maximal value in the region y_i^x.

$$z_i^x = \max_{t \in y_i^x} t \tag{3.10}$$

The output of the max pooling area is just a conglomerate of selecting maximal values in all possible y_i^x regions, sampled by window movement as specified by the stride values ps_h and ps_w.

The idea is clearly expressed in Figure 3.6, where a pooling window of $[p_h, p_w] = [2,2]$ and stride value of $(ps_h, ps_w) = (2,2)$ is used. The output z selects the maximal value in each sampled region. Intuitively it serves the purpose of singling out the most important feature in that window.

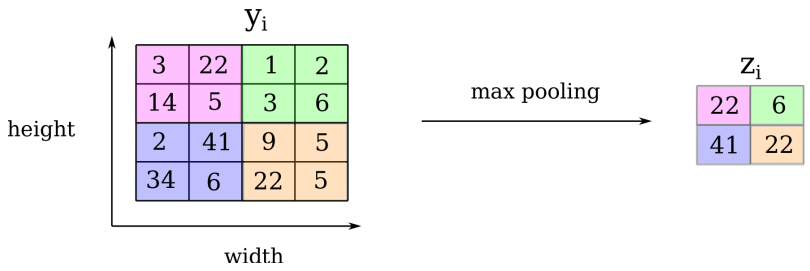


Figure 3.6: Max Pooling with Window Size [2,2] and Stride (2,2)

The max pooling layer contains no parameters to be learned. Unlike the convolution layer, which leaves W_i and b_i to be figured out by the learning process, there is no variables involved in any pooling window. The only parameters are shape and stride parameters. Same as the shape and stride parameters of convolution filters, they are hyperparameters, which are set arbitrarily before training.

In addition, different from convolution layer which uses zero padding, there's normally no padding for max pooling because it computes a fixed function of the input. The pooling also extends through the full depth of its input, i.e., for all y_i, the same pooling window parameters is used. Hence the pooling is like a max value selection in spatial plane, but independent in the depth slice of input.

Following our previous notations, let there be n feature maps $(y_1, ..., y_n)$ produced in the convolution layer, and are fed to the pooling area. Let $[y_h, y_w]$ be the shape of feature map y_i, the max pooling produces an output z_i of shape $[z_h, z_w]$ according to the following equation, and there are n such z_i produced and fed to the next layer.

$$z_h = \frac{y_h - p_h}{ps_h} + 1 \tag{3.11}$$

$$z_w = \frac{y_w - p_w}{ps_w} + 1 \tag{3.12}$$

3.2.3 Everything Put Together

Figure 3.7 illustrates the core component of our approach. x_e and x_c, which are the document representation after word embedding as described in Section 3.1, are taken as input and shown in

an x-y-z 3D space. The first dimension x represents the number of words in a document. The second axis y is the word embedding dimension d. The third dimension is the number of channels. x_e and x_c are respectively fed into *entity subnet* and *context subnet*, which is independent of each other.

Each subnet consists of multiple layers of convolution, activation and max pooling. Each convolution (activation) layer with a following pooling layer constitutes one layer of *ConvPool*. The *ConvPool* can be used for several times, and the specific number is a hyperparameters set in the experiment. Inside the subnet, each layer will witness varying number of convolution neurons. Since words are embedded into the high dimension space as a whole, it doesn't make sense for neurons to scan the dimensional space fractionally. Therefore, we make the convolution window $f_w = d$ for all neurons in layer 1 and $f_w = 1$ for all subsequent layers. Similarly, we set $p_w = d$ for max pooling in the first layers and $p_w = 1$ in subsequent layers. Besides, since the neuron only strides the input along x axis, it means $s_w = 0, ps_w = 0$. The f_h is an arbitrary hyperparameter. If f_h is chosen as 1, it indicates that only one word is covered in the convolution window, hence it simulates uni-gram model in NLP. If f_h is set to 2, it represents bigram, and so on.

A common pooling window of $p_h = 2$ will select the most important feature in two neighboring positions, and reduce the input by half along the x axis. As we go deeper in layer, we are actually observing a wider window of the input for more sophisticated patterns, with the lesser features in a narrower window already filtered by previous pooling layer. For instance, if unigram is used in conjunction with a 2-max pooling, in the first layer, the network is looking at important features in a size-2 window. In the second layer, the network looks at whatever size-2 neighboring features elected from layer 1, each of which is a result of previous size-2 window comparison with its neighbors in the input. Hence, the network is actually looking at size-4 window of the input, because the overall computation spans at most four words. That said, with more layers in the network, it has the capacity to uncover sophisticated n-gram patterns at the later stage of the network, by making use of shallow, short patterns discovered in preceding layers.

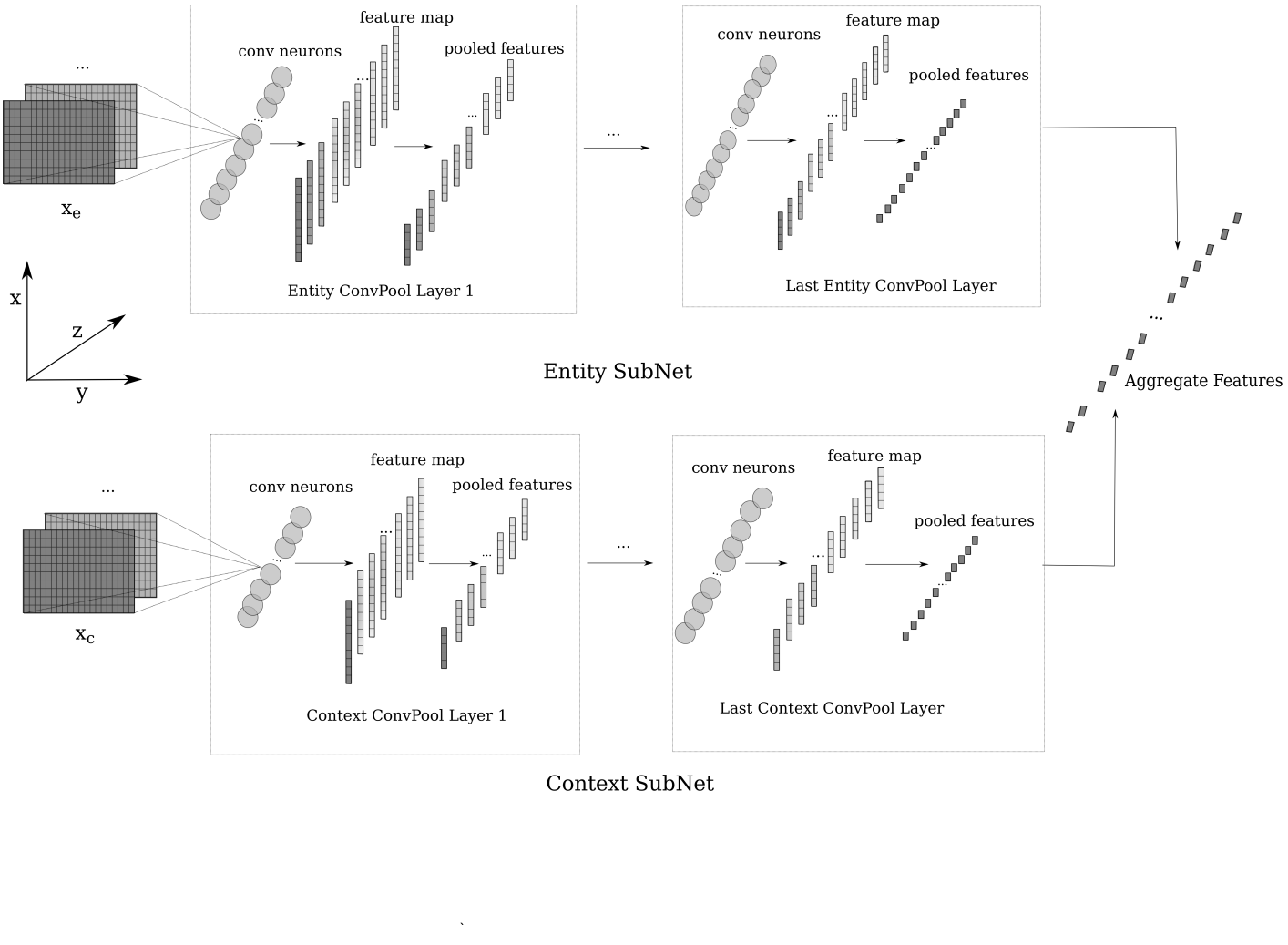


Figure 3.7: Convolution and Max Pooling Applied to Embedded Entity/Context Input

We make max pooling in all intermediate layers use the same fixed window size such as $p_h = 2$. In the pooling of the last entity/context ConvPool layer, we take whatever is left in the feature map, and apply a special max pooling that ends up with only one most important feature per neuron. Specifically, let the feature map of last layer be of length $\hat{y}_h$ (along the height axis), the max pooling size used in the last layer is then $p_h = \hat{y}_h$. We want everything to be covered in one pooling window so that we can select the most important features from previous convolutions (be it bigram feature or some mysterious n-gram feature).

In the last, we concatenate the last layer pooling results from both subnets, and make it our aggregated features. The aggregate features are fed into the next fully connected layer for classification and regularization purpose, which we shall elaborate next.

3.2.4 Fully Connected Layer and Dropout

The aggregate features represent all the important features selected by the neurons in the convolution network. One can think of neurons as templates for features, with each neuron looking for a unique feature. The combination of all of them contains information as to the presence/absence of certain clues critical for entity linking.

Figure 3.8 shows the structure for the last component of our network architecture. The aggregated feature is connected to a fully connected layer, where each neuron in that layer is connected to all features, and each feature is connected to all neurons in the fully connected layer. The neurons in the fully connected layer are then connected to all entities in the last layer.

Let z^* denote the aggregate features, $\widetilde{W}$ denote the weights between aggregate feature layer and fully connected layer, and $\widetilde{b}$ denote the bias for every neuron in the fully connected layer, the output (without dropout) of neurons in the fully connected layer $\widetilde{z}$ is then

$$\widetilde{z} = relu(\widetilde{W}z^* + \widetilde{b}) \tag{3.13}$$

Dropout [47] is a technique used to prevent overfitting and provide regularization. It does so by

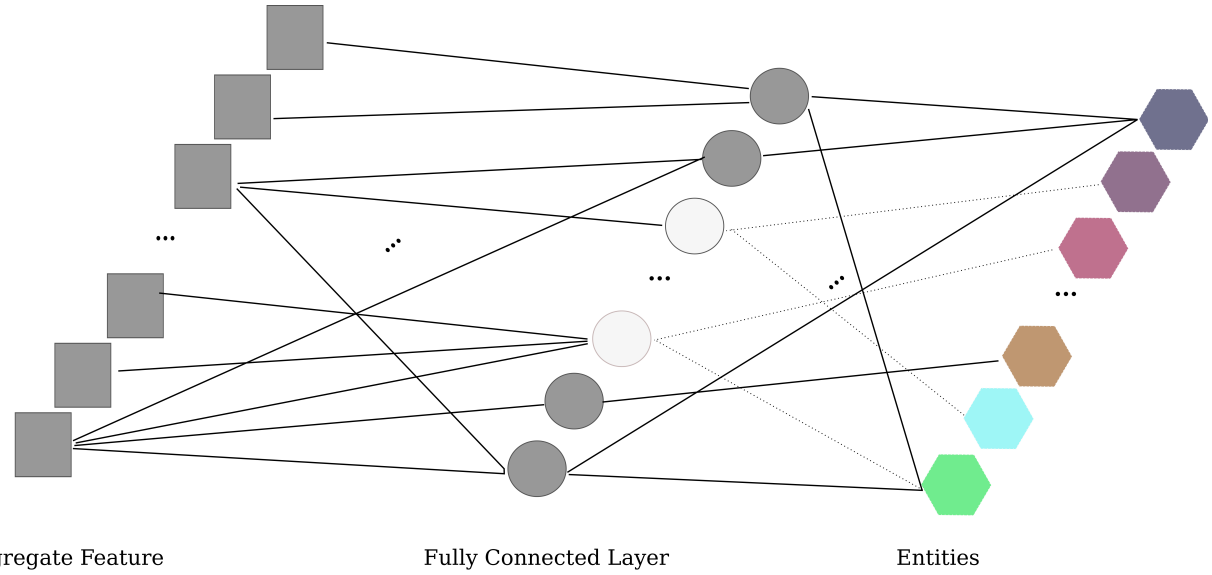


Figure 3.8: Fully Connected Layer and Dropout

randomly dropping some units in the network during training. This creates different sampling of "thinned" (subset of original) network from an exponential number of different choices, with each minibatch likely trained on different "thinned" networks. Therefore, under the original network lies different "thinned" networks, and we can use one model to simulate multiple choices. In test time, no units are dropped out, and prediction is made on the original network. By doing so, we make use of all underlying "thinned" networks, and literally perform an averaging of all the predictions. This echoes with the principle of "regularization", whose most ideal situation is to compute the weighted average of the predictions of all possible settings of the parameters, with weights set according to each setting's posterior probability given the training data.

We apply the technique of dropout in the neurons of fully connected layer. Let the dropout keep probability (the probability of surviving dropout) of each neuron being p. Then

$$r \sim Bernoulli(p),$$

$$\hat{z} = r * \widetilde{z}$$

where r is a binary vector of length equal to the number of neurons in the fully connected layer. It takes value 1 with probability p, and 0 with probability $1-p$. Multiplying r with $\widetilde{z}$ will drop output of neurons that fail to be selected in the Bernoulli experiment. The real output for the fully

connected layer, after applying dropout, is $\hat{z}$.

The next step is to feed value $\hat{z}$ to the last entity layer. Let s be the score of entities, W_f be the weight between fully connected layer and entity layer, and b_f be the offset for each entity in the entity layer. We have

$$s = W_f \, \hat{z} + b_f \tag{3.14}$$

Note there's no activation function, because we're basically performing a linear classifier here, which requires no non-linearity (activation is used for introducing non-linearity).

Usually s is called the *logits*, and is treated as unnormalized log probability of the probability distribution among entities. To get the probability, one can apply softmax function on the score s. Let p_e denote the probability distribution for all entities predicted by our model. Then

$$p_e = softmax(s) \tag{3.15}$$

In training, we can get loss values using p_e and ground truth entities. The target during training is to minimize the loss values over iterations. We can apply stochastic gradient descent to optimize the training target. In test, we use the latest trained variables to get p_e, and select the entity that has the highest probability among p_e as the predicted entity to be linked.

We list all variables in our model in Table 3.1 and hyperparameters that were discussed before in Table 3.2. There are other hyperparameters, such as the number of neurons in each layer, we will discuss them in experiment later.

