UNIVERSITY OF FREIBURG

BACHELOR THESIS

NEDard: Using Multi-Sense Embeddings for Named Entity Disambiguation

Author:
Felix JABLONSKI

Supervisor:
Professor Dr. Hannah BAST

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

at the

Chair of Algorithms and Datastructures
Computer Science

March 28, 2019

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum:

Unterschrift:

UNIVERSITY OF FREIBURG

Abstract

Faculty of Engineering
Computer Science

Bachelor of Science in Computer Science

NEDard: Using Multi-Sense Embeddings for Named Entity Disambiguation

by Felix JABLONSKI

English This thesis deals with the task of Named Entity Disambiguation. When analyzing a text, it is important to disambiguate mentions of entities such as people, companies, or concepts so that multiple appearances of the same entity can be identified as such. One way to disambiguate entities is to link them to their unique entity in a so-called knowledge base. A knowledge base is a source that stores information about objects like companies, fruits, or people. In this thesis given a text with mentions of such entities, ambiguous mentions are disambiguated and linked to the correct entity in Wikidata. For example "Apple" as fruit or as the company. This thesis proposes two new models (called NEDard and NEDardv2) to solve this task. Machine learning is used to derive a comparable representation of mentions depending on their textual context, and this representation is used to perform the disambiguation. The models are compared to other approaches on different evaluation sets. The evaluation has shown that the proposed models do not solve the NED task better than the baselines. However, it has been shown that there is potential in combining the proposed models with others.

German Diese Arbeit befasst sich mit der Lösung des Named Entity Disambiguation Problems. Eine Knowledge base ist eine Informationsquelle, die Informationen über Entitäte beinhaltet. Solche Entitäten sind zum Beispiel Firmen, Personen oder Früchte. Gegeben einem Text mit Nennungen (Mentions) von diesen Entitäten (Entities) - wie Personen und Firmen - sollen mehrdeutige Nennungen mit der richtige Entität aus einer Knowledge base verknüpft werden. Ein Beispiel dafür ist "Apple" als Frucht oder als Firma. In dieser Thesis wird Wikidata als Knowledge base verwendet. Es werden zwei neue Modelle vorgeschlagen, die mit Hilfe von Machine learning, die Bedeutung eines Wortes aus dessen Kontext ableiten und diese Information zum Erkennen der Entities verwenden. Die Modelle werden mit bestehenden Modellen auf mehreren Evaluationsdatensätzen verglichen. Die Evaluation hat gezeigt, dass die vorgeschlagenen Modelle nicht die naiven Ansätze schlagen können. Allerdings zeigte sich Potential bei der Kombination der vorgeschlagenen Modelle mit anderen Ansätzen.

Acknowledgements

Thanks to Prof. Dr. Hannah Bast and Mr. Niklas Schnelle for their support and supervision.
Special thanks to my fellow bachelor and master students for their motivation and advice.

Contents

List of Abbreviations

NED	**N**amed **E**ntity **D**isambiguation
NER	**N**amed **E**ntity **R**ecognition
KB	**K**nowledge **B**ase
MSE	**M**uli **S**ense word **E**mbeddings

Chapter 1

Introduction

1.1 Organization of the thesis

- In this chapter, an overview of the topics, challenges, the proposed algorithm, and its results is presented.
- Chapter 2 describes related work in comparison to this approach.
- Chapter 3 deals with the generation of the evaluation set from the Wikipedia corpus and the entity context extraction from the Wikidata knowledge graph.
- Chapter 4 introduces the techniques used. It explains the proposed models in detail. This chapter covers theoretical analysis.
- Chapter 5 discusses the results of the evaluation. This chapter covers empirical analysis and potential points of failure. In the end, a conclusion with possible improvements in future work is given.

1.2 Definitions

1.2.1 Named Entity Disambiguation

Named Entity Disambiguation (NED) is a task in natural language processing (NLP). NLP is a field of computer science that is dealing with the processing of natural languages such as text and speech. A knowledge base is a source that stores information about objects. Wikipedia and Wikidata are examples of knowledge bases. Such objects are called entities. A Named Entity is an entity in a knowledge base that can be referenced by a mention in a text. As an example, the mention "Apple Inc." is referencing to the "Apple Inc." company entity. A mention in this sense consists of one or more words. Other entity examples are persons, companies or cities. Recognizing mentions of such entities is called Named Entity Recognition (NER).
But some mentions are ambiguous. The word "Windows" can describe the operating system "Microsoft Windows" or the "Windows" in a house. The same applies to "Apple" as in "Apple (company)" and "Apple (fruit)". Selecting the correct entity is called Named Entity Disambiguation.

Consider the following example:

Windows is a widely used **operating system**, but **Jobs** and **Linus** wanted to build their own. Either **open source** or with an **apple** on it, we like both.

The desired entity for each mention are the following:
"Windows" → "Microsoft Windows" → wikidata.org/wiki/**Q1406**

"operating system" → "operating system" → wikidata.org/wiki/**Q9135**
"Jobs" → "Steve Jobs" → wikidata.org/wiki/**Q19837**
"Linus" → "Linus Torvalds" → wikidata.org/wiki/**Q34253**
"open source" → "open source" → wikidata.org/wiki/**Q1130645**
"apple" → "apple (fruit)" → wikidata.org/wiki/**Q89**

Wikipedia Wikipedia is an open-source encyclopedia with 5,801,234 content articles (February 2019)[1]. Articles are stored as text with additional knowledge represented as info-boxes and tables. The text of Wikipedia articles contains links to other articles (c.f. fig. 1.1). Wikipedia itself does not feature any complex relationships between articles. However, Wikipedia structures the articles in sections, and most articles have a similar structure.

FIGURE 1.1: Example of links in a Wikipedia article. Links are colored blue in the text.

Wikidata Wikidata is an open source knowledge graph that contains 53,646,279 items (entities) (January 2019) and relations between them[2]. "In Wikidata, items are used to represent all the things in human knowledge, including topics, concepts, and objects."[3]. Wikidata provides structured information about relations between entities. Wikidata stores relations in the form of (subject, predicate, object) triples. E.g. "(California) is (in) the (United States)" (c.f. fig. 1.2). Wikidata features most of the articles in Wikipedia as an entity. Wikidata provides a structured graph for knowledge retrieval and extends the abilities of Wikipedia as a source of knowledge. Wikidata is especially useful for computer-based processing because computers are better in dealing with structured data then unstructured (e.g., raw textual) data. A Wikidata entity is denoted by an id starting with "Q" and a number (e.g. "Q99" for California).

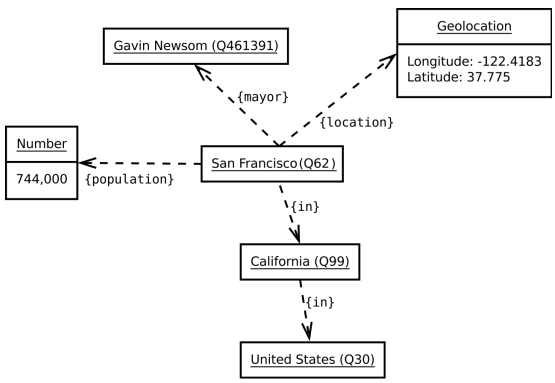

FIGURE 1.2: Example for Wikidata entity relations. Source:[2]

1.3 Motivation

The motivation for creating this thesis is to evaluate a novel way of linking text to knowledge base entities. This linking is useful to process ambiguous search queries and to enrich an article with links to further details. Displaying information about an entity requires the correct identification of this entity. As an example, when "Apple" as the company is recognized in a text, details about the company should be displayed. However, it would be wrong to display nutrition details for an "Apple" computer.

1.4 The objectives

The objective of this thesis is to create a model that links found Named Entities in a text to Wikidata entities. An NER tagger provides the Named Entities. The model therefore only performs the disambiguation (NED) part. As a source for the evaluation set, Wikipedia is used. In the text of Wikipedia articles, all links to other Wikipedia articles are remembered and linked to their Wikidata entity. Each proposed model is then given the plain text of the article together with all mentions that reference to Wikidata entities. Each model is evaluated on its ability to link the correct Wikidata entity. Each model is compared to two baselines and a reference model. The evaluation set is described in Chapter 3.

1.5 Challenges

In general, computers lack knowledge about natural language and real-world objects. One of the main challenges is, therefore, to teach them such knowledge. The correct entity needs to be identified given the natural-language context in which it appears. Therefore NED involves both natural language understanding and knowledge about entities and their relationship.
The three main challenges of NED are:

- Identify candidate entities for each mention.
- Choose the best candidate based on the context of the mention.
- Match partial mentions such as "Windows" to "Microsoft Windows" or "Obama" to "Barack Obama".
- Identify specific versions of the same thing. E.g., "Berlin" and "Alt-Berlin" as the current and historical entity.