Variable	Description
EW_n	the embedding table of n-th channel. EW_1 is initialized from a separate Skip-gram embedding training, EW_2 and above from some other sources. All can remain static in training or change over training. EW_2 and above has to be learned in training.
W_i, b_i	the weight matrix and offset carried by convolution filters. Each filter has different W_i and b_i, and they are to be learned in training.
$\widetilde{W}, \widetilde{b}$	the weight matrix and offset carried by the fully connected layer neurons, to be learned in training.

Table 3.1: Summary of Variables

Hyperparameters	Description
Embedding channels	number of channels used in embedding; determines the depth (z axis) of input to convolution network.
Embedding dimension d	dimension used in word embedding; determines the width (y axis) of input to convolution network. We use 50 in our model.
$[f_h, f_w]$	height and width of convolution filter window. f_w is set to d in the convolution layer of the first ConvPool, and 1 for all subsequent convolution layers. f_h indicates the number of words covered in a window, can be tuned in experiment, specific to each filter.
(s_h, s_w)	the stride values of convolution filter window along height and width axis. $s_w = d$ for the convolution layer of the first ConvPool and $s_w = 1$ for subsequent convolution layers. s_h can be tuned in experiment, specific to each filter.
$[p_h, p_w]$	height and width of max pooling window. p_w is set to d in the pooling layer of the first ConvPool, and 1 for all subsequent pooling layers. p_h indicates the number of words covered in a window, set $p_h = 2$ for all pooling in non-last ConvPool layers, and set p_h to length of last feature map in the last pooling layer.
(ps_h, ps_w)	the stride values of max pooling window along height and width axis. $ps_w = d$ for pooling of the first ConvPool layer and $ps_w = 1$ for subsequent layers. Set $ps_h = 2$ for all pooling layers.
p	dropout keep probability in Bernoulli experiment, usually set to 0.5

Table 3.2: Summary of Aforementioned Hyperparameters

Chapter 4

DATASET AND ENVIRONMENT

4.1 Tools and Environment

4.1.1 TensorFlow

We deploy our work using TensorFlow, which is a machine learning platform developed by Google. The learning process can be embedded in a graph. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows us to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

4.1.2 Word2vec

Word2vec (w2v) is a continuous distributed representation of words. It produces word embeddings such that each word has a vector in the space, and words with similar contexts are located in close proximity to one another in the space. Unless further specified, we used 50 dimension space *w2v*, which contains 400k vocab and was trained over 6 billion tokens from Google News dataset, across the entire experiment.

4.1.3 CUDA

CUDA is a GPU computing platform that drastically increases computing performance, especially in the case of neural network, where millions of parameters need to be computed efficiently. In this experiment, we used *CUDA* 7.5 available from NVIDIA website.

4.2 Dataset and Experiment setup

4.2.1 Dataset

The work was examined over the dataset provided by UIUC. It consists of archived news articles from MSNBC, an American based agency that provides news coverage and political commentary.

Table 4.1: Summary of the Dataset

Datasets	Number of Distinct Entities	Test Cases
MSNBC	271	625

The statistics of the dataset is displayed above. There are 271 different entities to be disambiguated, and 625 entity instances in total to be linked. Each entity instance is a person, a place or an event, to which there is a corresponding Wikipedia entry associated. In other words, all instances can find its way to the knowledge base and none of them is unlinkable.

4.2.2 Example of Test Data

Figure 4.2 illustrates an example of the test cases. The test case is a piece of business news about Bob Nardelli, who is a CEO of Home Depot Inc. The entity to be linked with Wikipedia entry is highlighted in red. The remaining text are context to be used for disambiguating *Bob Nardelli*. It's at the discretion of implementation to determine the size of the context window. Note that there could be multiple instances to be linked in one piece of news. For instance, *Home Depot* could also be an instance to be linked with the wikipedia entry The Home Depot.

Home Depot CEO Nardelli quits

--Home-improvement retailer's chief executive had been criticized over pay

ATLANTA - Bob Nardelli abruptly resigned Wednesday as chairman and chief executive of The Home Depot Inc. after a six-year tenure that saw the world's largest home improvement store chain post big profits but left investors disheartened by poor stock performance.

Figure 4.1: Example of a Test Instance

4.2.3 Problem Description

The *problem descriptions* are recorded in separate XML files which specify, in each news file, position the entity appears at, the canonical name, and the link to its Wikipedia entry. The XML

```
<ReferenceProblem>
  <ReferenceFileName>
    Bus16451112.txt
  </ReferenceFileName>
  <ReferenceInstance>
    <SurfaceForm>
      Bob Nardelli
    </SurfaceForm>
    <Offset>
      117
    </Offset>
    <Length>
      12
    </Length>
    <ChosenAnnotation>
      http://en.wikipedia.org/wiki/Robert_Nardelli
    </ChosenAnnotation>
    <NumAnnotators>
      1
    </NumAnnotators>
    <Annotation>
      Robert Nardelli
    </Annotation>
  </ReferenceInstance>
</ReferenceProblem>
```

Figure 4.2: Problem Description

file contains several key fields: *SurfaceForm* identifies the name under which the entity appears in the text; *Offset* and *Length* tells the position of the first character of the entity with respects to the beginning of the news, and the length of the surface name; *Annotation* is the canonical name of the entity, and *ChosenAnnotation* is the link to its associated Wikipedia entry.

4.2.4 Test Data Preparation

The raw test data is first normalized. We removed all characters that are not alphanumeric, single quote or bracket. White spaces, if more than one between words, are truncated so that words are separated by only one space. Abbreviation such as “I’ll” is expanded to “I ’ll” by adding a space between them. Similar changes are also done to “I haven’t ” so that it becomes “ I have n’t ”. In all cases, the abbreviation is separated to become an independent token.

Different from traditional NLP preprocessing, which requires stemmization (change derived word to their root word such as changing *makes* to *make*), and removing stop words (the most common words in language such as *the*, *which*, and *who*), we are saved of the trouble, because derived words are close to their root words in the space after word embedding. Removing stop words is unnecessary because the neural network could automatically figure out features that matter. As a result, stop words, which occur commonly in many articles, won’t be a key feature extracted by the neural network.

To facilitate data read operation and better organize the test, we convert test cases to the form of *TFRecords*, which is a serialized string containing *features* abiding by the *Example* protocol buffer in TensorFlow. Each feature is a key value pair, which allows the storage of large amounts of typed data.

Figure 4.3 shows a *TFRecord* of the news snippet above. *Entity* is the surface name of the entity to be linked. *Context* is the words surrounding the entity, and the context window is decided arbitrarily at runtime. Note the original surface name is replaced with underscore. This is a dummy placeholder to represent where the entity was in original text. Both of them, being strings, are stored as bytes list. *Label* is an integer value signifying the index of the ground-truth annotation. For example, “Bob Nardelli” is the 123rd entity among all the 271 entity labels.

```
features {
    feature {
      key: "entity"
      value { bytes_list {
        value: Bob Nardelli
      }}
    }
    feature {
      key: "context"
      value { bytes_list {
        value: "home depot ceo nardelli quits home
                improvement retailer 's chief executive
                had been criticized over pay atlanta ___ abruptly
                resigned wednesday as chairman and chief
                executive of the home depot inc.  after a six-year
                tenure that saw the world 's largest home
                improvement  store chain post big pro ts but left
                investors disheartened by poor stock performance."
      }}
    }
    feature {
      key: "label"
      value { int64_list {
        value: 123
      }}
    }
}
```

Figure 4.3: TensorFlow Record

4.2.5 Training Data Preparation

There's no training dataset accompanying the test dataset. Therefore, we need to manually create training data in order for our neural network to pick up entity knowledge. There are two sources where we gather our training corpus from. We grab contents from entities' wikipedia entry, and break them into fixed size snippets. Intuitively, the articles in wikipedia is a direct description of the entities itself and should be good materials for the neural network to understand the entities. Additionally, we built a web scraper to automatically crawl Google news and augment our Wikipedia dataset. Between 100 to 500 news articles were obtained for each entity, depending on the search result.

The first step may be just admitting there's a problem. If you look at the GOP as a business, says Bob Nardelli, a former CEO of Home Depot and Chrysler, "the CEO and the board have been somewhat in denial about the situation the company is facing." For example, the party barely tweaked its approach after Romney's presidential trouncing in 2012 even though elements, like more trade deals and tax cuts for the rich, proved very unpopular with customers...er, voters.

Bob Nardelli, former CEO of Home Depot (HD) and Chrysler. Named one of the "Worst American CEOs of all time" by CNBC after sending Chrysler into bankruptcy, Nardelli served as interim CEO for gunmaker Freedom Group and director of NewPage Corp. He recently stepped down as CEO of the operations and advisory company at private equity giant Cerberus Capital Management, which owned all those companies

Figure 4.4: Example of Augmented Google News as Training Corpus

Figure 4.4 shows two examples of the augmented training data for the entity *Bob Nardelli*. After obtaining the training corpus from Wikipedia and Google news, we converted them to *TFRecord* form in same way as the test dataset.

Chapter 5

RESULT

5.1 Training and Validation

5.1.1 Loss and Accuracy

Loss is a metric to measure in our experiment. It roughly reveals how confident the model is about the prediction result. Generally, the loss function represents the cost paid for inaccurate prediction result, given the batch sample. We sought to minimize the loss function, and in this case, we adopted entropy loss function given below in 5.1

$$loss(p_e, p_t) = -\sum_{x \in D} p_t(x) \cdot \log p_e(x) \tag{5.1}$$

p_e is the probability prediction by our model, p_t is the probability distribution according to the ground truth, or the target distribution, and x is a document in the training set D. $p_e(x)$ gives the probability prediction for all entities for current document x. In our case, since there's only one true entity for a given text, $p_t(x)$ has probability 1.0 on the correct entity of document x, and probability 0 for all other entities.

Given that, the loss function measures the aggregated probability of each document on their ground truth entity class, since other irrelevant class has a p_t of zero. The higher the probability on the right class, the lower the loss value (note the negative sign before summation). The optimization process will hence penalize documents that have lower probability on its correct class more. Neurons associated with the documents with an already high probability on its correct class will receive less, if there is any, adjustment. Since the loss function is also continuous, gradient descent based optimizer can be employed, which is readily available.

Accuracy is the other and the most important metric in our experiment. It measures the per-

centage of test instances that are linked to the correct entity class according the ground truth. It's also the benchmark that can be used to compare different models.

5.1.2 Training Set and Validation Set

The core with neural network is its ability to generalize. In some cases, the network may fit extremely well to the training data, but would fail to work with test corpus.

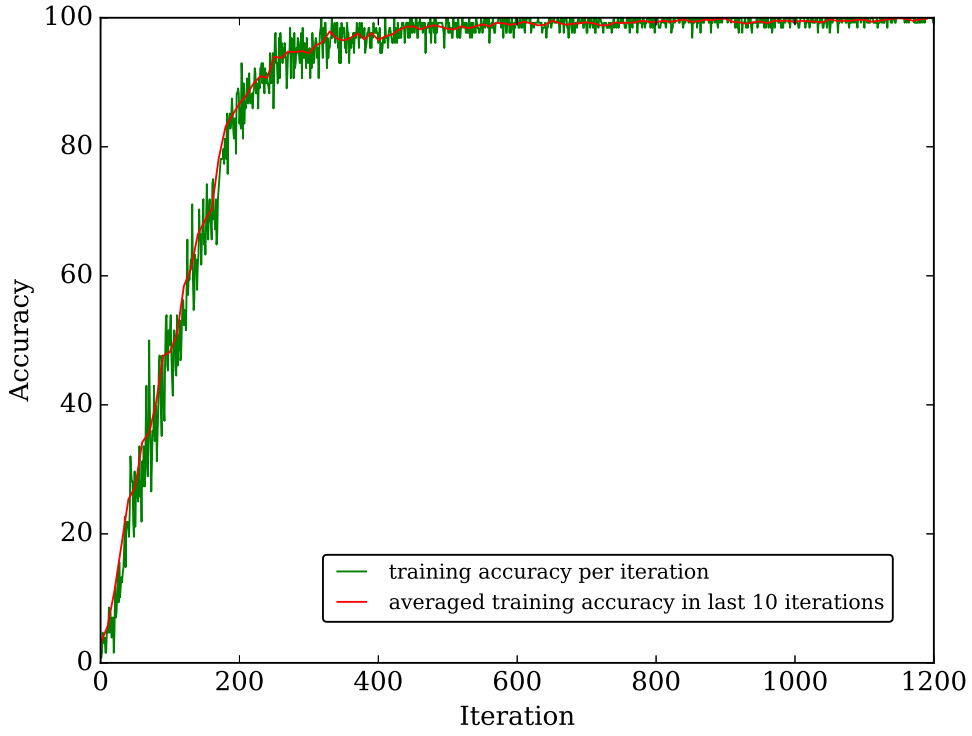
To monitor this issue, we divide the training data into training set and validation set. Note that the validation set is from our collected training corpus, but never participates in the training phase. Instead, it serves the purpose of test data, which is unseen from the perspective of training. Hence validation can help reveal how well our neural network is able to predict unseen data.

Our entire training time is interleaved with validation phase, where we just feed one minibatch of validation data, and use the existing parameters trained so far in the network to predict the validation data. Specifically, we collected accuracy on the training minibatch every step, and performed validation every 10 steps.

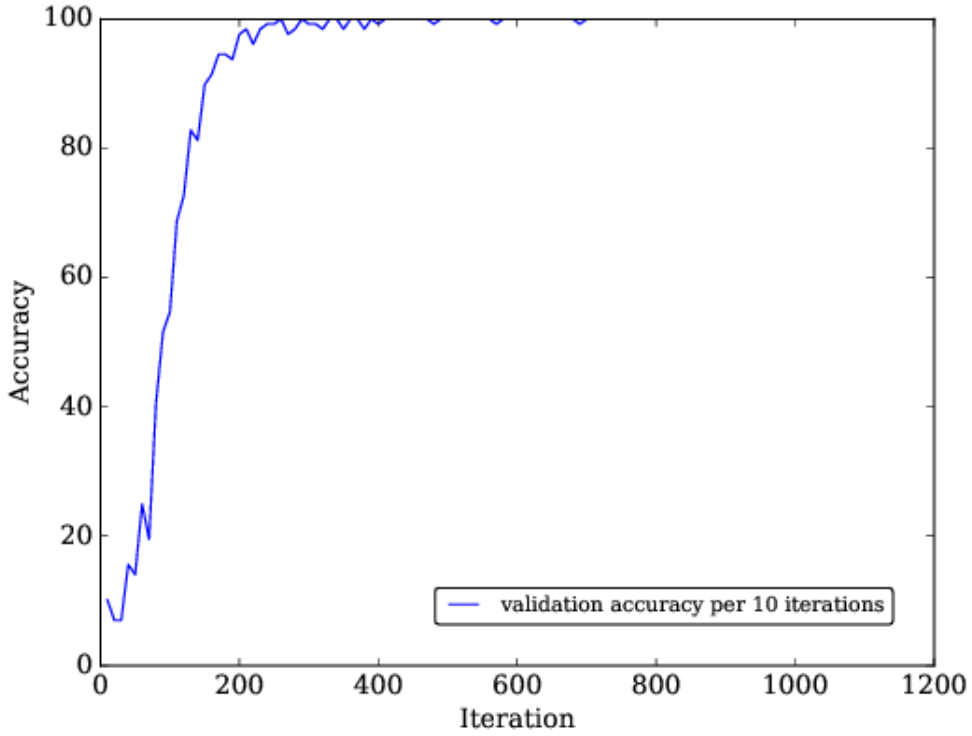
Figure 5.1 shows the accuracy result collected as described above. Figure 5.1a plots the accuracy in each step with green curve. There's some degree of oscillation as the model is adjusting its neuron weights. However as a general trend, we see that the accuracy in training gradually improves until getting flat around 500 steps. The red curve plots the average accuracy in latest 10 iterations, it is a much more smooth line showing the average result in latest training, and it gradually moves up until becoming stable around 500 steps as well.

Figure 5.1b shows the accuracy in validation dataset. In a similar fashion to Figure 5.1a, the accuracy increases significantly in the initial 400 steps. After that, the model is able to achieve an accuracy close to 100%. This shows that our model is able to predict very well in the unseen data, and is very promising if applied in the real test dataset.

Similarly, we plot the loss value in training and validation phase as well in Figure 5.2. Figure 5.2a shows the loss value of training in each step in green curve, and the average loss value in

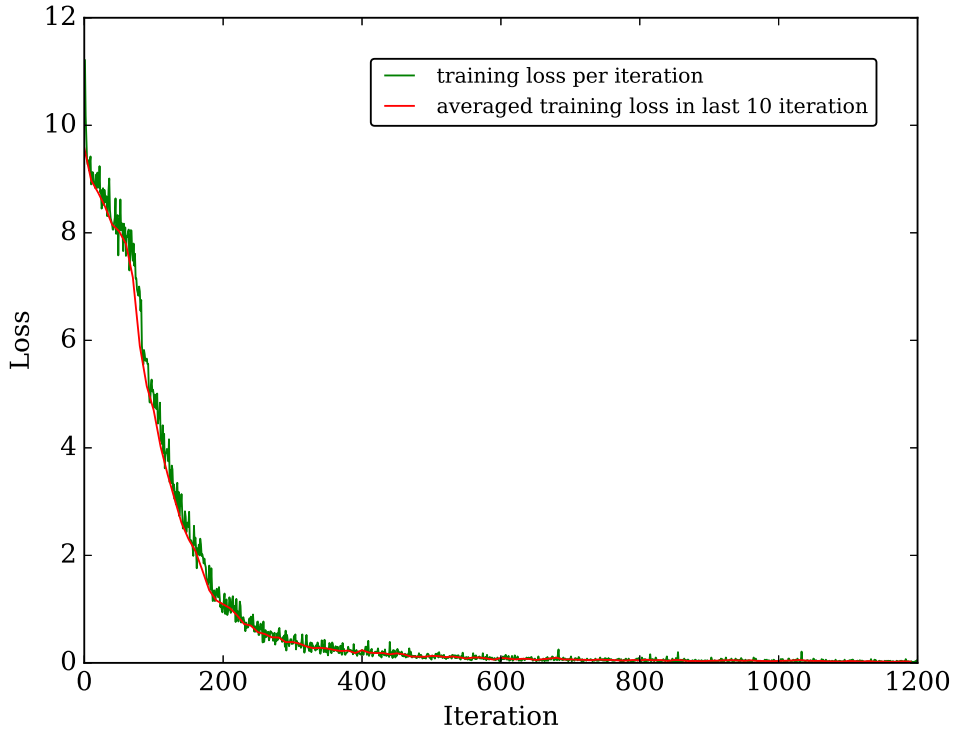


(a) Accuracy in Training

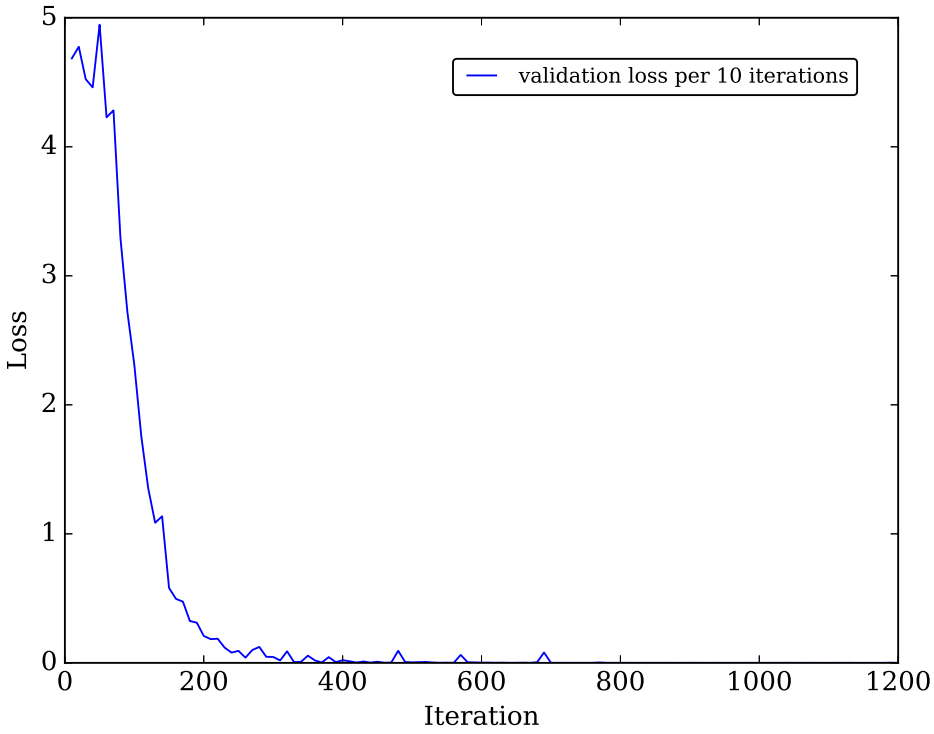


(b) Accuracy in Validation

Figure 5.1: Accuracy in Train and Validation



(a) Loss in Training



(b) Loss in Validation

Figure 5.2: Loss in Training and Validation

the latest 10 iterations is drawn in red. Again, there is some vibration leading to 400 steps, partly due to the fact that the optimization algorithm may have overshoot, and was trying to change its course back to the right direction. Overall, the training loss is reduced at the later stage of training, suggesting that the network is getting mature at fitting data.

Figure 5.2b plots the loss in validation, it follows a similar fashion to that of Figure 5.2a. The loss value decreases quickly in the initial 400 steps and becomes stabilized after that. We notice that the stabilized value is very close to zero. This indicates that our model has very good confidence even in predicting unseen data, and should be very promising when applied to the test dataset. It confirms the high accuracy achieved in Figure 5.1, and demonstrates a very strong ability: not only the most probable entity is predicted correctly by our model, but also it has close to 1.0 probability on the right class. It shows that our model is extremely confident. Otherwise, the aggregated loss value cannot be close to zero according to the definition of entropy loss in 5.1.

5.2 Variants of Baseline CNN

We next investigate variants of our baseline CNN. We evaluate the dataset on different configurations, and see how different hyperparameters could have varying impact on the performance. We seek to find the optimal configurations that lead to the best accuracy, and from this point on, all the accuracy refers to those achieved in the test dataset, not the validation or training dataset. We trained the model using 1000 iterations, and test the neural network with the latest gained parameters (weights and biases).

embedding channel configuration	entity filter size	number of entity filters	context filter size	number of context filters	ConvPool layers of entity subnet	ConvPool layers of context subnet	number of neurons in fully connected layer	dropout keep probability
two nonstatic channels	2	1024	5	512	1	1	1024	0.5

Table 5.1: Default Hyperparameters Configuration

The hyperparameters mentioned in 3.2 are set accordingly because they are predetermined part of the model. We only investigate hyperprameters that are not set previously, because they bring about variants of the baseline model and introduce varying entity linking capacity. Table 5.1 summarizes the hyperprameters we will study. In each of the following subsection, we investigate the impact of one hyperprameters by assigning it different values, while keeping the others constant. Unless it's specifically investigated/mentioned, the hyperprameters adopt the default configuration in Table 5.1.

5.2.1 Embedding Channels

We start by discussing the impact of embedding channels. There are different channel configurations. Single channel is the one where we only have one table of word embedding, whereas double channels adds an extra layer of embedding table, so that the lookup tables are stacked depthwise.

We can initialize embedding in two ways: one via an established, well-known embedding word2vec (w2v); the other is via random distribution. Over the course of training, we can keep the table unmodified/static. We can also keep updating the embedding parameters, and make the table part of our training variables to optimize over. We call this approach nonstatic.

Configuration	Description	Accuracy
Single Channel 1	randomly initialized, static	33.5%
Single Channel 2	w2v initialized, static	74.3%
Single Channel 3	w2v initialized, nonstatic	84.5%
Single Channel 4	randomly initialized, nonstatic	72.2%
Double Channel 1	channel 1 w2v initialized and static, channel2 randomly initialized and static	53.2%
Double Channel 2	channel 1 w2v initialized and static, channel2 randomly initialized and nonstatic	87.6%
Double Channel 3	channel 1 w2v initialized and nonstatic, channel2 randomly initialized and static	82.3%
Double Channel 4	channel 1 w2v initialized and nonstatic, channel2 randomly initialized and nonstatic	**92.6**%

Table 5.2: Channels Effect on Accuracy

Table 5.2 summarizes the accuracy obtained with different aforementioned channel configurations. We noticed several things. First of all, single channel 1 has the worst result amongst all. This is expected because the table is randomly initialized. It doesn't convey effective message about words, and yet it's prohibited from learning along the way. It's therefore the least desired model.

Secondly, in single channel configurations, initializing from w2v and optimizing the table over training achieves the best result. It indicates that, starting from a reliable source of word embedding and customizing it over training to fit particular use case, is the recipe for better disambiguation. This is also seen in the result produced by double channels configuration, with double channels configuration 4 topping the comparison.

In addition, double channel configurations seems to outperform single channel configuration in most cases, indicating the benefit of providing more words information. The only two exceptions are double channel 1 and double channel 3 (worse than single channel 3). In these two cases, channel 2 are both randomly initialized, and not allowed to evolve over time. Hence although one more channel is provided, it's actually backfiring by not providing effective information.

Last but not least, double channel 4 achieves the best result among all configurations. This is because it takes advantage of reliable embedding table, optimization over training, and an extra source of information.

5.2.2 Convolution Filter Window Size (f_h)

We conduct evaluations on how filter window sizes could have different impact on accuracy. We've used different combinations of window sizes for entity and context texts. Because of the findings in the last subsection, we use two channel configurations (one initialized from word2vec and the other initialized randomly, and both channels are allowed to optimize over the training) in all following subsections. The results are summarized below in Table 5.3.

First of all, we noticed that the combination of entity window size of 1, and context window

of size 2, 3, and 4 achieved the best result. It seems a little counter-intuitive to see how a small entity window size, which is essentially a uni-gram model, achieving this result. We attributed it to common patterns in our test cases. Many of the entities in tests are one word entity. For example, in the *Bob Nardelli* case, one of the test instance goes like this: "**Nardelli** has also been under fire by investors for his hefty pay and is leaving with a severance package valued at about 210 million". The highlighted word is the surface name to be disambiguated, and yet it consists of only one word. Although the training corpus includes the full name of Nardelli (*Robert Nardelli*), the neural network seems to prefer learning the single most important word in that name. Hence the size 1 filter window in entity seems to work pretty well in this test dataset.

The second best window size is the combination of entity window size of 1 and 2, and context window size of 2 and 3. It loses to the best combination only by a slight margin. Yet this combination includes an entity window of 1 and 2, which caters to many cases where surface names are two words long.

In addition, we noticed that there's no good in having too large window sizes for entities. For example, the models where entity window size is purely 5 has the worst performance among all. This is intuitive in that, the neural network is trying to find the most important 5-gram feature of the entity, which is a linear combination of the words in the 5-gram frame. The real differentiating words in the 5-gram may be diluted as a result of this over generalization. In contrast, models that consist of a combination of entity window size 5 and other entity window sizes perform better than a pure 5-gram. This is because there are windows of finer granularity to discover important 2-gram or 3-gram, which may override the over-generalized 5-gram discoveries in the fully connected layer. As a result, the model is more reliant on these finer-granularity patterns.

On the other hand, the patterns in context window size is less clear. There are cases where a larger window size produced better results and also cases where larger window sizes achieved a lower accuracy. For example, in the model of entity window size 1, and context window size 2 and 3, we see an increasing accuracy when adding another context window of size 4. However,

	Context Window Size		
Entity Window Sizes	2,3	2,3,4	3,4,5
1	87.50%	**90.20**%	88.60%
2	86.20%	86.10%	81.90%
3	78.40%	78.70%	74.60%
4	76.60%	77.00%	78.20%
5	73.40%	71.50%	74.60%
1, 2	89.80%	86.90%	84.80%
2, 3	85.40%	85.60%	84.60%
3, 4	77.80%	79.70%	77.80%
4, 5	78.40%	80.00%	74.60%
2,3,4	80.50%	82.90%	81.40%
3,4,5	82.10%	82.60%	78.70%
1,2,3,4	83.70%	85.00%	82.40%
2,3,4,5	81.10%	82.90%	85.10%
1,2,3,4,5	85.40%	83.50%	82.40%

Table 5.3: Filter Window Size

replacing the context window of 2 with a context window of 3 caused the accuracy to drop from 90.2% to 88.6%. Hence, it's a much more intricate question as to which context window is the best, as there's no straightforward patterns observed here.

5.2.3 Number of Filters

We next investigate the interaction between accuracy and number of filters used in the neural network. We used the entity filter size of 1, and context filter sizes of 2, 3 and 4, which is the best scenario configuration uncovered in the last subsection. We made filter quantity range from 32, 64, 128, 256, 512, 1024, and 2048, and make all entity filters of size 2, 3 and 4 share the same quantity in one test. So in one configuration, there could be 32 size 1 entity filters and 128 size 2, 3, and 4 context filter each. In another, there could be 2048 size 1 entity filters and 1024 size 2, 3, and 4 context filters each. We don't differentiate quantity of different-in-size context filters

because there's no evidence that this minor detail will significantly improve accuracy, and we want the granularity control to stay in a manageable level.

Table 5.4 summarizes all the configurations used in this experiment and their result. It shows that the configuration of 2048 entity filters and 64 context filters each of size 2, 3, and 4 gave us the best accuracy. This makes sense as we have many entity filters to figure out the pattern in surface names, while a reasonable number of context filters aids with context to disambiguate.