1.6 Solutions

The thesis proposes two models for solving the NED task. NEDard and NEDardv2. With the recent progress in the word embedding technology, multi-sense word embeddings (MSE) are used. A word embedding is a dense vector representation of a word. The embedding encodes relevant information about the word. E.g., it incorporates semantics when the embedding model learned to predict a word's context. Word embeddings are introduced in detail in Chapter 4. Multi-sense word embeddings have different embeddings per word. One embedding for each sense/meaning

of the word further called sense embeddings. This is used to disambiguate the meaning of words and to link them to entities. More precisely, the NED task is solved using the word sense disambiguation capabilities of the MSE models. For the implementation the Sensegram[33] MSE model is chosen.

The main idea behind both models is, for each word in the mention, to identify the best sense and its embedding given the context words around the mention. A vector index is built up front on Wikidata entities and a textual context for each entity. During query time, the most similar entities are selected given an embedding representing the sense of the mention given its context. This is done by comparing the embedding for the mention with the learned embeddings for the entities. The system exploits that related embeddings are more similar than not related ones using a vector similarity metric. NEDard and NEDardv2 differ only in their way of dealing with multi-word mentions and entities (e.g., "Microsoft Windows"). NEDard performs disambiguation on a per word basis. NEDardv2 performs disambiguation on a per mention/entity basis. In comparison to other approaches, the models derive the embedding from pre-trained embeddings instead of learning them for the task. The exact implementation of both NEDard models can be found in Chapter 4.
This approach has the following advantages:

- Information about the context and meanings of each word are learned in an unsupervised manner, not requiring manual tagging and annotation.
- In the embedding vector space, the computer sense of "Windows" is close to the entity "Microsoft Windows" solving the partial mentions issue.
- Even words that are not featured in entity labels can be linked in a meaningful way. Given that "Apple One" (computer) has no entity in the knowledge base, but a sense in the embedding model. Then "Apple One" can be linked to "Apple (company)" due to their sense embedding similarity.
- New entities can be added to the index on the fly without retraining the rest.
- There is no need for costly training of an embedding model as pre-trained vectors can be used.
- The models can be easily adapted to any other knowledge base by generating a similar file with entities and their contexts.

1.7 Results

All models have been evaluated with different settings against different evaluation sets. Each evaluation set features different characteristics. One evaluation set contains more ambiguous mentions, and one contains only single-word mentions. Insights are gained on the pitfalls of using multi-sense embeddings for the NED task. The two proposed models do not advance on the NED task on their own. They might improve a specific part of the task and therefore are useful in combination with other techniques. The best results obtained on an evaluation set is having 74,039 out of 100,320 disambiguations correct (accuracy of 0.738).
In comparison, the baseline that is always choosing the most likely entity for a mention achieved 0.895 accuracy on the same evaluation set. Combining the score obtained by the proposed models with the baseline increased the accuracy on more ambiguous evaluation sets. The best combination accuracy on the test set of a linear

combination is 0.650 in comparison to 0.616 of the baseline alone.
The detailed evaluation results are presented in Chapter 5, discussing possible points of failure and improvements to the model in future work.

Chapter 2

Related work

This chapter briefly describes other approaches to solve the NED problem.

In his Master Thesis **Entity Disambiguation using Freebase and Wikipedia** Ragavan Natarajan builds a knowledge base out of Wikipedia and Freebase (a knowledge graph similar to Wikidata)[31]. The knowledge base is built by linking a normalized mention (so-called key phrase) to all Wikipedia articles it can refer to. This is done by looking at all Wikipedia internal links from Wikipedia articles. As an example, for the key phrase "casablanca" the entities "Casablanca (film)" and "Casablanca (volcano)" are stored. A key phrase can also consist of multiple words like "air transport". A given text is then split into n-grams (consecutive word pairs up to n words). From the n-grams, potential key phrases (mentions) are selected. Therefore the model has its own entity recognition step. With multiple metrics, a score for relations between entities from the knowledge base and a score between the key phrases in the given text is computed. A graph is used to propagate these scores as evidence and link each key phrase to the entity with the highest score. This is done by adjusting the relationship scores with propagated values from adjacent nodes. In his thesis, the entities used were articles from Wikipedia.
This differs from this thesis as it computes similarity scores over the whole document, and not for each mention and context on its own. The propagation done in this thesis is by the other mentions being in the context of or sharing the context with the current mention. Also, the similarity in vector space allows implicit propagation between words.
The paper **Joint Learning of the Embedding of Words and Entities** by Yamada et. al (2016) proposes a model that adapts and extends the Skip-gram model of word2vec[30] in order to jointly learn embeddings for words and entities[24]. It maps mentions and entities to the same vector space. It allows a direct comparison between entities and words. The NED step consists of creating an embedding from the context of the mention and training a model to match it to entity embeddings.
In comparison to the two models proposed in this thesis, the model of the paper trains the entity embeddings actively instead of deriving them from an existing model. Also, the authors published the Wikipedia2vec tool, which allows creating the embeddings with a python script[36]. However, Wikipedia2vec does not provide a way of performing NED.
DeepType is a Type System developed by Raiman and Raiman in 2018 that can be used to achieve state of the art NED performance[29]. Their model learns around 100 categories from Wikipedia which can separate all entities the best. The best categories should be easily inferred from context and still discriminate entities well. These categories form the Type System. To learn the Type System, all possible entities for mentions are extracted from Wikipedia internal links in Wikipedia articles. Their model extracts all Wikipedia links with their target article as an entity and

link text as its mention. Afterward, for each entity, Wikidata is used to get the set of categories to which it belongs. E.g., the "jaguar" car entity belongs to the categories "Jaguar vehicles", "Jaguar Cars", "Car brands", "British brand". For each category, a binary classifier is trained to predict if the mention has this category or not. The performance of each classifier becomes the "learn-ability score" for that type. More precisely the Area under the curve (AUC) metric is used as a performance score. Around 100 categories are selected which separate all entities extracted from Wikipedia the best. This selection is based on the learn-ability score and count statistics. An LSTM neuronal network is then trained on each mention and the categories of its target entity. It learns to predict the categories for a mention given its context as "is in category" probability vector. At retrieval time the LSTM is used to infer the type vector of a mention and its context. The correct entity for the mention is then selected by matching the categories to the type vector. The NED step is not available publicly.

DBPedia Spotlight is a tool developed by Mendes et. al in 2011 that allows recognizing or disambiguating entities in text and to link them to the DBPedia knowledge graph[32]. DBPedia is a knowledge graph featuring Wikipedia articles and links to Wikidata entities. They build a Vector Space Model that is based on an entity context matrix. In the matrix each row representing an entity and each column representing a word. Therefore, each entity is represented by a vector made of word counts. For each paragraph in Wikipedia mentioning the entity all words in this paragraph are counted. These word counts are summed up over all paragraphs resulting in the vector for the entity representing the counts for each word that ever occurred in the same paragraph. The resulting matrix, therefore, features contextual information about each entity.

During disambiguation, candidate entities are selected, and a representation for the mention is computed by counting the words in its context. Resulting in a word count vector comparable to the entity context matrix. The entity context matrix and the word count vector are then normalized using *tficf*. *tf* is the count of the context word for each entity. *icf* is the inverse frequency of the word over all candidates for the mention calculated by $log(\frac{|Candidates|}{|Candidates\_with\_word|})$. The goal is to normalize the importance of words for disambiguation. All candidates are then ranked by cosine similarity of their normalized context vectors with the mentions context vector. This results in the entity, which context matches the mentioned context the most.

In comparison to the model in this thesis, DBPedia Spotlight uses a word occurrence matrix and no word embeddings. DBPedia Spotlight is used as a reference for this paper as it provides a reproducible NED step.

Chapter 3

Data sets

3.1 The evaluation set

In order to evaluate the performance of each model, an evaluation set is crafted out of the Wikipedia corpus. The core idea is to take a large set of mentions (link texts) linking to the correct entity (articles) as ground truth.
The generation involves the following steps:

- Take a Wikipedia data dump with all articles and get the MediaWiki markup for each article[4].
- Ignore disambiguation pages or articles that only redirect to other articles. They do not provide the context we would expect from regular articles.
- Remove all information except links to other articles from the markup. For example tables, infoboxes and charts or external links.
- Replace all inter-article links with their plain text and remember their position, text and target article.
- Resolve the Wikidata id of the target articles to generate the ground truth. Discard links without Wikidata id. However, most Wikipedia articles can be resolved to a Wikidata entity.
- (Optional) NER process the plain text using the Spacy[5] NLP toolkit. Then match the identified named entities to the found links. The goal is to remove noise from mentions (e.g., "click here") and other mentions that are no proper names. It also gives a score for the real world "plain text" scenario without taking different NER tagger into account. The resulting mentions are a subset of the original ones. In this thesis, another set without this filtering is evaluated, too.

The result is a file with the plain text of the article and a file with the extracted mentions with their Wikidata ids. In the file start is inclusive, and the end is exclusive, both measured in UTF-8 char positions. A tab separates each column:

```
start_position end_position Wikipedia_link mention_text Wikidata_id
```

During an evaluation, only the position of the mention and the text is passed to the model.

3.1.1 The Wikipedia data dump

Wikipedia can be downloaded as a data dump directly from the Wikimedia website[6].

3.1.2 The inter-article links

The Wikipedia data dump and the markup contains much more information than needed for the evaluation set such as tables, images, and templates. A template is a code that generates elements when the markup is compiled to HTML. These additional elements add noise to the text as they do not contain natural sentences. The relevant information are the inter-article links denoted by either:

```
[[wikipedia_article]]
```

resulting in "Wikipedia_article" as link text,
or denoted by:

```
[[wikipedia_article|text]]
```

resulting in "text" as link text. As an example:

```
[[Freiburg_im_Breisgau|Freiburg]]
```

There exist many parsers to extract information out of the data dump file[7]. For this thesis, the Gensim wikisegments[8] script is used to load the Wikipedia data dump. The mwparserfromhell[9] library is modified to change everything to plain text keeping all inter-article links as follows:

```
[[wikipedia_article|link_text]]
```

The links are then extracted using a regular expression. Any redirect and disambiguation (identified by name_(disambiguation)) pages are ignored in the process.

For future work, we recommend using the wikiextractor[10] tool, because it is faster and has less overhead. With the program call

```
WikiExtractor.py --filter_disambig_pages --no-templates -l <input>
```

we can extract articles with plain text and links as `<a href="/xy">Text</a>` and parse them with a simple regex.

3.1.3 Resolving Wikidata entity ids

In order to resolve the Wikidata ids a file from Niklas Baumert's bachelor thesis (2018) mapping Wikipedia article to Wikidata id is used[26]. The mapping has been extracted using the QLever[11] engine. Each line links a Wikipedia article to its corresponding Wikidata entity id. The mapping also links Wikidata discussion Entities for a few Wikipedia articles to these articles, resulting in a Wikipedia article pointing to multiple Wikidata entities. The Wikidata entity with the smallest id is preferred for each Wikipedia article as a heuristic for the real entity. As an example, "Q14412542" is the entity for the "Wikimedia template Latest stable software release/Microsoft Windows" linking to the article "Microsoft_Windows". The correct entity for "Microsoft_Windows" is "Q1406". As observed this approach works for the evaluation set.

3.1.4 Recognizing named entities

According to the given objective mentions are identified by performing POS tagging of each sentence with Spacy[5] and selecting maximal NNP (proper noun) sequences.
Part of speech tagging (POS) is an algorithm that identifies the grammatical structure of a sentence.

"The **Apache Project** also supports **Windows**." has the tags
"The/DT Apache/NNP Project/NNP also/RBS supports/VBZ Windows/NNP.".

With the POS tags DT (determiner), RBS (adverb, superlative) and VBZ (third person verb).
In this case the maximal NNP sequences "**Apache Project**" and "**Windows**" are identified as named entities.

3.2 The entity context

In order to learn an embedding representation for each Wikidata entity, a context file with

```
Wikidata_id -> label -> raw textual context
```

is extracted from Wikidata.

3.2.1 Extracting the context

The context is extracted using the QLever[11] SPARQL engine. SPARQL is a query language designed for relationship-based knowledge bases like Wikidata[12]. The result is then downloaded as a TSV file and processed to match the desired file format. This includes filtering out "P" (predicate) entity ids, "Templates:*", "Category:*" and "Portal:*" labeled entities. Predicate entities are the entity representations of the relations between entities.

Description only

A simple approach is to extract only the description of each entity as context.

```
PREFIX wikibase: <http://wikiba.se/ontology-beta#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <http://schema.org/>
SELECT DISTINCT ?thing ?thing_label ?description WHERE {
  ?thing rdfs:label ?thing_label .
  ?thing schema:description ?description .
  FILTER langMatches(lang(?description), "en") .
  FILTER langMatches(lang(?thing_label), "en")}
ORDER BY ?thing
```

```
Q1000006    Florian Eichinger       German film producer and screenwriter
```

```
Q1000007    IFA S4000               truck
Q1000008    Neuvireuil              commune in Pas-de-Calais, France
Q1000009    Neuville-Saint-Vaast    commune in Pas-de-Calais, France
Q100000     Cadier en Keer          town in Limburg, the Netherlands
Q1000010    Königsmitteltor         city gate of Aachen, Germany
Q1000011    Neuville-sous-Montreuil commune in Pas-de-Calais, France
```

Subclass of

A more involved approach includes the descriptions of all "subclass_of" related parents of the entity. Or only their labels, whatever one desires. Adding subclass information extends the context with broader information used during entity learning.

```
PREFIX wikibase: <http://wikiba.se/ontology-beta#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <http://schema.org/>
SELECT DISTINCT ?thing ?thing_label ?description GROUP_CONCAT(?o_desc)
WHERE {
  ?thing rdfs:label ?thing_label .
  ?thing wdt:P31 ?other .
  ?thing schema:description ?description .
  ?other rdfs:label ?other_label .
  ?other schema:description ?o_desc .
  FILTER langMatches(lang(?other_label), "en") .
  FILTER langMatches(lang(?description), "en") .
  FILTER langMatches(lang(?o_desc), "en") .
  FILTER langMatches(lang(?thing_label), "en")}
GROUP BY ?thing ?thing_label ?description
```

```
Q1000006    Florian Eichinger
German film producer and screenwriter
common name of Homo sapiens, unique extant species of the genus Homo
Q1000007    IFA S4000
truck type of large automobile
Q1000008    Neuvireuil
commune in Pas-de-Calais, France
France territorial subdivision for municipalities
Q1000009    Neuville-Saint-Vaast
commune in Pas-de-Calais, France
France territorial subdivision for municipalities
```

3.3 The candidate data set

Most models evaluated in this thesis rely on a set of candidates for a mention. For this purpose, a knowledge database crafted from Wikipedia by Niklas Baumert in 2018 is used [26]. It contains, for each mention, all the Wikipedia articles linked to as well as their relevance score. The relevance score is the probability of the article being the target article of the mention. This is calculated by counting the number of

times each entity is the target of this mention over all of Wikipedia. The knowledge database is available as an SQLite database or a text file with the following schema:

mention (lnrm)	wiki link	relevance	Wikidata
lrnm__freiburg	Freiburg_im_Breisgau	0.99	Q2833
lrnm__freiburg	Canton_of_Fribourg	0.01	Q12640
...	...	...	...

The lnrm is a normal form for text, stripping it of diacritics, lower casing, and removing all non-alphanumeric characters. The candidates for mention m are all Wikipedia articles a with $m \in linktexts(a)$. Wikipedia articles are then resolved to their Wikidata entities as mentioned above.

Chapter 4

Theoretical analysis

4.1 Prerequisites

4.1.1 Word Embeddings

Teaching a computer about the meaning and semantics of words is a tough task. Natural language is ambiguous on a word and a sentence level. "Apple" can describe the fruit or the company. "I saw the man holding the apple" is ambiguous as it is unclear who is holding the apple.
Encoding words can be done by using one-hot vector representations with a dimensionality of the length of the vocabulary. While this gives a unique representation for each word, it does not include information about semantics. With one hot vector representations, words cannot be compared to each other. Also, the vector size introduces a considerable overhead to the representation.

The most recent way of overcoming this issue is to train dense, high dimensional vectors (so-called embeddings) for each word on text corpora. Embeddings incorporate information about each word, such as their semantic relationship, which can be utilized in different ways. Embeddings can be learned by a neural network optimizing an objective such as the similarity between embeddings of related words. This is called distributed semantics model as the semantics of a word is defined by its relationship to other embeddings. A common dimensionality for those embeddings is between 100 and 500 dimensions, most often 300. More dimensions allow more information to be encoded while having an impact on training time. More dimensions have an impact on downstream dimension size sensitive models and metrics. More dimensions can result in more data needed for training as more information is needed to learn more complex relationships.