In contrast, it's a little interesting to see that the configuration featuring 32 entity filters and 2048 context filters of size 2, 3, and 4 gave us the worst result. This seems to indicate that putting a wrong focus on context wouldn't do us any good when there is not enough capacity to differentiate the surface names.

To better unveil the indication of the result, we've plotted Figure 5.3. The horizontal plane represents the number of filters in both entity (size 1) and context (size 2, 3, and 4) setting. The vertical axis represents the accuracy achieved. The scattered point are the exact result obtained from the experiment, and the 3D surface is interpolated from theses discrete points. The color on the surface is positively related to the accuracy. The higher the accuracy is, more redness the patch receives. As the accuracy decreases, darker shade of blue is drawn.

We've had three observations. First, despite the hollow in the middle of the surface and flatness over the rightmost strip, it seems that for any fixed number of context filters, increasing the number of entity filters generally improves the accuracy, and for any fixed number of entity filters, decreasing the number of entity filters boosts the accuracy. This is counter the intuition that the both filters would go neck-to-neck in disambiguating entities. As it turns out, the number of entity filters is way more important than the number of context filters. We attribute this to the greater power of recognizing different variance of surface names given by entity filters, which saves the trouble of relying on context to differentiate. There are only some limited variance of surface names, while there could be tons of context words surrounding a surface name in the wild world, many of which hardly relevant to the entity. Context window is only a physical way to fetch context, relying on

number of entity filter (size 1)	number of context filters (size 2, size 3, size 4)	Accuracy
32	32,32,32	0.835
32	64,64,64	0.821
32	128,128,128	0.819
32	256,256,256	0.749
32	512,512,512	0.683
32	1024,1024,1024	0.528
32	2048,2048,2048	0.235
64	32,32,32	0.850
64	64,64,64	0.816
64	128,128,128	0.786
64	256,256,256	0.795
64	512,512,512	0.765
64	1024,1024,1024	0.669
64	2048,2048,2048	0.331
128	32,32,32	0.920
128	64,64,64	0.890
128	128,128,128	0.886
128	256,256,256	0.846
128	512,512,512	0.789
128	1024,1024,1024	0.755
128	2048,2048,2048	0.488
256	32,32,32	0.920
256	64,64,64	0.922
256	128,128,128	0.912
256	256,256,256	0.877
256	512,512,512	0.872
256	1024,1024,1024	0.771
256	2048,2048,2048	0.683
512	32,32,32	0.910
512	64,64,64	0.922
512	128,128,128	0.909
512	256,256,256	0.910
512	512,512,512	0.883
512	1024,1024,1024	0.730
512	2048,2048,2048	0.682
1024	32,32,32	0.925
1024	64,64,64	0.931
1024	128,128,128	0.920
1024	256,256,256	0.917
1024	512,512,512	0.899
1024	1024,1024,1024	0.858
1024	2048,2048,2048	0.776
2048	32,32,32	0.933
2048	64,64,64	**0.934**
2048	128,128,128	0.925
2048	256,256,256	0.928
2048	512,512,512	0.907
2048	1024,1024,1024	0.878
2048	2048,2048,2048	0.798

Table 5.4: Number of Filters

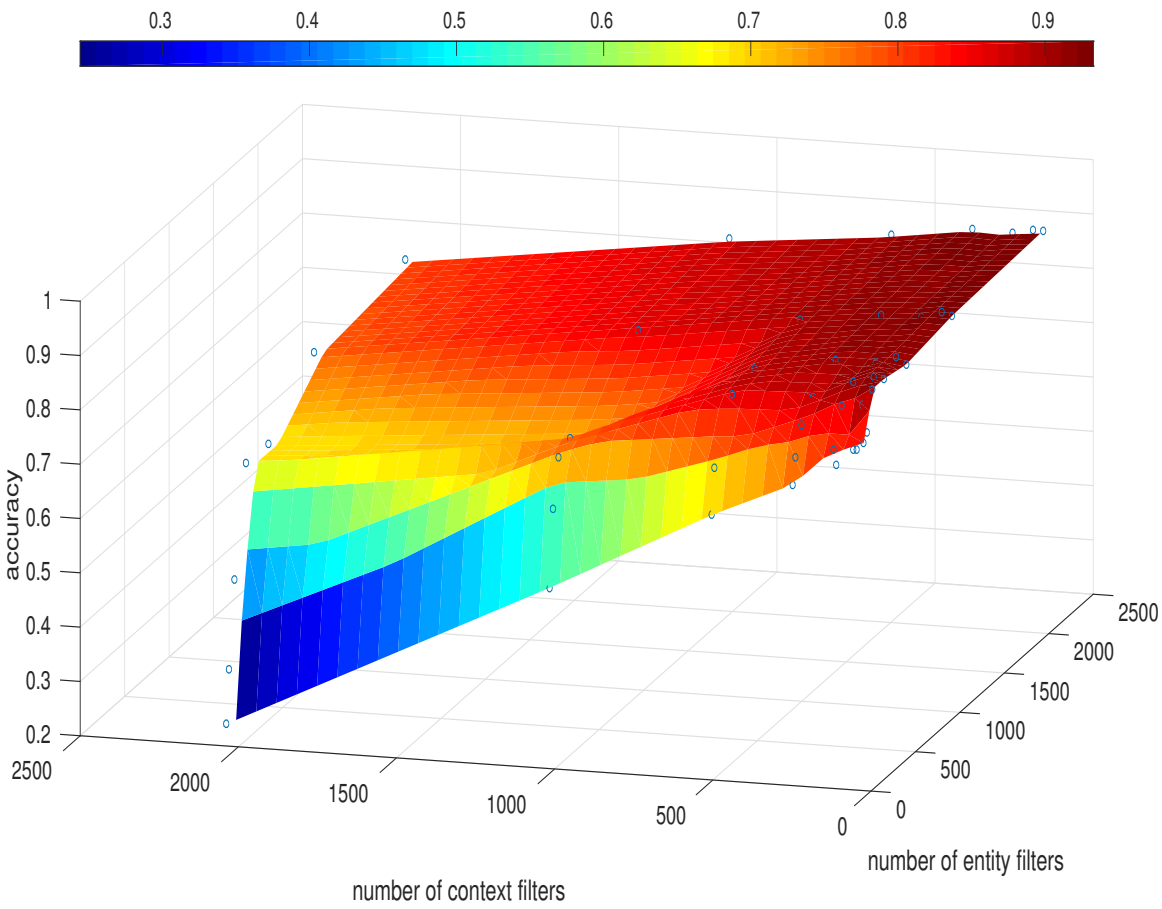


Figure 5.3: Impact of Number of Filters

an arbitrarily set distance away from the surface name. There's no guarantee that the real relevant context information is included in the context window. Hence focusing our limited computing power on the limited variance of surface names is more cost effective than focusing on the largely varying contexts, which easily gains us a competitive edge.

Secondly, we noticed that the lower left strip of the surface remains largely blue. This is the region where number of context filters overwhelm number of entity filters. It suggests that without a large number of entity filters, the context filters cannot alone shoulder the task of entity linking.

Lastly, the middle right region remains largely deep red though their configuration only spells for a reasonable number of entity filters and small number of context filters. Increasing the number of entity filters doesn't significantly improve the accuracy. This might imply the number of entity filters is not the sole ingredient in improving accuracy. One can achieve a near top result without wasting lots of computation power.

To further investigate this issue, we've plotted Figure 5.4 which illustrates how the ratio of entity and context filters influences the result. The horizontal axis represents the ratio between number of entity filters and context filters. There are seven groups of such ratio, according to our experiment. Each group includes a bunch of experiment configurations where the number of filters may vary, but their ratio is a constant value. The vertical axis represents the accuracy of these experiments.

The red line is the median value of a group, the top bar and bottom bar of the rectangular box represents the third and first quartile of a certain group of configurations. The short horizontal lines outside the box represents the maximum and minimum value of a group of experiments.

As revealed by the plot, the median accuracy seems to have an exponential relationship with respect to ratio. As the ratio slightly increase from 1/64, there is a drastic improvement in median accuracy. Yet when the ratio approaches 4 or 8, the increase becomes flat, and little improvement is observed beyond this point. This indicates that having entity filters four/eight times the number of context filters seems to be a good choice for a reasonably top performance. Increasing the ratio

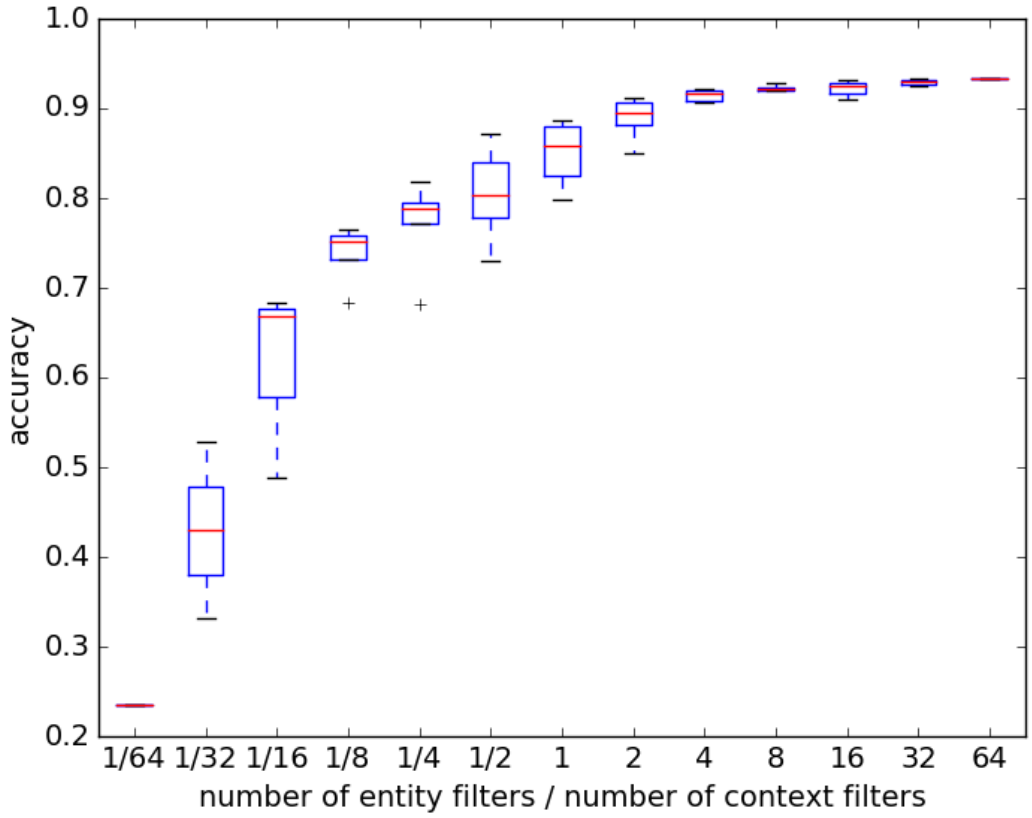


Figure 5.4: Accuracy Distribution with Respect to Entity and Context Filters Ratio

further on wouldn't drive the accuracy up by a significant margin.

Another thing we've noticed is that, as the ratio increases, the interquartile range (IQR), which is the distance between the third and first quartile, reduces. This implies, as we increase the ratio, the accuracy is more stable around median value, and has less variance. Therefore, having a reasonably high ratio is also recipe for more stable performance.

From the figures above, we conclude that having an entity/context filter ratio of being 4/8 is a recipe for reasonably high performance. We don't have to feed 2048 or more filters to the network, because more filters will require significantly more training time. From an application's perspective, having entity filters 4 or 8 times the number of context filters and choosing numbers such as between 256 and 1024 instead of 2048 may seem like an informed decision without degrading the performance too much.

5.2.4 Convolution Layers

We next analyse how much effect the number of convolution layers has on performance. We've picked two configurations from our previous observation. One has 2048 entity filters of size 1, and 64 context filters for each size of 2, 3, and 4. This configuration also has the best performance according to our previous experiment and shall speak for our top-notch configuration so far. The other has 256 entity filters of size 1, and 64 context filters for each size of 2, 3, and 4. This configuration has a reasonably good performance in our previous observation, and shall serve as a representative for medium-high configurations. For each of these two configurations, we vary the number of layers in entity and context subnet from 1, 2, and 3. In each of these layer, the input is convolved and down-sampled before feeding the output to the next layer, which repeats the convolution and max pooling process again.

The result is shown in Figure 5.5 and Figure 5.6. The horizontal plane shows the number of layers for entity and context subnet, and the vertical axis shows the accuracy. The color of the mesh grid is plotted according the vertical axis, with higher accuracy being colored more red and

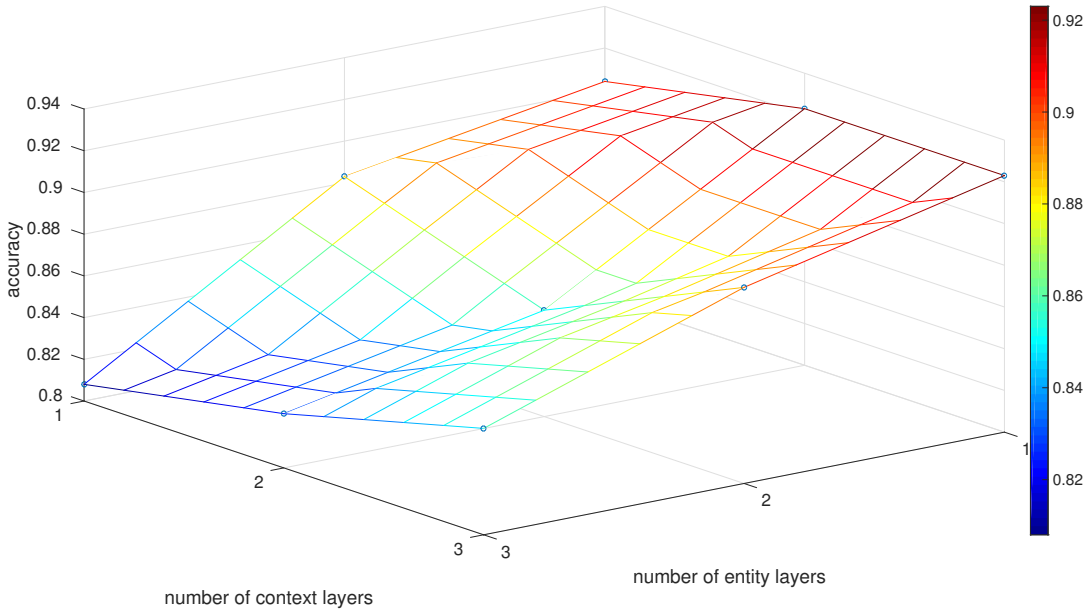


Figure 5.5: 256 entity filters of size 1 and 64 context filters of each size 2, 3, and 4

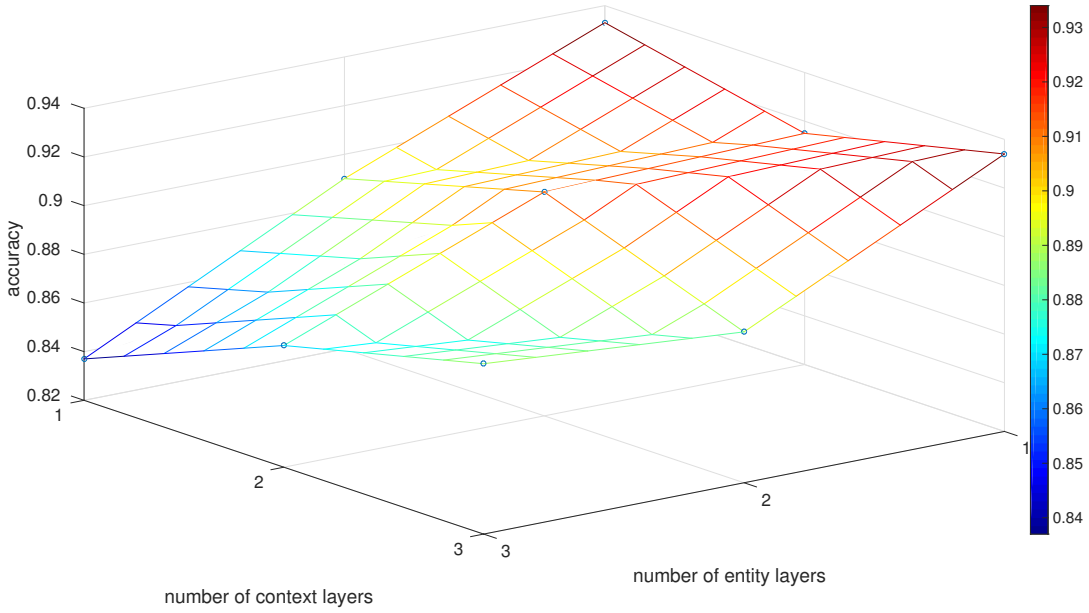


Figure 5.6: 2048 entity filters of size 1 and 64 context filters of each size 2, 3, and 4

lower accuracy colored more blue.