In 2013 the word embedding approach got a huge boost by the paper of Mikolov et al.[30] introducing Word2Vec as a fast and easy way of computing word embeddings.
In each step, a word from the text is selected as the center word, iterating through all the words in the corpus. The model applies a neural network to the center word and all the words around it, predicting their co-occurrence; e.g., "what words are used alongside 'Apple'" The model is based on the assumption, that one can identify a word "by the company it keeps" (Firth, J. R. 1957). Therefore words with similar contexts result in more similar vectors.
In word2vec a neuronal network with one hidden layer is used. In comparison to other word embeddings models, the network only has linear activation functions except for the output layer, and thus, improving training speed. The network is trained on a text corpus to predict probabilities of word occurrences near other words. The

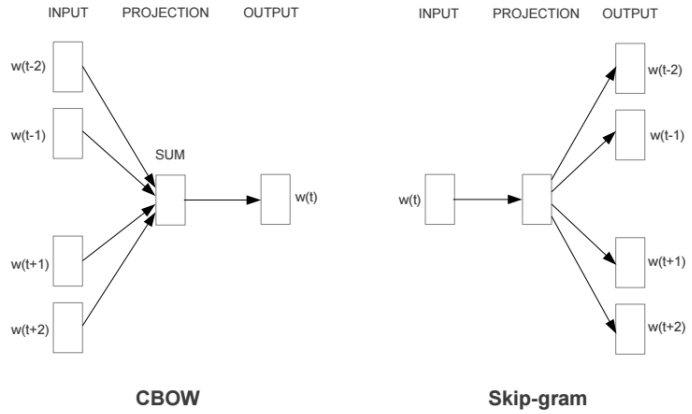

FIGURE 4.1: Network structure of both word2vec models. w(t) being the word at position t. Source:[30]

word2vec model is an unsupervised technique requiring only (large) text corpora as input and no labeled data. An input to word2vec is a one hot vector representing a specific word and the output is a probability vector for other words near it. Word2Vec optimizes the weights in the input and output layer using backpropagation. Those weights correspond to specific words, therefore becoming optimized word embeddings after training. The size of the hidden layer, therefore, determines the dimensionality of the embeddings.

Word2vec either works with the CBOW (Continuous bag of words) or the Skip-gram approach. The network structure for both can be found in 4.1.
CBOW asks the model to predict the central word given a context. In the input layer, an average of the embeddings for context words is computed. This average is then multiplied by dot product with all output layer weights which correspond to words as center words. Then a softmax classifier predicts the probability of these center words. The model gets a penalty if the real center word is not likely enough.
Skip-gram asks the model to predict the context given the central word. For every word, combinations with its context words are sampled. The input embedding for the center word is then multiplied by dot product with the embeddings for other words in the output layer which correspond to words as context words. Then a softmax layer is used to compute probabilities of these context words. The model gets a penalty for predicting words, that are not in the context.
Skip-gram works well even with small amounts of data and can represent rare words well [37].
On the other hand, CBOW is faster by several magnitudes and with slightly better accuracy for frequent words [13].
CBOW also is independent of the word order. Skip-gram can discount words that are further away from the center word by sampling them less frequently during training. To speed up the training, multiple techniques are used. For example, subsampling frequent words together with techniques to speed up the softmax layer.
After training the input-layer weights are extracted and saved as embeddings. In theory, the output weights could also be used as embeddings, but in practice, the input weights are referred to as word embeddings. The rest of the network is discarded unless further training is desired [14] [15] [16].

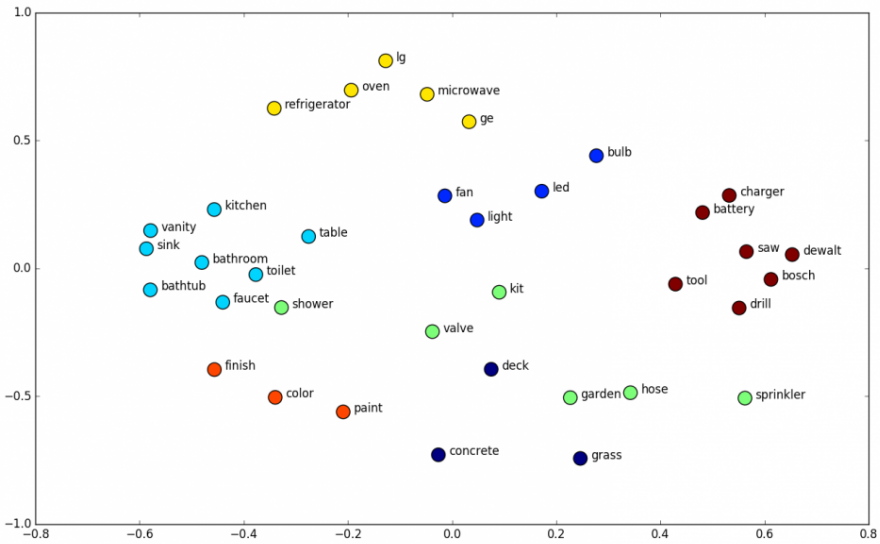

FIGURE 4.2: Visualization of word embeddings projected to 2D using t-sne. Clusters indicate semantic relations. Source: [18]

Word embeddings have a lot of interesting properties. For this thesis, we focus on the ability to compare words. As shown in figure 4.2, words that are related tend to be close together. Such visualizations are achieved by dimension reduction using t-sne or PCA into a three or two-dimensional space[17]. Specially t-sne is useful to visualize the relationship between embeddings in lower dimensional space. Embeddings close together in the visualization are more likely to have something in common. However, the interpretation of such visualizations should always be taken with a grain of salt as intuition can fail in high dimensional space [28]. Just because they look close together does not mean, that they have the desired property in the original space.
The similarity between embeddings can be computed with vector space metrics such as cosine similarity. Cosine similarity is the most used metric to achieve the best word similarity results. This is because word2vec optimizes dot products between embeddings of related words to be large and we want to get the direction of word relationship evidence and not its magnitude[19]. Table 4.1 illustrates a possible relationship between words.

Cosine similarity

Cosine similarity is defined as the measure of the angle between two n-dimensional vectors.

$$sim(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}|| \cdot ||\mathbf{y}||}$$

It is robust to the different vectors lengths and high dimensionality. The values are in $[-1, 1]$ where 1 is the same vector, 0 independent and -1 the opposite direction. If the values are restricted to positive only (e.g., word counts) cosine similarity is in $[0, 1]$. In information retrieval it is frequently used to determine the similarity between documents without considering their length. In addition to the similarity

WORD a	**WORD b**	$sim(a, b) \in [-1, 1]$
Coffee	Tea	0.68
Toyota	Freiburg im Breisgau	0.12

TABLE 4.1: Possible cosine similarity between words in the embedding space

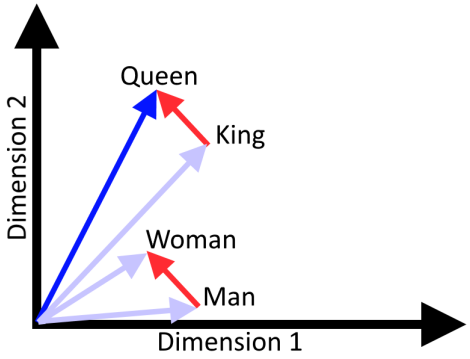

FIGURE 4.3: Visualization of the semantic reasoning with word embeddings. Each axis represents a dimension of a two-dimensional embedding. Adapted from: [21]

the cosine distance is defined as $1 - sim(x, y)$.

Additionally, word embeddings support arithmetic operations that mimic the semantics of the represented words. The most famous example is shown in figure 4.3. The vector for the queen can be derived from the vector for the king as language would suggest:

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

According to Mikolov given the word embeddings for "River" and "Russian" the sum of both embeddings will be close to "Volga river" [20].
Averaging over embeddings is known to give a representation for multiple words, sentences or even documents. This can be used to combine words in case one wants to compare documents or paragraphs.

However, it needs to be said that word embeddings bear a risk of generating fuzzy and nontransparent results. The similarity between embeddings can be unexpected. As an example, "Bad" and "Good" can be really close together by cosine similarity depending on the training. This would happen for example if the model learned that they are both used to describe the quality of things.
To be more precise if the text is always "This product is good" and "This product is bad" the context learned for both words is the same. This results in the two embeddings having a high similarity.
Therefore word embeddings need to be evaluated on a specific task, and word embeddings are or are not useful for all objectives. It is often unclear which information is encoded in the embeddings. Thus word embeddings increase the fuzziness of a model.
In a practical context, word embeddings are likely to be used as weight initialization

for other text processing models.

4.1.2 Multi-Sense Word Embeddings

While word embeddings are known to provide a measure for similarity between words, they "squeeze" all meanings of a word onto the same embedding. Therefore our "Apple" embedding will feature both meanings of "Apple Inc." and "Apple (fruit)" as it learned the same word "Apple" in two different contexts.
To tackle this issue multiple models of multi-sense embeddings (MSE) were developed.
Similar to clustering algorithms some of them need a specific number of senses per word upfront, others can identify the necessary number of senses on a per word basis.
Sense2Vec [35] is capable of learning one representation for every part-of-speech version of a word such as Apple/NN (Noun) or Apple/NNP (Proper noun). But Sense2Vec does not learn multiple representations for the same POS-Tag and is therefore not suitable for NED as entity mentions are mostly proper nouns.
Tian et al. [34] proposed a technique which is capable of learning multiple prototypes (embeddings) per word but needs a fixed amount per word up front. This amount is not determined easily.
In 2015, Bartunov et al. [25] developed the AdaGram model which uses Bayesian Nonparametrics and the Dirichlet process to modify the original word2vec skip-gram model with an additional latent variable. The model is capable of learning a different amount of embeddings per word and of increasing the amount with more text during training, if necessary. The semantic resolution is set by a hyperparameter α. Results of this model can be found in 4.2. Those results can be seen as an example of how multi-sense word embeddings represent the senses.
Pelvina et al. [33] proposed the Sensegram model, which like AdaGram is capable of learning a per word amount of embeddings. It uses an ego network approach to derive sense embeddings out of an existing word embedding model. An ego network consists of a word (ego) and related words (alters). The alters are connected by edges if they are similar themselves in embedding space. An example can be seen in 4.4. This approach allows the model to be trained on existing word embeddings. The semantic resolution is again modified by a hyper-parameter n.
As Sensegram makes up a significant part of this thesis model the following section is a summary of the original paper:

1. Take existing word embeddings or train new ones. Based on tests a context window of 3, minimum word frequency of 5 and CBOW over Skip-gram is used for better performance on sense disambiguation.

2. For each word, the most similar 200 words are selected as related words using the embedding model. 200 is motivated by prior studies as almost all words do not have more related words.

3. For each word (as ego) build a graph out of all related words as nodes (alters). Connect each of the nodes to other nodes if they are in their n most similar words.

4. On each graph do graph clustering with the Chinese Whispers algorithm [27] and compute a embedding for each cluster by the similarity-weighted average. Consider a function $\gamma : V \rightarrow \mathbb{R}$ mapping an embedding $v \in V$ to its similarity

TABLE 4.2: Nearest neighbors of meaning prototypes in embedding space learned by the AdaGram model with $\alpha = 0.1$. In the second column the relevance of each sense is given. Source: [25]

WORD	$p(z)$	**NEAREST NEIGHBOURS**
python	0.33	monty, spamalot, cantsin
	0.42	perl, php, java, c++
	0.25	molurus, pythons
apple	0.34	almond, cherry, plum
	0.66	macintosh, iifx, iigs
date	0.10	unknown, birth, birthdate
	0.28	dating, dates, dated
	0.31	to-date, stateside
	0.31	deadline, expiry, dates
fox	0.38	cbs, abc, nbc, espn
	0.14	raccoon, wolf, deer, foxes
	0.33	abc, tv, wonderfalls
	0.14	gardner, wright, taylor
rock	0.23	band, post-hardcore
	0.10	little, big, arkansas
	0.29	pop, funk, r&b, metal, jazz
	0.14	limestone, bedrock
	0.23	'n', roll, 'n', 'n

to the initial (ego) word embedding. The similarity-weighted average then becomes

$$s_i = \frac{\sum_{k=1}^{a} \gamma(vec_k) vec_k}{\sum_{k=1}^{a} \gamma(vec_k)}$$

for each cluster $C_i = \{vec_1, ..., vec_a\}$. Those cluster average embeddings then become the sense embeddings for the initial word.

Chinese Whispers does not need a number of clusters up front. The amount is determined by links between nodes and therefore by the parameter n. To remove noise clusters with a size $|c_i| < min\_cluster\_size$ are discarded. The paper's authors tested the unweighted average as well and found it to be inferior at the disambiguation task.

4.1.3 Word sense disambiguation

The desired property of disambiguation of meanings of words given a context is defined by the word sense disambiguation (WSD) task. WSD is the task to disambiguate between meaning/senses of words given a context. In contrast to NED, it does not link them to an entity in a knowledge base, but identifies the same sense in different texts.
The models proposed in this thesis are using the word sense disambiguation performance of the Sensegram to disambiguate and match mentions to entities in a knowledge base.
According to the Sensegram authors, Sensegram performs comparably to AdaGram on the WSD task [33].
Both AdaGram and Sensegram offer open source implementation and pre-trained models to solve the WSD task.

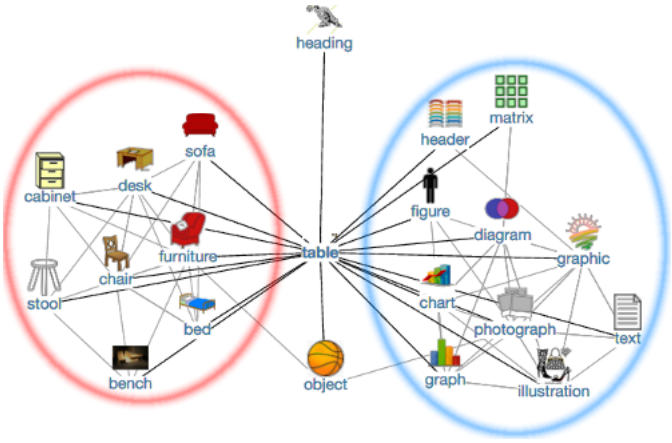

FIGURE 4.4: Visualization of the ego-network of "table" with furniture and data sense clusters. Note that the target "table" is excluded from clustering. Source: [33]

Because the Sensegram implementation is written in Python and AdaGram in not maintained Julia code (although an unofficial Python version exists), Sensegram is used.
Sensegram does not care about the order of the words in the context window, therefore, sees them like a bag of words.
The proposed "NEDard" models are capable of working with every sense embedding model that offers a disambiguation method. This method needs to provide a sense embedding given a word and its context.
Again, as Sensegram is the basis for the proposed models, a description of the sense disambiguation step is given.
1. For a word w all known senses $S = \{s_0, ..., s_n\}$ are retrieved.
2. For the context words $C = \{c_0, ..., c_k\}$ of w the average of their word embeddings(vec) $\bar{c} = k^{-1} \sum_{i=0}^{k} vec(c_i)$ is calculated.
3. The most similar sense with cosine similarity is selected as result:

$$s^* = \arg\max_i sim(s_i, \bar{c}) = \arg\max_i \frac{\bar{\mathbf{c}} \cdot \mathbf{s_i}}{||\bar{\mathbf{c}}|| \cdot ||\mathbf{s_i}||}$$

The authors of Sensegram[33] also investigated a "probability-based" approach using the "context embeddings" (the output layer weights of the network). As this approach requires the context embeddings to be known and does not improve over the similarity-based approach, they stayed with the similarity based one.
In both cases, the amount of context words is reduced to the *max_context_words* most discriminatory words. This means selecting the context words with the highest evidence towards one and against another sense. This is done because typically only a few context words are relevant for the disambiguation. For each word c_j in the context and all word senses s_i calculate $\max_i f(s_i, c_j) - \min_i f(s_i, c_j)$ as score. Where f is the disambiguation function $sim(s, c)$. Choose the top *max_context_words* scoring words. For this thesis, the default "similarity" based approach is used.

4.2 The NEDard models

4.2.1 Overview

In a nutshell, the NEDard NED step consists of the following steps:

1. Load the multi-sense word embedding (MSE) model. In this case, this is the pre-trained Sensegram model.
2. Preprocess the text and mentions to retrieve word tokens.
3. For every mention, use the MSE model to get the matching sense embedding given the surrounding context.
4. Use a vector index learned from Wikidata to get the best matching entity. Candidates are either all entities or a pre-selected subset.

Learning the NEDard index is done by:

1. Load the MSE model.
2. Fetch a context for each entity in Wikidata.
3. For every entity label, ask the MSE model for the sense embedding given the context.
4. Store the vector together with the entity id in a vector index for fast lookup.

4.2.2 Preprocessing

To illustrate the preprocessing consider the following example:

> **Windows** is a widely used **operating system**, but **Jobs** and **Linus** wanted to build their own. Either **open source** or with an **apple** on it, we like both.

As example given to the model is the text and following mentions found by an arbitrary NER tool:

Mention text	Start (inclusive)	Stop (exclusive)
Windows	0	7
operating system	25	41
Jobs	47	51
Linus	56	61
open source	96	107
apple	119	124

Given the text t and mentions M the model splits t into sub strings S between its mentions.

$$t = s_0 M_1 s_1 M_2 s_2 ... M_n s_n \qquad (M_k \in M, s_k \in S) \tag{4.1}$$

Each sub string $s_k \in S$ and mention $M_k \in M$ is then tokenized by

- Remove accents
- Remove numbers
- Replace punctuation with white spaces

- Keep a word w only if $2 \leq len(w) \leq 20$
- Keep only words known to the MSE model as unknown words are ignored by it. For mention tokens consider the other cases too if ignore_case is set to true.

Resulting in a list of tokens and an index for each mention mapping it to its tokens. The following sentence is an example including punctuation, a fictional unknown word ("Malagati") and a single character word ("a").

Microsoft's Windows10 is a widely-used **operating system** in Malagati.

It results in the following tokenized list and the mentions:

['**Microsoft'**, **'Windows'**, 'is', 'widely', 'used', **'operating', 'system'**, 'in']

Mention tokens	Start (inclusive)	Stop (exclusive)
'Microsoft' + 'Windows'	0	2
'operating' + 'system'	5	7

Now the whole list is used to select the context for each mention token in it based on its position. Stop words are not removed in this step.
This way the text only needs to be preprocessed once for all mentions.