As shown in the picture, the highest accuracy is achieved when there is only one convolution layer in the entity subnet. This is because we only need the most important key word in the surface names. Having one layer, while using a filter size of 1, will allow us to find the most important unigram in the surface names. It is bewildering to increase the layers in the entity subnet, because more layers will allow us to find more sophisticated higher structures at the expense of overfitting. At a time when most key disambiguating clue hidden in the surface names is one word long, constructing more layers distracts the neural network from focusing on the real clue, and as a result, the performance wanes.

Convolution layers in context subnet is, on the other hand, more sophisticated. This is inherent in the varying nature of disambiguating context key words. Some disambiguating context key words might be far apart from the entity position, and fall outside of the filter window size. Hence it requires deeper convolution layers to construct high dimension patterns based on preliminary structures captured directly by filter window. Some disambiguating context key words are close to the entity, and hence can be directly processed by filter window. Therefore they don't need deeper neural network, and constructing more layers would backfire by increasing the risk of overfitting. This explains the irregular and seemingly patternless surface presented by the figures. An increase in the number of context layer leads to an increase in accuracy such as the middle point in Figure 5.6, while an increase in the number of context layer results in a drop of accuracy in others (such as the middle points in Figure 5.5). The figures show us how intricate it is to capture context disambiguating key words, and one should be careful in choosing his layers in context subnet.

It's also worth noting that the worst performance in both experiments occur with 3 entity layers and 1 context layers. This is reasonable because we give too much computation power where it needs the least (entity) and too little where it is most sophisticated (context). This indicates that the design of a neural network must agree with the complexity of test dataset. We don't want to misplace the computation power to places where its complexity doesn't match.

5.2.5 Fully Connected Layer

We next investigate the effect of fully connected layers on the overall performance of the network. We've used three different configurations to run our experiments, as summarized in Table 5.5. *config1* is the best model observed so far given our previous experiments, and hence serves as a representative for our high-end configurations. *config2* is almost the complete opposite of *config1*. With very few entity filters and many context filters, it's supposed to represent the low-end configuration. *config3* is a more mild setup. Its hyperparameters lie between *config1* and *config3*, and it is meant as a representative of average configurations.

Configuration Number	entity filter size	number of entity filters	context filter size	number of context filters	ConvPool layers of entity subnet	ConvPool layers of context subnet
config1	1	2048	2,3,4	64,64,64	1	1
config2	1	32	2,3,4	2048,2048,2048	3	1
config3	1,2	512,512	2,3,4	512,512,512	2	2

Table 5.5: Different Configurations for Fully Connected Layer Neurons Experiment

For each configuration, we train our neural network using the parameters stated in the table, and vary the number of neurons in the fully connected layer from 2^6, 2^7, 2^8 all the way to 2^{12}. We are interested to see if the number of neurons in the fully connected layer has a universal impact pattern on the performance across low-end, average and high-end configurations.

Figure 5.7 illustrates the result we've obtained. First of all, we noticed that, the curves all present an exponential-like relation with regards to the number of neurons in the fully connected layer. The accuracy increases drastically in the beginning but grows flat at the end. Most of the increase happens before 256. Recall that we have 271 different entities, it indicates that as long as the number of neurons in the fully connected layer remains smaller than that of the number of classes, the full potential of the network is not tapped. This is intuitive in the sense that we can see each neuron in the last layer as an information aggregation for each class, and the number of neurons needs to no fewer than the number of classes to clearly differentiate the surface names to

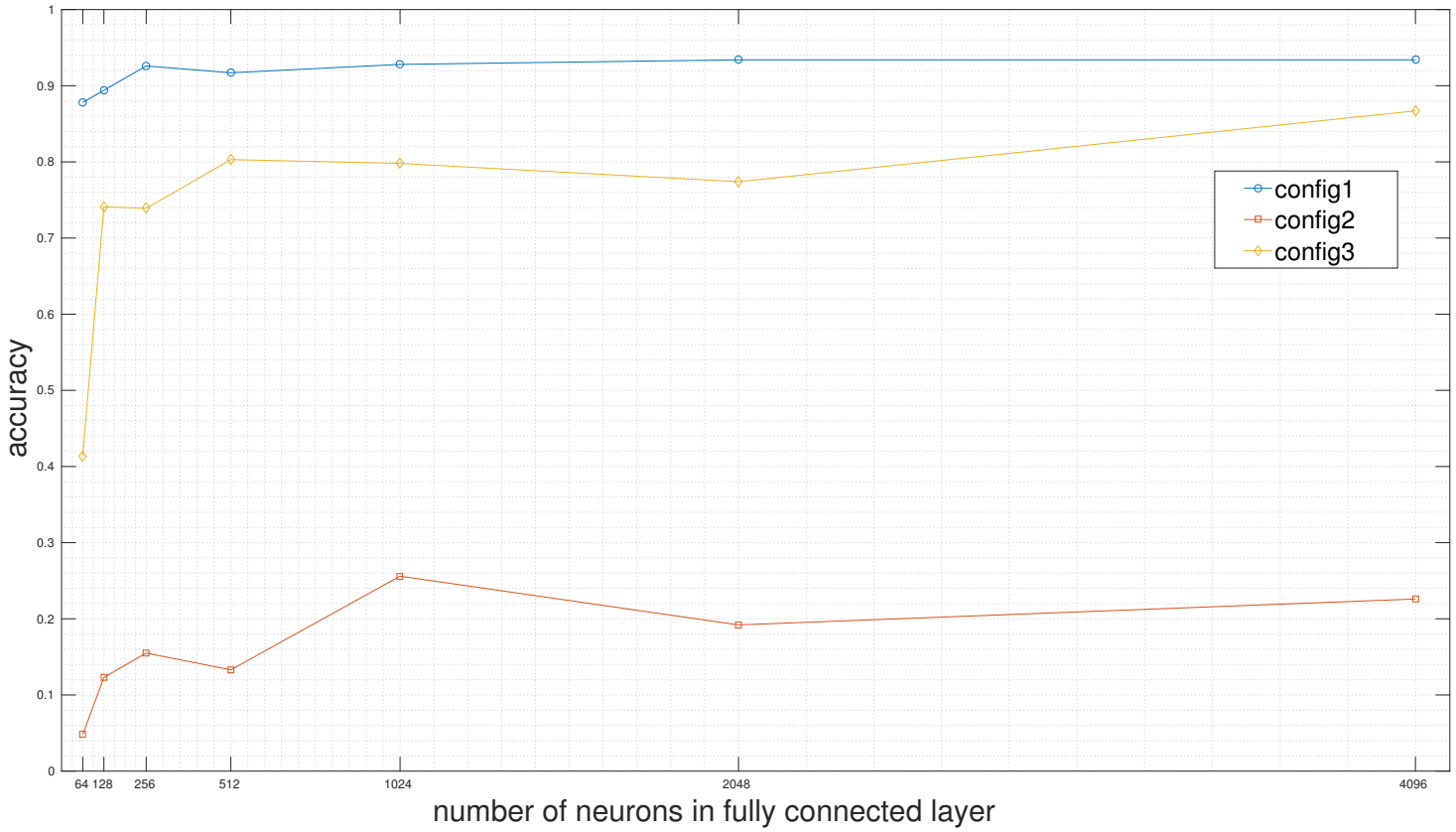


Figure 5.7: Performance with Respects to Neurons in Fully Connected Layer

different entity.

Another interesting thing to notice is that, even when the number of neurons in the fully connected layer is low in config1, the accuracy in config1 still outperforms the best ones in config2 and config3. This suggests how important it is to process patterns early on in the ConvPool layer instead of later at the fully connected layer. Fully connected layer is, after all, an aggregation for patterns found in previous layer. If patterns are not even discovered early on, there is little chance of finding anything substantial at later stage. This nicely demonstrates how each layer is geared into each other, and early loss of information cannot be restored by later layers.

5.2.6 Dropout

Large and deep neural network contains tremendous parameters, which often causes overfitting problems. Dropout is a simple technique to address this issue. In this subsection, we investigate how well dropout is able to manage overfitting by running two sets of configurations. The configurations are summarized in Table 5.6. For each configuration, we trained neural network with dropout keep rate of 0.1, 0.2, 0.3, all the way up to 1. In the last case, there is no dropout because every neuron in the final fully connected layer is kept.

Configuration Number	entity filter size	number of entity filters	context filter size	number of context filters	convolution layers of entity subnet	convolution layers of context subnet
config1	1	2048	2,3,4	64,64,64	1	1
config2	1,2	512,512	2,3,4	512,512,512	2	2

Table 5.6: Different Configurations for Dropout Experiment

The result is illustrated in Figure 5.8. First of all, we noticed that for all sets of configurations, using dropout improves accuracy by varying margins. This verifies the effect of dropout as a simple regularizing technique.

In addition, a small dropout keep probability means a smaller subset of the network is used in training. This might has the risk of underfitting depending on the initial number of parameters

inherent in the network. In our experiment, setting the probability to 0.1 or 0.2 results in significant lower accuracy than using higher probability. It indicates that the hidden combination of neural network is not fully exploited, and there's significant underfitting in the network. However as we tune the probability to 0.3 and above, the performance gradually improves. There are different places where you can achieve top performance. For configuration 1, it's 0.5, and for configuration 2, it's 0.6. In general, on a larger scale, we find the curve becoming flat in the range of 0.3 to 0.8. This might imply that the best dropout keep probability lies somewhere in that range. Continuing to increase the probability beyond that range results in a drop of performance, and using no dropout (probability 1.0) at all is also inferior to its peak values as well.

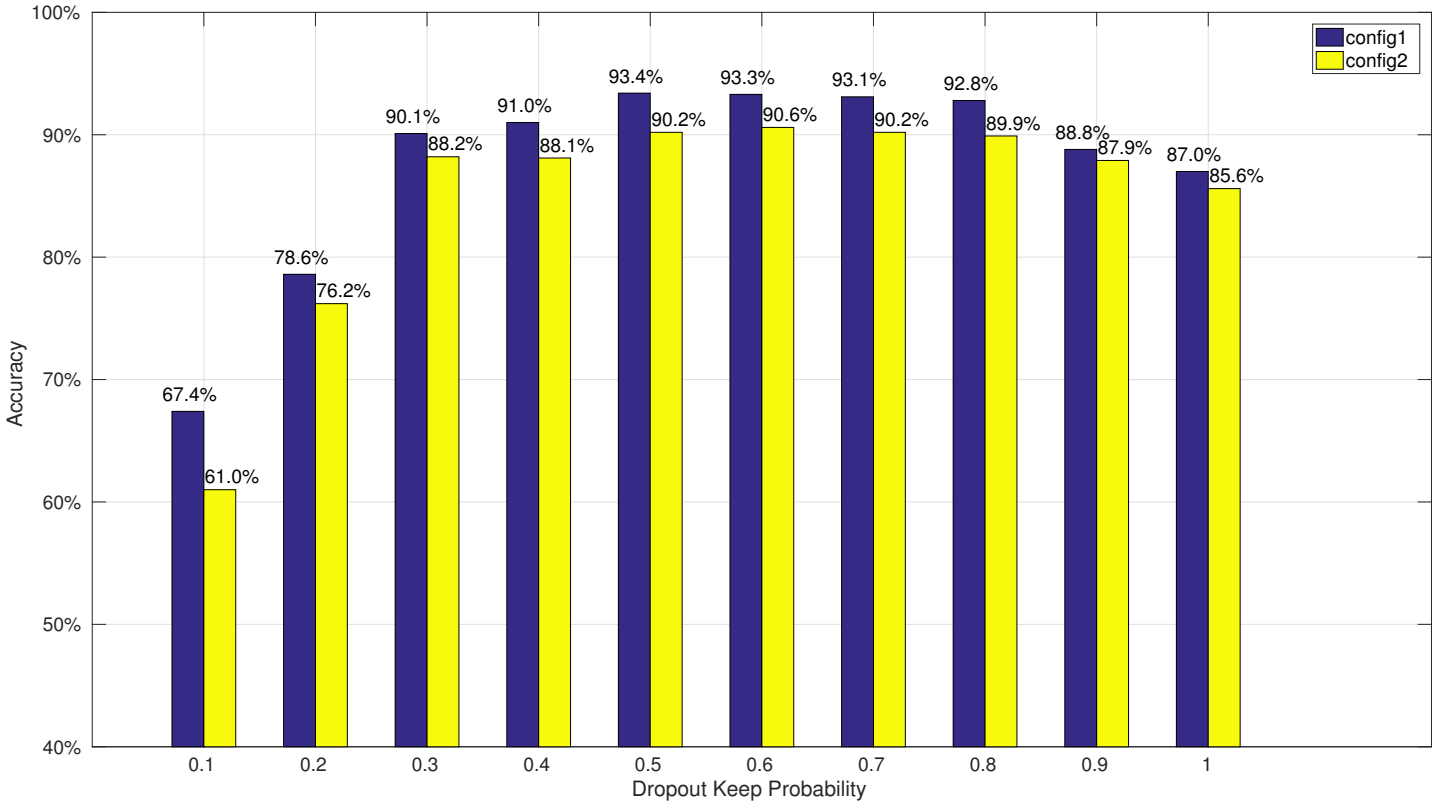


Figure 5.8: Performance Comparison with Different Dropout Probability

5.2.7 Comparison with Previous Approaches

Given our previous experiments, we find the best model to be the one list in Table 5.7. We call it CNN-link (Convolution Neural Network for entity link)

entity filter size	number of entity filters	context filter size	number of context filters	ConvPool layers of entity subnet	ConvPool layers of context subnet	number of neurons in fully connected layer	dropout keep probability
1	2048	2,3,4	64,64,64	1	1	2048	0.5

Table 5.7: Best Model Configuration

We compare accuracy of our model with that obtained by *GLOW* [39] and *NEREL* [44], which are the previous state-of-art approaches on the MSNBC dataset respectively in 2011 and 2013. There are three variants of the *GLOW* algorithm. The *GLOW-local* makes use of local features such as the text similarity between mention and Wikipedia title. The *GLOW-global* makes use of global features such as some sophisticated relatedness measure making use of in/out links of the Wikipedia page. *GLOW-local+global* is a combination of these two. Either way, the GLOW approach makes heavy use of man-defined features, and leverages heuristic and empirical design to link entity mentions to their knowledge base equivalent. Similarly, *NEREL* also makes use of numerous features such as *capitalization*, *number of tokens inside mention*, and etc.