Note on casing

The Sensegram pre-trained model used is trained on a not lowercased version of the word embeddings. Therefore different cases have both different word and sense embeddings.
The Sensegram model supports "ignore case", which means, that for the query "Florian" also the senses for "florian" are considered. The casing of the context words, however, remains unaltered. Therefore, mention words in the context of other mentions are case sensitive. As mention, they are processed according to *ignore_case* parameter.
This means that mention tokens and context tokens are stored separately. The original casing of the mention tokens is still present in the context token list. Otherwise, other mentions would lose the correct case in their context.
This thesis evaluates ignore-case and keep-case models.

4.2.3 Multi word mentions

Most entities contain more than one word in their label. To get a vector representation for those mentions, there are multiple ways one can think of. This is necessary in case the embedding model does not contain representations for multi-word entries.
First, identify the sense for each word in the mention, then do one of the following:

- Calculate the average of the sense embeddings for each word. This is often done when working with word embeddings to get composite representation. It might increase the fuzziness of the model as the outcome quality is unclear.
- Calculate the weighted average of the sense embeddings for each word. E.g., use *tf-idf* overall mentions as weight. Weighting the embedding helps to keep the semantic information as the resulting vector is closer to the more relevant

sense embedding. Botanical names, for example, have a common part (family name), but a rare specific part.

- Create its own entry in the vector index for each word in the mention. This might help if only one word has the correct sense, that we are looking for. Adding a wrong sense can destroy the entity vector. During retrieval, the different results need to be ranked to select the final suggestion. For example by highest similarity. Using this technique increases the risk of learning too similar embeddings for different entities.

In this thesis, the vector per word approach "NEDard" and the *tf-idf* weighted average approach "NEDardv2" are evaluated. The *tf-idf* score is calculated over the entity names during training. See 4.2.4. Other scores are up for future work.

tf-idf is a common weight/normalization in information retrieval. It is used to understand the importance of a word in a document compared to all other documents. E.g., normalize word counts across documents. Term frequency (tf) is the number of occurrences of the word in the document. Inverse document frequency (idf) is defined as

$$log(\frac{|documents|}{|documents\_containing\_word|})$$

The formula is $tf * idf$ for each word, document combination. As an example, if a large text with 2000 words contains a word ten times consider another text with 100 words also containing the word ten times. The 2000 words text is most likely not "about" that word as much as the 100-word text.

For multi-word mentions, every word in the mention is disambiguated on its own. To include the other words O of the mention in the context C, it is extended to

$$C_{new} = o_1 + ... + o_n + C \qquad (o_k \in O) \tag{4.2}$$

creating an own context for every word in the mention.

4.2.4 Training

Input

As input for training the previously introduced context from Wikidata is taken.

```
Q1000006    Florian Eichinger    German film producer and screenwriter
Q1000007    IFA S4000            truck
Q1000008    Neuvireuil           commune in Pas-de-Calais, France
Q1000009    Neuville-Saint-Vaas  commune in Pas-de-Calais, France
...
```

From each line, the id, label, and context are extracted. As an example, the first entry is selected:

```
Florian Eichinger       German film producer and screenwriter
```

Given this context, the model should be able to extract his name, his nationality and his profession.
In the best case, the MSE model is capable of disambiguating **"Florian Eichinger"**

into a **German**, who works in the **film industry** as a **producer**, and to return a vector representation of this "German movie maker" sense of "Florian Eichinger".
During retrieval, **"Florian Eichinger"** given a context related to movies and Germany should resolve to the learned entity.
Descriptions of the entity are a good choice for the context as they are a short, but informative description of the entity. Good descriptions contain a word for every aspect of the entity. Adding labels from related nodes in Wikidata extends the context.

Word sense disambiguation

During training, the model extracts the mentions words and preprocesses the context as mentioned above.

token	context
Florian	`['Eichinger', 'German', 'film', 'producer', 'and', 'screenwriter']`
Eichinger	`['Florian', 'German', 'film', 'producer', 'and', 'screenwriter']`

For each token a window of size *context_window* is extracted from the context starting from the left.
Stop words are removed from this window and the result is passed into the MSE model for disambiguation. Resulting in word#sense_id.

token	context window (size 5) without stop words	result of MSE
Florian	`['Eichinger', 'German', 'film', 'producer']`	Florian#0
Eichinger	`['Florian', 'German', 'film', 'producer']`	Eichinger#3

Internally the Sensegram [33] model chooses up to *max_context_words* with the highest predictive power.
The step results in a vector for every mention token that represents its sense.
The default value is to have overall 10 words (5 on each side) of the mention and to choose the 3 most predictive words out of them. Mention token for which no sense embedding could be retrieved are discarded. This can happen if the mention token is not featured in the Sensegram model.
The result of this sense disambiguation is:
`[vector("Florian#0"), vector("Eichinger#3")]`

Learning weights

The NEDardv2 model needs information about the importance of each word in a mention. This is done by learning the *tf-idf* weights for words overall entity labels in the context file. Entity labels are treated as documents and their tokens as words. Given only the entities "Washington D.C." and "George Washington", the word "Washington" appears in both. Therefore its idf is lower resulting in more weight given to the "George" part. This is done in the hope of "George" being more descriptive for an entity. NEDardv2 learns a weighting function $\phi(w, W)$ that returns the *tf-idf* weight of a word w in a list of mention words $W = \{w_0, ..., w_n\}$.

Index storing

As the retrieval depends on the fast comparison of a vector to the vectors in the index, a suitable vector index is chosen. Gensim's "KeyedVectors" [22] representation

allows querying of "most similar" entities by a vector and by entity name. Querying by an entity can be used to get similar entities to a given one. It allows adding new vectors as desired and saves the index to disc with the ability to load it as a fast memory map. It is the same index used by Sensegram and the Gensim word2vec implementation. To speed up querying in future work a nearest-neighbors index can be set on top of KeyedVectors.

The storing of the entity representation depends on the model used:

NEDard `[vector("Florian#0"), vector("Eichinger#3")]`

results in two embeddings for the entity `Q1000006` namely `Q1000006#0` and `Q1000006#1`. One links the entity to this sense of "Florian" and the other to this sense of "Eichinger".

NEDardv2 `[vector("Florian#0"), vector("Eichinger#3")]`

results in the embedding `Q1000006`. It is the *tf-idf* weighted average of the sense embeddings "Florian#0" and "Eichinger#3". Using the ϕ weighting function defined above each entity embedding becomes

$$e = \frac{\sum_{i=0}^{n} \phi(w_i, W) s_i}{\sum_{i=0}^{n} \phi(w_i, W)}$$

given the entity tokens $W = \{w_0, ..., w_n\}$ and their sense embeddings $S = \{s_0, ..., s_n\}$. In addition mention token and senses for which no *tf-idf* weight could be resolved are ignored.
After adding all entities to the index this way the index is saved to disc for later use denoted by `CONTEXT_FILE.nn`.

4.2.5 Querying

Input

Input to the retrieval method is the previously introduced text and its mention mapping.

Windows is a widely used **operating system**, but **Jobs** and **Linus** wanted to build their own. Either **open source** or with an **apple** on it, we like both.

The text is now converted to a list of tokens and an index that denotes the position of each mention in it as mentioned above.

Word sense disambiguation

For each mention, a context of n tokens to the left and n tokens to the right are selected, potentially including other mentions. Similar to the training step, stop words are filtered as this matches the way the Sensegram model works.
Analog to the training step the MSE model is queried for the best matching senses of the given mentions.

Mention	context window ($n = 5$) without stop words	result of MSE
Windows operating + system	`['widely', 'used', 'operating']` `['Windows', 'widely', 'used', 'Jobs', 'Linus', 'wanted']`	Windows#1 operating#1 + system#2

As a result we now have a sense vector for each word in the mention. Mention token for which no sense embedding could be retrieved are ignored.

Linking the entity

The missing step is to link the mention given its sense embedding(s) to an entity. Again NEDard and NEDardv2 differ in the way they handle multiple tokens per mention.
NEDard handles each mention token as own linking try and chooses the entity with the highest similarity score.
NEDardv2 computes the weighted average of the mention token analog to the training step.

$$e = \frac{\sum_{i=0}^{m} \phi(w_i, W) s_i}{\sum_{i=0}^{m} \phi(w_i, W)}$$

given the mention token $W = \{w_0, ..., w_m\}$ and their sense embeddings $S = \{s_0, ..., s_m\}$.

There are two ways of doing the entity linking evaluated in this thesis.
1. Search the full index for the best matching entity embedding and return its id.
2. Restrict the search space to entities, which are candidates for the given mention. Candidates are extracted using the knowledge base introduced in 3.3.

While
1. allows finding related entities in case the real entity is not known
2. offers better precision and speed due to restricted search space with a high probability of containing the desired entity.