Figure 5.9 shows the accuracy achieved in four approaches. CNN-link outperforms all variants of *GLOW* as well as *NEREL*. We attribute this to the power of CNN to be able to identify intricate patterns that are hard to engineer by bare hands. As we have found in earlier subsections, CNN can automatically single out key words in the text by making use of a wide array of filters (especially entity filters). Context filters sometimes help, as we will see in later visualization sections, though their patterns are more intricate to generalize. As a result, words that are more related will receive higher focus/weights by the neurons, and push the network training to proceed in the direction that minimize loss (i.e., cost paid for inaccurate prediction). On the other side, handcrafted features as those perceived in *GLOW* and *NEREL* may not capture the key information or could lose the

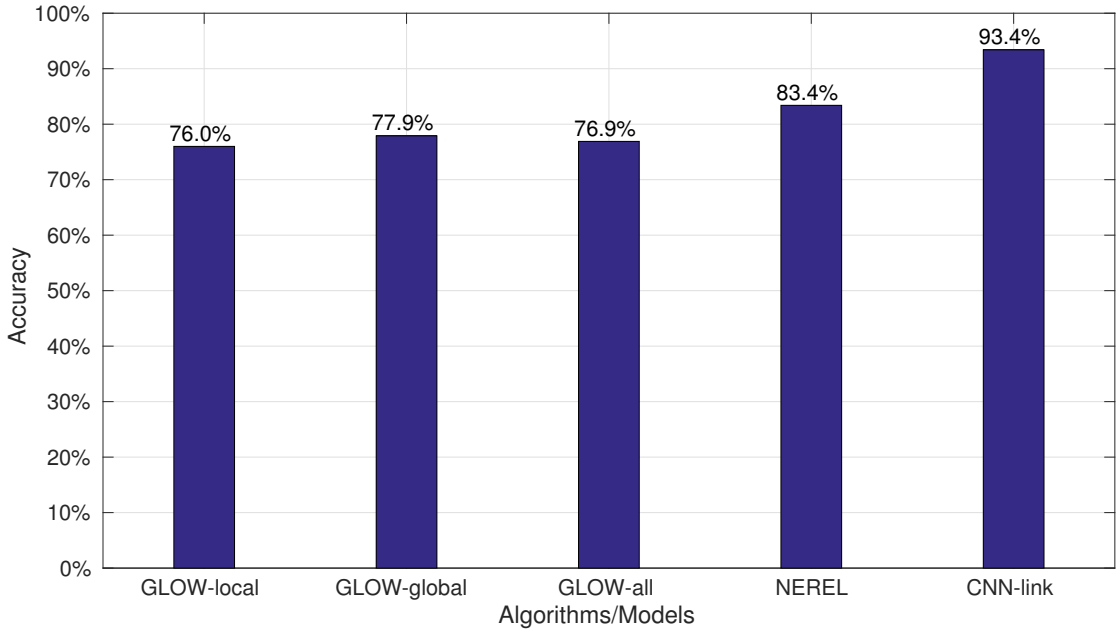


Figure 5.9: Accuracy Comparison with GLOW and NEREL

information because some lesser features override the key ones.

5.2.8 A Note on Comparison with Previous Approaches

We attempted to compare with [39] and [44] mainly because they are previous state-of-art approaches in feature based designs on the MSNBC dataset, which is accessible to public. There are several difficulties in comparing with other approaches mentioned in chapter 2.

The first and foremost reason is that many approaches (such as the more recent neural network based approaches as in [50]) either didn't publicize their dataset or used the dataset provided by LDC *(Linguistic Data Consortium)*, which is an organization that creates and distributes a wide array of language resources. Unfortunately, University of Calgary is not their member, and we don't have access to their dataset (in the entity linking case, it's the *TAC KBP* dataset).

Secondly, some of the unified approaches that complete entity linking with other tasks such as entity recognition and relation extraction [7] used joint measures such as F1 (which is a combination of precision and recall rate), because their problem involve false positive and false negative.

They didn't mention their accuracy/precision result alone.

Other than that, some approaches used dataset that's closely bound with other KBs such as *CoNLL-YAGO* used in [19], and it's unfair to compare because their dataset comes with KB specific attributes.

5.3 Visualization

Neural network is known for its inherent complexity, which involves tremendous parameters that are hard to interpret. In this section, we provide some visualization primitives to understand our neural network, and try to unveil what happens under the hood.

5.3.1 Dimension Reduction and 2D Plane Embedding

Our Convolution Neural Network can be seen as a black box function to encode the original input with some high dimensional representation, which is further used by a linear classifier to link to different entities. Therefore, we can try to understand its high dimension space topology by reducing it to two dimensions, which can be then easily plotted in 2D space. The dimension reduction is such that pairwise distance of points in high dimension space is preserved in the 2D space, so that the farther apart two points are in the original space, the more distant they will be in 2D space. There are many existing techniques and tools to help streamline the process of embedding high dimension space into low dimension space. Among these, we used *t-SNE*. It is one of the highly popular tools to produce solid results on 2D plane.

In this subsection, we use the output of the last fully connected layer (without dropout) as the encoded code of a text containing entity. We call this the *CNN code* of the text. We used 1024 neurons in the last fully connected layer, therefore each text is represented by a 1024-dimension vector. We chose five entities to visualize and some examples of the visualized sentence is shown in Table 5.8. The bold texts are surface names identified in the dataset, and their canonical names are listed in the rightmost columns.

Text Examples	Canonical Entity Name
Bayh decided not to run. "Washington does not make the case", said **dan pfeiffer** who worked for Bayh	Dan Pfeiffer
one thing for sure, says **pfeiffer**, voters are tired of arguing about the culture of the 1960s and other Boomer issues	Dan Pfeiffer
she traveled all night from the central Indian state of Madhya Pradesh to bathe in the Ganges, as she has done at every **kumbh mela** over the last 25 years	Kumbh Mela
Garuda's flight lasted 12 divine days, or 12 years of mortal time, hence the celebration of "**Maha Kumbh Mela**" every 12 years.	Kumbh Mela
A draft portion of the report obtained Tuesday by The Associated Press gives the best ratings to the Washington, D.C., area; San Diego; **Minneapolis-St. Paul**; Columbus, Ohio; Sioux Falls, S.D.; and Laramie County, Wyo.	Minneapolis Saint Paul
Light, sweet crude settled 2.73 to 58.32 per barrel on the New York Mercantile Exchange as mild weather continued its hold over much of the **United States**, cutting demand for heating oil and natural gas.	United States
The battle began on Wednesday's show, when a peeved O'Donnell said: "(He) left the first wife had an affair. (He) had kids both times, but he's the moral compass for 20-year-olds in **America**. Donald, sit and spin, my friend."	United States
The gap between the Medicaid and private health insurance leaves up to 9 million **U S** children uninsured	United States
Instead, Americans chose Jimmy Carter, a peanut farmer who had never worked in **Washington**, and who promised never to lie to the American people	Washington

Table 5.8: Examples of Visualized Sentences

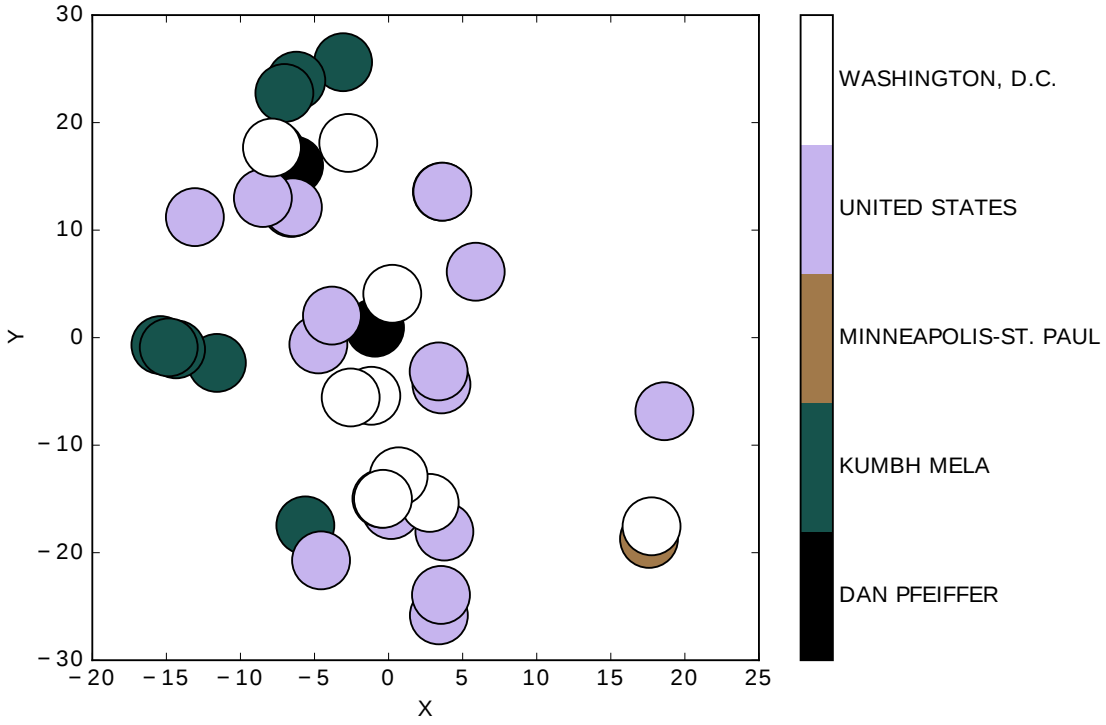
Table 5.8 well illustrates a subset of problems we need to resolve in entity linking, which are also common intricacy in surface names. This includes, but is not limited to, partial name vs full name (Dan Pfeiffer vs Pfeiffer), irregularities(Minneapolis-St. vs Minneapolis Saint), synonym (America vs US) and abbreviation (US vs United States).

Ideally, a good neural network should produce *CNN code* such that all the surface names of the same entity would end up being closer to each other in the high dimension space, forming a group of itself, and being well separated from other groups. In that sense, we borrow the concept of *cluster* from data mining, and denote each entity as one cluster. Every test instance is assigned a cluster membership ID based on what its containing surface names refer to, as per the ground truth. Since the membership is pre-assigned, our convolution neural network is more or less performing a supervised clustering process. Though we are not changing the cluster membership id throughout the training process, we are iteratively changing the position, in the 1024 dimension space, of each minibatch example in the training phase, so that the network can develop some idea of high dimensional regions belonging to each different entity. Our test cases are simply mapped to positions in the 1024 dimension space to see how good our network develops the idea of regions, or clusters.

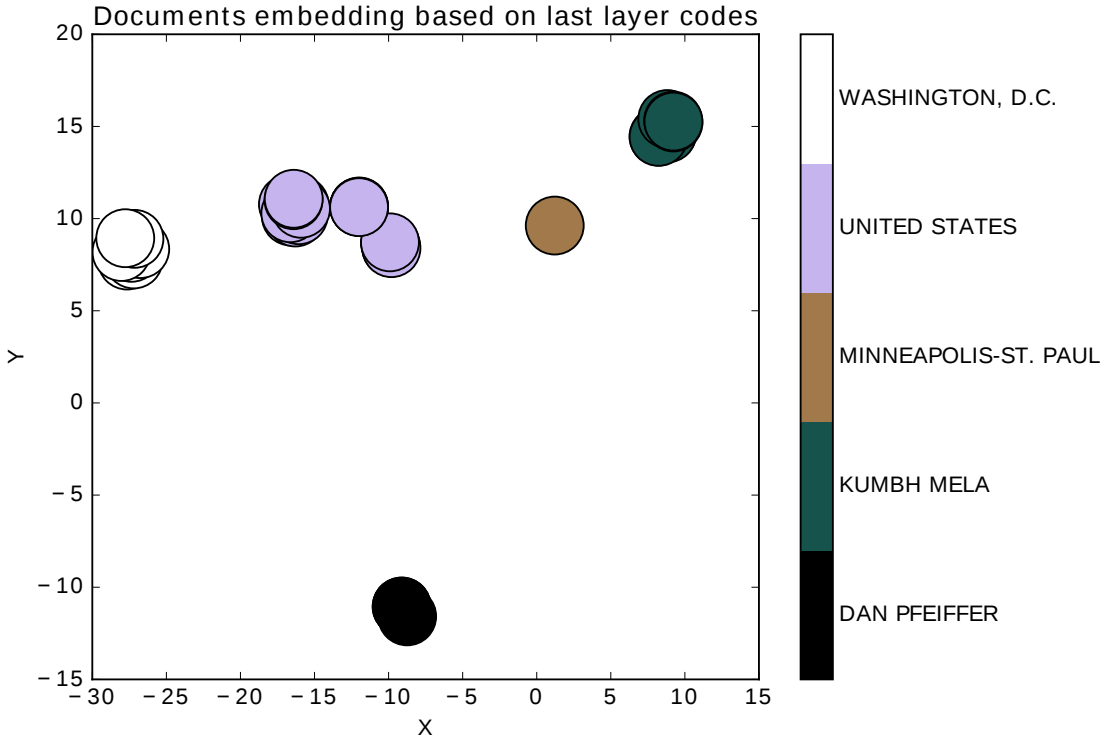
Figure 5.10 shows the result we've obtained after embedding high dimensional positions expressed as *CNN code* into 2D space. Each bubble represents a text that contain certain entity, and the same entities are colored with the same color. The colorbar on the side shows which color is used for entities.

Figure 5.10a is the baseline approach that we compare with. It simply does a word embedding for all words in the text and then averages them to get the vector which represents the position of the sentence, in order to be reduced in dimension by *t-SNE*. Figure 5.10b is produced by treating the *CNN code* as the position of text, followed by the same dimension reduction carried out by *t-SNE*.

As seen clearly from the figures, Figure 5.10a demonstrates a messy, seemingly random clus-



(a) Visualization of text based on averaging word embedding



(b) Visualization of text based on last layer codes

Figure 5.10: Visualization of text on 2D Space with Dimension Reduction

tering. Texts which contain the same entities are scattered across the space, with no clear convex hull formed that is well separated from the others. It shows that the baseline has a very bad idea about entity and its topological shape/closure.

On the other side, Figure 5.10b clearly separate different entities into further apart regions, forming a convex hull that fully closes the shape. There is no intersection between regions, and there's no bubble that is far away from its owning cluster. Figure 5.10b well demonstrates the capacity of convolutional neural network in recognizing entities, forming high dimensional topological closures. The *CNN code* hence looks robust.

Figure 5.10 gives us a qualitative appreciation of neural network's inherent ability to learn topological shapes. In order to gain a quantitative measure of how well the clustering is, we resort to some common measures popularly used in the field of data mining. One of these measures is *intra-cluster distance*. There are many different definitions about *intra-cluster distance*. We used the definition as in 5.2.

$$\text{intra-cluster distance}_{\mathscr{A}} = \max d(x, u_{\mathscr{A}}), \forall x \in \mathscr{A}, \text{ where } u_{\mathscr{A}} \text{ is the arithmetic mean of cluster } \mathscr{A} \tag{5.2}$$

It's also known as the *radius* of a cluster, because it measures the distance between a cluster's center to its most off point. The smaller the *radius*, all points are more compactly squashed together, with no outliers.

Figure 5.11 plots the intra-cluster distance of every cluster/entity. There are some clusters that have 0 intra-cluster distance. It's either because all their test instances are mapped to indifferentiable 2D positions or there's only one test instance in that entity so that the center and far-off point of the cluster is the same.

Most of the clusters have very low intra cluster distances, with their values below 0.5. There are, though, a fair number of clusters having between 1 to 2 units of intra-cluster distance. The highest number is 2.5, achieved at cluster ID 59, which is entity 'Donald Trump'.

To get a better sense of how good/bad those values are, we've taken the clusters with top 3

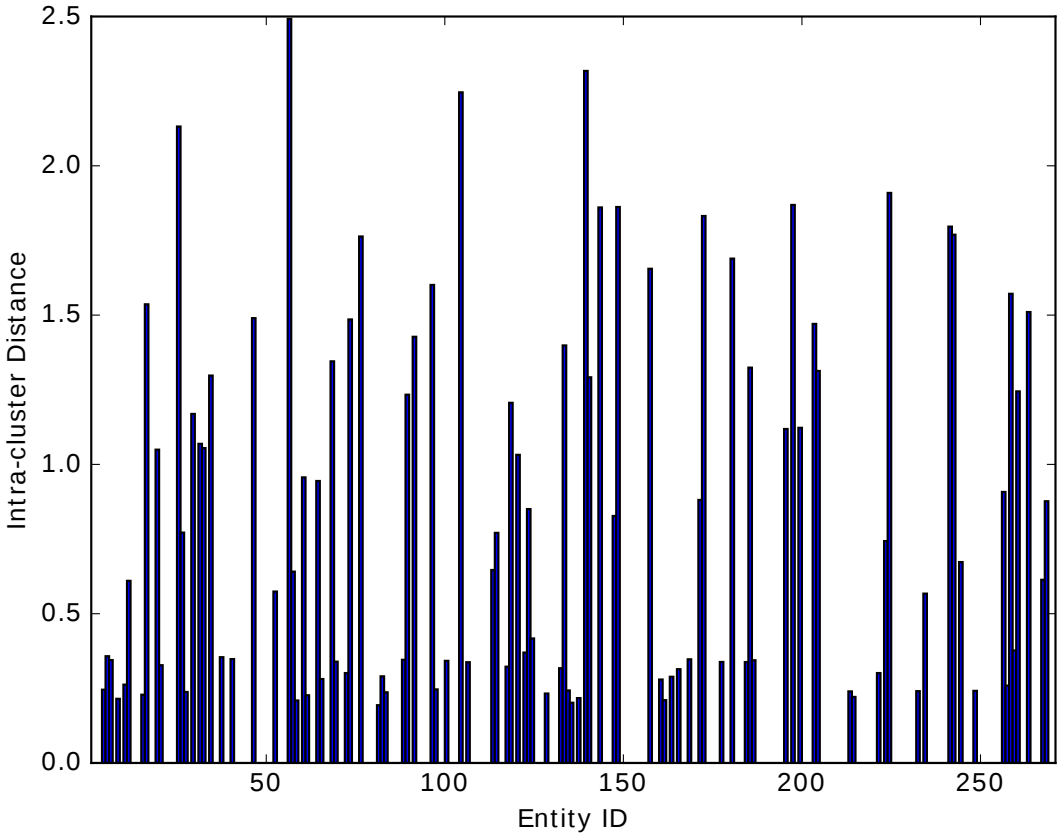


Figure 5.11: Intra-cluster Distance

intra-cluster distances and plot them in to a 2D plane as we've done before.

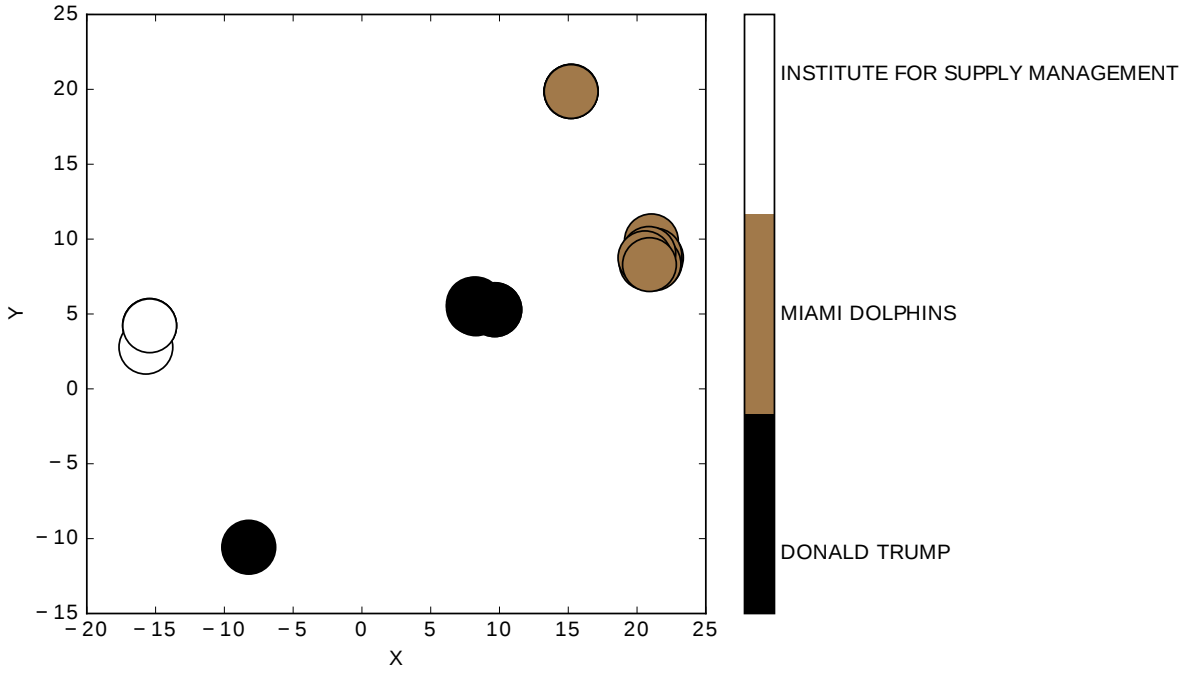


Figure 5.12: 2D Embedding of Entities with Large Intra-cluster Distance

Figure 5.12 shows the embedded plot. Entity 'Donald Trump' and entity 'MIAMI DOLPHINS' seems to be nicely grouped together except for one instance each. Those two outliers are far apart from their counterparts, indicating an error in linking the entity.

Text	Ground Truth Entity	Classified Entity
Alabama Sweetened an offer that will make him the highest-paid coach in college football. He has three years remaining on his **Miami** contract at 4.5 million a year	Miami Dolphin	Miami, Florida
Walters also took a moment to try to smooth things over with **The Donald**, who got all riled up when O'Donnell said on "The View" that he had been "bankrupt so many times."	Donald Trump	The Associated Press

Table 5.9: The Sentences That Are Outlier

We put the two outliers in Table 5.9. It shows that *Miami* is mis-linked to *Miami, Florida* when it should be linked to *Miami Dolphin*, the football team in Miami. This is more of an intricate question, because interpreting *Miami* as *Miami, the city* makes sense as well, though it may not

be as informed as *Miami, the football team* in this context. However, to sort out this delicacy involves some deeper level reasoning, and we are afraid that it goes beyond the capacity of our convolutional neural network.

The Donald is mis-linked to *The Associated Press* when it should be linked to *Donald Trump*. This looks more like a mistake of the neural network. The annotator of the dataset includes one more word *'The'* into the more common surface name *'Donald'*, and we suspect it causes the neural network to shift away from the promising course, and end up with *The Associated Press*. We believe this error could be more or less corrected by involving training dataset that goes by the surface name of 'The Donald'. The neural network can learn that the differentiating term *The* not only appears in *The Associated Press* but can also appear in *Donald Trump*. It will learn to be more cautious in choosing which one is the right entity to link.

In sharp contrast to the two wild off entities, the entity of *Institute for supply management* seems to have an acceptable clustering result. The two test instances are relatively close to each other. The reason that they still contribute to a third largest intra-cluster distance might be that, in one test case, the surface name is the abbreviation *ISM* while in the other, the surface name is the full name. Different names, especially those involving shorthands and looking like very irrelevant, will have far initial distance after word embedding. Neural network learns to pull them closer, but still it is a complicate function of initial word embedding and the neural network that decides their final position in the 1024 dimension vector space.

Since we know that *"Institute for supply management"* already has the third largest intra-cluster distance, and it doesn't look too bad on the embedded 2D space. We are assured that other clusters shouldn't have too loose closures, and the topological clustered result is good.

5.3.2 Occlusion Test

Occlusion test is a technique proposed in computer vision [56]. It seeks to know if the model is able to locate the object in the images that provide key identification clues. It can also show

whether the surrounding context has a role to play in image classification. In *occlusion test*, a grey patch is gradually slided across the image, each time occluding part of the image, and one can observe how the probability of the class of interest, such as its correct class, changes as a function of the occluded portion.

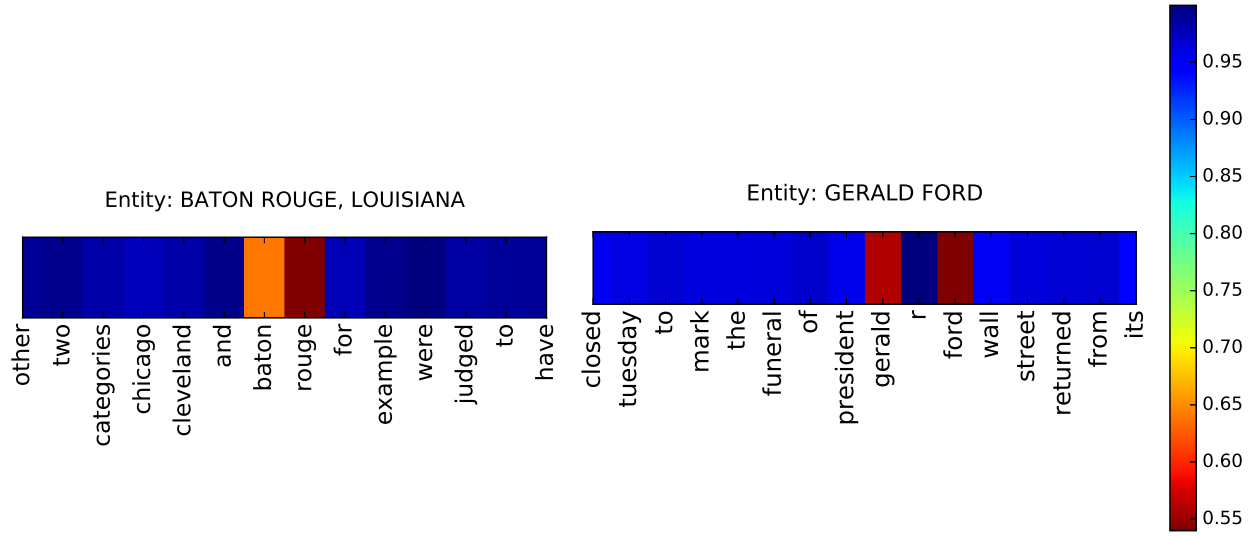
Ideally, if the key part of the object is occluded, say the head of a dog image, the probability of the object belonging to the class of interest, say the class dog, will plummet. Hence, one can plot a heatmap, showing the probability of class of interest as a function of different occluded image patches.

In the context of entity linking, we can slide the occluder across the sentence, hiding one word each time, and observe how the probability of the correct entity changes as different words become invisible.

In our occlusion experiment, for each test sentence of n words, we generate n occlusion test instances, each replacing one different word with our predefined padding symbol word. We used padding word because padding is normally done at the end of each sentence to keep all sentences of equal length, and our neural network is supposed to learn that the padding word means nothing. This is analogous to the grey patch used in image classification, which simply means nothing of particular interest is here, and the neural network should be intelligent enough to move on.