1.1 **All known entities | NEDard**

- For each word in the mention take its sense embedding and query the closest entity in the index given by cosine distance.
- Return the entity that has been the closest overall words

1.2 **All known entities | NEDardv2**

- Return the entity that has been the closest to the averaged sense embedding e

2.1 **Candidate entities only | NEDard**

- Get all candidate entities and all vector representations of them inside the index. A candidate entity in NEDard has multiple representations "Q123#0", ..., "Q123#n" inside the index. One for each word in its label during training.
- For each word in the mention take its sense embedding and get distances to all candidates. Return the candidate with the lowest cosine distance (highest similarity) for each word.
- Return the candidate that has been the closest overall words in the mention. This is the candidate that contains a word with the sense embedding closest to a sense embedding of a word in mention.

2.2 **Candidate entities only | NEDardv2**

- Get all candidate entities and all embedding representations of them inside the index. A candidate entity in NEDardv2 has a single embedding "Q123" in the index representing the averaged embedding learned during training.
- Take the average sense embedding of the mention e and compare it to all candidate embeddings. Return the candidate with the lowest cosine distance (highest similarity).

Considering our example text and the **"Windows"** mention the following is achieved:

NEDard:
Inside the vector index is a "computer" sense representation learned for "Windows" and "Microsoft". Both linked to the entity called "Microsoft Windows" (Q1406) in Wikidata.
Now the mention text "Windows" is resolved to the "computer" sense embedding given the "computer" context around it.
In vector space, this sense is now close to the learned "computer" sense of "Windows" or in fact the same vector, which then is resolved to the "Microsoft Windows" entity. Now the correct entity can be returned even if the label text does only have a partial match.
Another advantage is that given "Windows Vista" is not an entity in Wikidata the "Windows" word is still resolved to the "computer" sense and the entity "Microsoft Windows" is returned, which is better than no match. In addition "Vista" can have a "Microsoft" representation in the MSE model, which is close to the sense vectors for "Microsoft" or "Windows" and is therefore linked to "Microsoft Windows" or "Microsoft".
For the mention "operating system" the most "computer" sense of "operating" and "system" are found. The target entity "Operating System" (Q9135) has two representations inside the vector index. One for "Operating" and one for "System". Each learned with the "computer" context of the entity. Now the model compares the cosine similarity of all sense embeddings for mention words with all representations inside the index. This should return the correct entity Q9135 as one of its embeddings is really close to one of the mentions embeddings. The hypothesis is that NEDard works better for single word mentions and entities.

NEDardv2:
Inside the vector index is an embedding for "Microsoft Windows" (Q1406), which is a weighted combination of the "computer" sense of "Microsoft" and "Windows". Assumed that "Windows" is the term with the higher *tf-idf*-score (more importance in entity labels) the "Windows" sense embedding has more impact on the entity embedding.
The mention text "Windows" is resolved to the "computer" sense embedding given the "computer" context around it.
This embedding should now be similar to the "Microsoft Windows" vector caused by their "computer" sense.
For "operating system" there is an embedding in the index with the weighted average of the "computer" sense of "operating" and "system". The embeddings for the words of the mention are now averaged using the same weight. This should result in a similar vector to the one in the index. Therefore, resulting in the correct entity to be matched. The hypothesis is that NEDardv2 works better for multi-word mentions.

4.2.6 Theoretical run time

Given the number of mentions during querying or the number of entities during learning as m, the time complexity can be defined as follows:
The lookup of the sense for each mention is constant in average over all mentions. This is because it depends on the number of senses per word, which is specific for each word and independent of the other words, and a fixed number of context words. The computation of the embedding for the all mentions/entity labels runs in $O(m)$.
The training, therefore, runs in $O(m)$ where m denoted the number of entities to learn.
For a query, the embedding of each mention needs to be compared to all candidates C of that mention. Resulting in $O(m * |C|)$ with $|C|$ being the upper bound for the number of candidates per mention. When using the candidate list $|C|$ is bounded by the max amount of candidates for a mention. This is 148,641 for the candidate list used in this thesis. However, the average number of candidates per mentions is 1.28 on this list. If no candidate list is used $|C|$ is the number of entities known to the model and therefore up to all Wikidata entities (53,646,279 in February 2019).

Chapter 5

Results

5.1 Evaluation setting

This section explains the setup of the evaluation and introduces the models evaluated.
In this section, a mention is a substring in text denoting an entity whereas disambiguation is a unique occurrence of a mention at a specific position. Therefore a text can contain the mention "Apple", but one disambiguation in a "computer" context and one in a "fruit" context. When referring to candidates, the candidate set introduced in section 3.3 is meant.

5.1.1 Wikidata context

To speed up run time we train the NEDard models used for evaluation on a subset of the Wikidata entities. Representations for the first 5 million Wikidata entities are learned. The first entities are the ones ordered numerically by the Wikidata id with "Q45" < "Q100". These entities cover most entities found in the evaluation set. This results in 3,661,533 learned entities. The rest of the entities could not be learned as there are missing vector representations for their label words in the underlying Sensegram model.

5.1.2 The evaluation sets

Chapter 3 introduces the evaluation set and describes its creation. Note that both NEDard models never use the evaluation sets for training.
The evaluation set used for evaluation is forged out of the English Wikipedia (2018-11-23 12:02:59). It consists of the plain text and a list of mentions in each article. This is the X_{eval} set:

Article	Articles text	Position	Mention
Windows	"Windows is a operating system..."	[12:28]	operating system
Windows	"Windows is a operating system..."	[0:7]	Windows

For each entry in X_{eval} its correct Wikidata entity is stored in y_{eval}:

Mention text	Correct Wikidata id	Wikipedia article
operating system	Q9135	Operating_system
Windows	Q1406	Microsoft_Windows

Full set The main evaluation is done on a data set created by matching found links to entities identified as named entities with the NER method introduced in section 3.1.4. The full evaluation set X_{full} contains 1.5 GB of text from the first 168941 Wikipedia articles in the data dump. It includes 5609258 mentions.

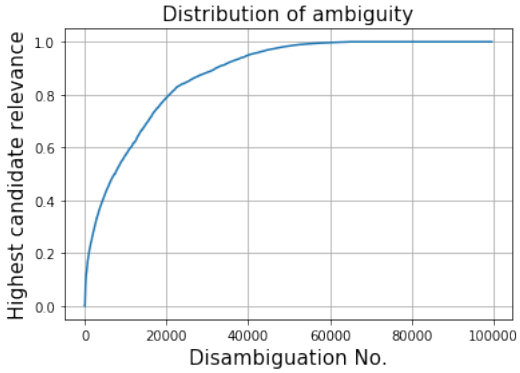

FIGURE 5.1: Ordered distribution of the maximum candidate relevance for each disambiguation inside X_{2k}. Disambiguation No. is a counter.

No NER set Another evaluation set is created similar to X_{full}, but without restricting to identified named entities. X_{noner} contains the first 2,000 articles from the Wikipedia data dump. Including 151,215 disambiguations.

2k set For this thesis, a subset of X_{full} is used. The first 2,000 articles out of X_{full} are selected, resulting in X_{2k}. As mentioned above the NEDard models are trained only on a subset of Wikidata entities. To give them a fair comparison to other models the X_{2k} evaluation set is filtered for mentions linking only to known entities. The model is trained on the first 5 million entities out of Wikidata, resulting in 3,661,533 actually learned entities.
The evaluation set is limited to the mentions linked to those 3,661,533 entities. The comparison to other models stays the same as they are evaluated on the exact same mentions.
This results in 2,000 articles and 100,320 disambiguations.
To understand the characteristics and create more evaluation sets with different characteristics, we analyze X_{2k}. We research the question is "how ambiguous is the evaluation set"? As a rating for ambiguity, the maximum relevance score for a mention's candidates is chosen. A high maximum relevance indicated one strong candidate and therefore less ambiguity of the mention. The distribution of the maximum candidate relevancy for all disambiguations inside X_{2k} is plotted in 5.1. We can see that out of the 100,320 disambiguations almost 50 percent have a single really strong candidate. Therefore, they are not ambiguous and considering a candidate other than the most relevant is certainly a bad idea.

Ambiguous set Using the insights from X_{2k}, a subset of ambiguous mentions is selected. This means selecting mentions that have more than one strong candidate. As a threshold, 0.8 for the highest relevance score is chosen, resulting in 20,626 disambiguations from X_{2k} meeting the criteria. The new evaluation set is called X_{ambig}.