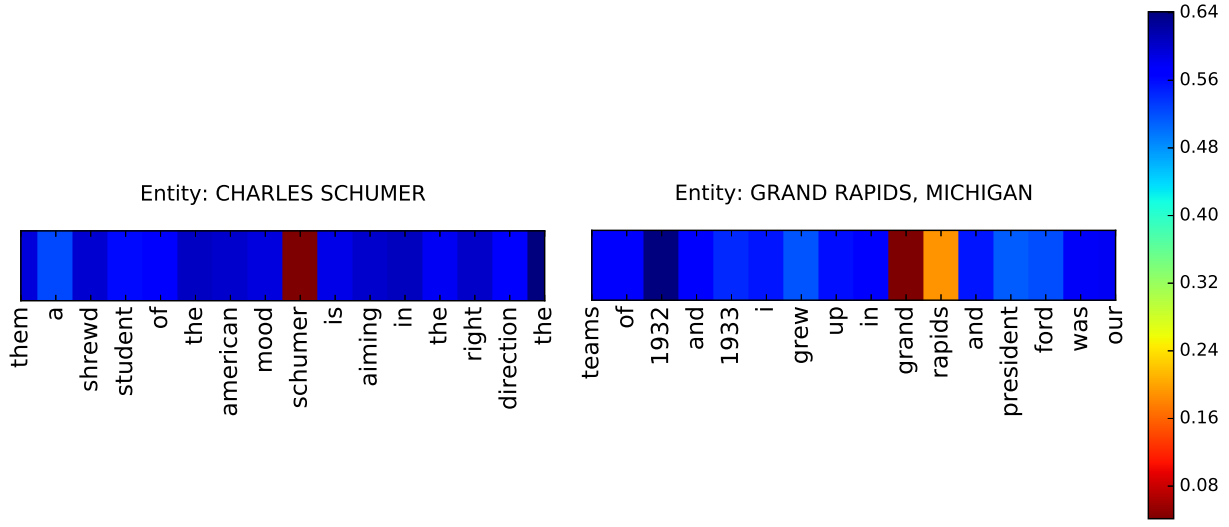
To get the probabilities of class of interest, we took the final output of our neural network, note that this is the *logit* applied to a softmax function. It is a huge matrix that contains probability for all test instances and all entity classes. We then took the slice of probability that belongs only to the correct entity (class) of each text. So each test instance will have a different slice to take depending on their ground truth entity.

We took four examples and show their heatmap in Figure 5.13. Note the difference between our heatmap and that of the computer vision is that since the sentence is one dimension, i.e., you can only move the occluder across the words, the resulting heatmap is one dimension, instead of two dimensions in computer vision because you can move occluder horizontally and vertically across



(a) Heatmap for Entity 'Baton Rouge, Louisiana'

(b) Heatmap for Entity 'Gerald Ford'



(c) Heatmap for Entity 'Charles Schumer'

(d) Heatmap for Entity 'Grand Rapids, Michigan'

Figure 5.13: Heatmap of Occlusion Test

the image.

The colorbar at the side shows the color associated with the probability. The lower the probability, more red is shown, and the higher the probability, the more blue is drawn.

For Figure 5.13a, we noticed that there is a drastic decrease in probability as the word *baton* or *rouge* is occluded. It drops to as low as somewhere between 0.55 and 0.7, which is in clear contrast to when the other words of this sentence are occluded. It shows that our model is able to identify the key words in the sentence that differentiate themselves from others. There is hardly any drop in probability as we occlude words such as *other*, *two* and *example*. It indicates that they probably don't have any context clue to offer in disambiguating the surface name *Baton, Rouge*.

For Figure 5.13b, again, a sudden drop of probability is captured as we occlude *Gerald* or *Ford*. It makes sense as they are a common name the entity goes by. What is interesting though, is that although the word *r* stands for the middle name and it's part of in the surface name, occluding it doesn't lead to a significant decrease in probability. This well demonstrates neural network's ability to match men's reasoning behaviour as we can still recognize a person even without his/her middle name.

In addition, we find that occluding *president* leads to a lower probability than occluding other context words. This shows that *president* is an important attribute of the entity *Gerald Ford*. As we know, *Gerald Ford* is the 38th president of the United States, and our neural network, in this case, is able to find key context information about the entity.

Figure 5.13c shows the heatmap for entity *Charles Schumer*, and as with previous examples, occluding the keyword *Schumer* decreases the probability to a great extent. We are not sure why occluding the word *a* results in a larger drop of probability than occluding other context words. This shows that the neural network is not quite mature and stable in picking the context word, and it needs to be improved, either with the model itself or with the dataset used in training.

Figure 5.13d shows the heatmap for entity *Grand Rapids, Michigan*, which is the second largest city in Michigan, and it has a library named after president *Gerald Ford* within the city. As before,

there's a significant drop of probability when *Grand Rapid* is occluded, which is reasonable. We are, however, delighted to find that the probability drops by a larger margin when the word *president* is gone than when the other context words are hidden. This shows that the our neural network might have some intuition of the *Gerald Ford Library* within its city, and it learns from the training corpus that the library is named after one of the presidents of United States.

Entities	Keyword
Automatic Data Processing	automatic, data, processing, report, payroll
British Airways	airways, airline, planes, freight, condition
Cameron Diaz	Timberlake, breakup, magazine, star
Donal Trump	bankruptcy, underage, moral
Frank Blake	experts, home, depot, Nardelli, company
FBI	officials, interrogate, border
George W. Bush	idea, Iraq, troops
Gerald Ford	form, president, waited, Reagan, Nixon
Heathrow Airport	airline, luggage, terminal, fog
Saddam Hussein	execution, arrested, leaked, cell, Iraq

Table 5.10: Words Resulting in Top Probability Decrease in Entities

We summarize the context words in test corpus that lead to significant probability decrease for some entities as above in Table 5.10. For each entity, the keywords are compiled across all test instances of the same entity. It illustrates an interesting concept/impression of the following entities developed by our neural network during training. Here's what we've found:

- *Automatic Data Processing*, an American company specialized in providing human resources management software and services, leaves an impression on our convolutional neural network that embodies automatic, data, processing, report and payroll.
- *British Airways*, an airline company, pivots around airlines, planes, etc.
- *Cameron Diaz*, who is an American actress, model, and known for her relationship with Justin Timberlake, would find her romantic story well picked up by our neural

network, with Timberlake shown up in one of keywords. Our neural network also learns that she's a star.

- *Donald Trump* is besieged with negative words such as bankruptcy and underage.
- *Frank Blake*, a former CEO of Home Depot Company, is associated with this company and his predecessor Robert Nardelli.
- *FBI*, features on officials and interrogate
- *Bush*, features on his idea of sending troops into Iraq.
- *Gerald Ford*, features on being a former president and being well associated with his predecessor Richard Nixon and his opponent in the 1976 GOP presidential primary campaign, whom he defeated.
- *Heathrow Airport*, a major international airport in London, pivots around airlines and luggage, etc.
- *Saddam Hussein*, the infamous president of Iraq, is known by our neural network to have execution, arrested, leaked on his keyword list.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis discusses the entity linking problem, which sees huge applications in areas such as automatic knowledge base populating, healthcare, and identity crime prevention. It can also improve people's experience of reading documents by enabling concept lookup in place, saving the trouble of unnecessary context-switching during reading.

Different from previous approaches which heavily relied on handcrafted features, including textual similarity and semantic closeness, our model used *convolutional neural network* to address the entity linking problem without the need to manually select the features to incorporate. In contrast, our model can implicitly figure out the useful features such as semantic similarity of words, topic key words, etc.

Our model makes use of word embedding to turn semantically similar words to closer continuous high dimensional space. We feed entity mention and context words via word embedding independently to the convolution neural network, which consists of two subnets of convolution and max pooling layers. Convolution uses a lot of neurons filters that scan through the spatial plane of the input, and collect preliminary feature information. Max Pooling filters out the most important feature in a given window and feed the output to the next interconnecting layers. In general, this architecture of convolution and max pooling can be repeated several times, independent in each subnet, to build more sophisticated patterns in deeper layers. The max pooling in each subnet's last layer employs a "*max only*" pooling which ends up with only one maximal feature per filter from last layer. All the remaining features of two subnets are further combined to produce the features representing the query document. The combined features then go through a fully connected

dropout layer before being linked to its referent entity using a fully connected linear classifier.

We performed extensive experiment to evaluate our model using the MSNBC dataset. We showed that using two channels of word embedding with each embedding matrix customized during the training produced better result than other alternatives. Our top model features one layer of entity subnet of 2048 size-1 filters, one layer of context subnet of 64 filters of size 2, 3, and 4 respectively, and 2048 neurons in the fully connected layer with dropout keep probability of 0.5. This top model can achieve an accuracy of 93.4% which stands out by a large margin from its comparator.

We also provided visualization to better understand the neural network mechanism. We retrieve the last layer output of neural network as the document code (*CNN code*), and visualize them in a 2D space via *t-SNE*. The result showed that entity mentions sharing the same canonical entity are closely clustered with small *intra-cluster distance*. We also performed *occlusion test* which draws the heatmap of the probability of right entity as a function of occluding different words in the query document. It reveals interesting context words for entity disambiguation, and they can be interpreted as the key topical words, when comparing entities to topics.

6.2 Future Work

There are several ways to extend the current line of work, and expand the scope of coverage. We explain them as following:

First of all, we can extend the domain of general web documents to special heterogeneous domains such as social network and bibliographic network. Under those settings, there is a cult of more noisy and informal wording. We can see if our model would be powerful enough to deal with such challenging environment.

Secondly, all we've been making use of so far is training set constructed from Google news and Wikipedia articles. We haven't exploited any structural and type information provided by Wikipedia (such as Wikipedia's infobox which provides type information). We might be able to

figure out a way to incorporate this information to further improve our model. Also we can include Freebase or other KBs to orchestrate better model.

In addition, we can look for a larger test dataset and cover more types of entities. The current test dataset only have entity types that typically appear in news such as political figures and places. By having a richer dataset, we can see if our model is flexible enough to deal with the complexity and intricacy presented by the real world.

We could also construct more visualization primitives. There's work in computer vision that provides *de-conv* [56] [45] techniques to synthesize input signals that maximally activate neurons. By presenting this synthetic signals, which in our case is a synthetic document, we can understand the "template of entities" developed by the neural network, and see if it matches our understand of entities.

Last but not least, we might be able to build an online document reader that pops up a window whenever the user choose to disambiguate a given span of textual words. The popped window will show the Wikipedia article of the linked entity, if the selected words can be linked. However this requires a robust model that properly trains for all Wikipedia entries, which means the model needs to scale well. It would need a better way to construct the training dataset if it's to be deployed with all the Wikipedia entries. Making use of the anchor text within the Wikipedia might be a good starter, but it is of limited coverage and there may not be enough number of samples for each entity entry. As a result, underfitting is expected, which should be addressed in future work.

Bibliography

[1] H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. R. Shadbolt. Automatic ontology-based knowledge extraction from web documents. *IEEE Intelligent Systems*, 18(1):14–21, 2003.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[3] R. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Proceesings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, pages 9–16, Trento, Italy, 2006.

[4] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1306–1313. AAAI Press, 2010.

[5] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on neural networks*, 3(5):698–713, 1992.

[6] M.-W. Chang, V. Srikumar, D. Goldwasser, and D. Roth. Structured output learning with indirect supervision. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 199–206, 2010.

[7] X. Cheng and D. Roth. Relational inference for wikification. *Urbana*, 51:61801, 2013.

[8] O. Cordón, F. Herrera, and P. Villar. Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Transactions on fuzzy systems*, 9(4):667–674, 2001.

[9] S. Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 708–716, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[10] M. Dredze, P. McNamee, D. Rao, A. Gerber, and T. Finin. Entity disambiguation for knowledge base population. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 277–285. Association for Computational Linguistics, 2010.

[11] G. Durrett and D. Klein. A joint model for entity analysis: Coreference, typing, and linking. *Transactions of the Association for Computational Linguistics*, 2:477–490, 2014.

[12] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1535–1545, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[13] J. R. Finkel, A. Kleeman, and C. D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[14] M. Francis-Landau, G. Durrett, and D. Klein. Capturing semantic similarity for entity linking with convolutional neural networks. *CoRR*, abs/1604.00734, 2016.

[15] Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[16] X. Han, L. Sun, and J. Zhao. Collective entity linking in web text: A graph-based method. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 765–774, New York, NY, USA, 2011. ACM.

[17] X. Han and J. Zhao. Named entity disambiguation by leveraging wikipedia semantic knowledge. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 215–224, New York, NY, USA, 2009. ACM.

[18] S. Haykin and N. Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[19] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 782–792, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[20] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[21] H. Huang, L. Heck, and H. Ji. Leveraging deep neural networks and knowledge graphs for entity disambiguation. *CoRR*, abs/1504.07678, 2015.

[22] H. Ji and R. Grishman. Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1148–1158. Association for Computational Linguistics, 2011.

[23] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[24] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale

video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[27] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[28] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[29] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[30] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[31] C.-T. Lin and C. S. G. Lee. Neural-network-based fuzzy logic control and decision system. *IEEE Transactions on computers*, 40(12):1320–1336, 1991.

[32] O. Medelyan, I. H. Witten, and D. Milne. Topic indexing with wikipedia.

[33] R. Mihalcea and A. Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 233–242, New York, NY, USA, 2007. ACM.

[34] D. Milne and I. H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 509–518, New York, NY, USA, 2008. ACM.

[35] V. Ng. Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1396–1411, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[36] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43, 2014.

[37] S. Pradhan, L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue. Conll-2011 shared task: Modeling unrestricted coreference in ontonotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, CONLL Shared Task '11, pages 1–27, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[38] D. Rao, P. McNamee, and M. Dredze. *Entity Linking: Finding Extracted Entities in a Knowledge Base*, pages 93–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[39] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, 2011.

[40] B. T. C. G. D. Roller. Max-margin markov networks. *Advances in neural information processing systems*, 16:25, 2004.

[41] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38, 1998.

[42] W. Shen, J. Han, and J. Wang. A probabilistic model for linking named entities in web text with heterogeneous information networks. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1199–1210, New York, NY, USA, 2014. ACM.

[43] W. Shen, J. Wang, P. Luo, and M. Wang. Linking named entities in tweets with knowledge base via user interest modeling. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 68–76, New York, NY, USA, 2013. ACM.

[44] A. Sil and A. Yates. Re-ranking for joint named-entity recognition and linking. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, CIKM '13, pages 2369–2374, New York, NY, USA, 2013. ACM.

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] W. M. Soon, H. T. Ng, and D. C. Y. Lim. A machine learning approach to coreference resolution of noun phrases. *Comput. Linguist.*, 27(4):521–544, Dec. 2001.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[48] E. K. Stephen Guo, Ming-Wei Chang. To link or not to link? a study on end-to-end tweet entity linking. In *NAACL-HLT 2013*, June 2013.

[49] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM.

[50] Y. Sun, L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang. Modeling mention, context and entity with neural networks for entity disambiguation. In *IJCAI*, 2015.

[51] C. Sutton and A. McCallum. An introduction to conditional random fields. *Found. Trends Mach. Learn.*, 4(4):267–373, Apr. 2012.

[52] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(Sep):1453–1484, 2005.

[53] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 384–394, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[54] Y. Wang, K.-F. Loe, and J.-K. Wu. A dynamic conditional random field model for foreground and shadow segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(2):279–289, 2006.

[55] I. Witten and D. Milne. An effective, low-cost measure of semantic relatedness obtained from Wikipedia links. In *Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: an Evolving Synergy, AAAI Press, Chicago, USA*, pages 25–30, 2008.

[56] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.