Single word set To further understand the evaluation set we plot the distribution of words per mention in 5.2. This represents how many disambiguations of our evaluation set have that number of words in their mention. E.g. "Apple" has only one word, but "Microsoft Windows" has two. We can deduct that the ambiguous mentions are skewed towards one word. Therefore, for our third evaluation set $X_{oneword}$

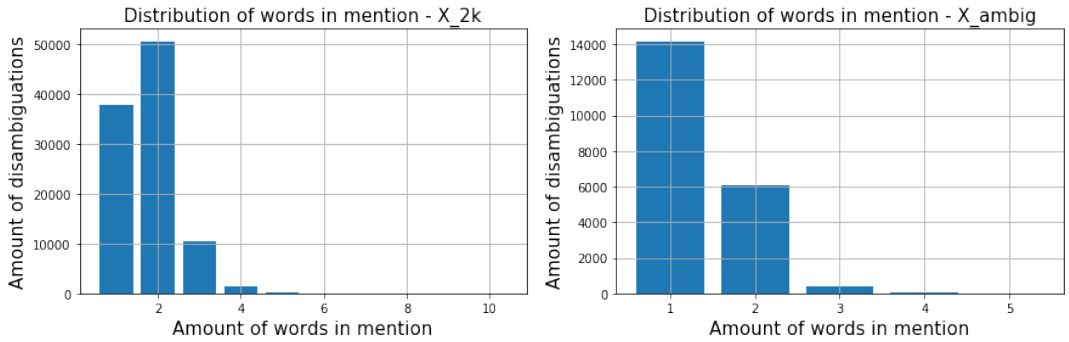

FIGURE 5.2: Distribution words per mention. Showing the number of disambiguations inside the evaluation sets with this number of words in the mention.

the single word mention subset of X_{2k} is chosen, resulting in 37,732 disambiguations.

All evaluation sets represent different aspects of NED and give us the ability to compare the models on slightly different tasks.

5.2 The models compared

To make meaningful evaluations, we compare NEDardv1 and NEDardv2 on multiple baselines as well as a naturally arising upper bound (Oracle performance).

5.2.1 Oracle performance

The Oracle performance is the performance of a model that always chooses the correct candidate. This means that it is only wrong if the correct candidate is not present in the candidate list. The Oracle performance has been calculated for each evaluation set to get to upper bound for the performance of the models.

5.2.2 Random entity

This model is the expected outcome of choosing a random entity out of the 3661533 entities learned by NEDard. This baseline gives a representation of the difficulty without using a candidate list. The chance of a disambiguation being correct is $\frac{1}{3661533} \approx 2.73 * 10^{-7}$.

5.2.3 Baseline 1 - Random choice

The first simple baseline takes a random candidate for each word with a uniform random distribution.
This simple baseline helps to get a lower bound for disambiguation. Words with few candidates will have a higher random chance, therefore modeling the difficulty of disambiguation on the given mentions. In other words, the baseline identifies the difficulty of the evaluation set.

5.2.4 Baseline 2 - Link relevance

For the second baseline, the same candidates as for baseline1 are used. For each mention text, the entity with the highest probability (relevance) of having it as link text is returned.
While this works well for the really popular meanings of mentions (such as the computer sense for "Apple"), it cannot disambiguate the less popular meaning ("Apple" as fruit).

5.2.5 Reference - DBPedia spotlight

As a reference, DBPedia spotlight is used. DBPedia links given mentions to DBPedia resources, which are linked to Wikipedia articles or Wikidata entities. DBPedia spotlight is used as a reference as it provides reproducible disambiguation only step. For evaluation, we run a local instance of the DBPedia spotlight server using Docker.

5.2.6 The NEDard models

The NEDard models used in this evaluation are trained with the following settings:

- A Sensegram model trained by the Sensegram authors is used. It contains word embeddings without lower casing for UTF-8 words.
- The embeddings used have 300 dimensions.
- Sensegram is configured to use a context window size of 10 tokens and the three most predictive words ($max\_context\_words = 3$ given to Sensegram model).
- The models are learned with the Wikidata context out of entity description and the labels of all "subclass_of" parents.
- The Wikidata knowledge graph is reduced to the first 5 million Wikidata entities for faster runtime.
- The machine used for training had 23GB of RAM and an AMD FX(tm)-8150 Eight-Core Processor with 3.6GHz clock speed.

Note: Some entities do not have a vector representation, because none of their label tokens were known to the Sensegram model.

NEDardv1

- Two models are built: One with $ignore\_case = True$ and one with $ignore\_case = False$. Therefore considering senses of all casings or only of the given one during sense disambiguation.

Both NEDardv1 variants result in a 16GB large index file with around 7 million entries. This is because an entity has one vector representation per word in its label. Training consumed 20GB RAM and took 40 hours.

NEDardv2

- A *tf-idf* model for the embedding weights is learned based on the labels of the given 5 million Wikidata entities. Considering the entity labels as documents.
- The model is trained using *ignore_case* = False. This matches the best performance in earlier testing on NEDard.

The model results in an 8.4GB large index file. Each entry is representing a different Wikidata entity. Training consumed 16GB RAM and took 30 hours.

5.3 Evaluation run

We evaluate every model on the same task:
Given a mention and the text around it from an evaluation set X_{eval} suggest the Wikidata id for this disambiguation. Considering the example in 5.1.2 a model is asked to disambiguate "operating system" and "Windows". Then a 1 is assigned if the suggestion is correct or a 0 if not, thus becoming a binary rating for each disambiguation.

No.	Mention	Suggestion	Correct	isCorrect
0	operating system	Q42 (Douglas Adams)	Q9135	0
1	Windows	Q1406 (Microsoft Windows)	Q1406	1
...				

The Metric

The NED task in this thesis has only one correct result per disambiguation. As metric the accuracy of the prediction is chosen. For each article $a \in \{a_0, ..., a_n\}$ we record the correct disambiguations $correct(a)$ and total disambiguations $total(a)$. The accuracy of the disambiguation is defined as

$$Accuracy = \frac{\sum_{i=0}^{n} correct(a_i)}{\sum_{i=0}^{n} total(a_i)} \tag{5.1}$$

Given one article with 100 mentions where 80 were linked correctly ($isCorrect = 1$) the result becomes $80/100 = 0.8$. Given two articles with one time 100 and one time 50 mentions where one time 40 and one time 80 are correct the result becomes $\frac{80+40}{100+50} = 0.8$. This metric is chosen as it is the most used metric to evaluate disambiguation-only NED models.

5.4 Empirical results

Table 5.1 lists the results for all evaluation runs. For each model and model configuration we report the accuracy.
Running the evaluation takes between 20 minutes to 1 hours depending on the evaluation set and candidate policy. Memory consumption is about 12-16 GB.

5.4.1 Oracle performance

It can be observed that the Oracle has almost 100% accuracy on all evaluation sets. This means that in theory, each model should be able to find the correct candidate. The Oracle is the upper bound for evaluation results on the evaluation sets.

Accuracy	Evaluation set #disambig.			
Model	X_{2k} 100,320	X_{ambig} 20,626	$X_{oneword}$ 37,732	X_{noner} 151,215
Oracle	0.989	0.989	0.990	0.977
Random entity	0.000	0.000	0.000	0.000
Baseline1 (Random candidate)	0.449	0.083	0.152	0.465
Baseline2 (Highest relevance)	**0.895**	**0.613**	**0.817**	**0.876**
Reference (DBpedia spotlight)	0.866	0.607	0.780	0.835
NEDardv1 (case + candidates)	0.719	0.400	0.515	0.720
NEDardv1 (nocase + candidates)	0.720	0.397	0.523	0.719
NEDardv1 (case + all)	0.288	0.265	0.548	0.240
NEDardv2 (case + candidates)	0.738	0.402	0.567	0.731
NEDardv2 (case + all)	0.313	0.142	0.241	0.379

TABLE 5.1: Results of the evaluation runs rounded to 3 decimal digits. Case = case is not ignored during disambiguation. nocase = case is ignored during disambiguation. candidates = only candidates are considered. all = all entities in the index are considered.

5.4.2 Baseline 1 - Random choice

The random choice behaves like expected on the three evaluation sets. It can be seen as a lower bound for disambiguations and a metric for the difficulty of the evaluation set. This is because a mention with few candidates has a higher chance of being correct at random. X_{2k} has a moderate difficulty with the random guess being correct in 45% of the cases. On X_{ambig} the chance of getting the correct entity at random is way less. Therefore X_{ambig} is a more demanding evaluation set. On $X_{oneword}$ the chance is also low, probably because single words are more likely to be more ambiguous.

5.4.3 Baseline 2 - Highest relevance

Although baseline 2 is rather simple, it provides high scores on all evaluation sets. In general choosing, the most relevant candidate is not a bad idea, and as the evaluation set is extracted from Wikipedia, it makes sense that baseline2 performs rather well. Baseline 2 is the best model on X_{2k} as this evaluation set is skewed towards mentions with a single relevant candidate as we saw earlier in 5.1. This means that using the prior probability to choose a candidate is a good guess and the relevancy is an essential feature for disambiguation.
On X_{ambig} the baseline does not perform equally well. This means that X_{ambig} gives a chance to models focusing on mentions with multiple strong candidates. Still, baseline 2 has the highest accuracy of all models which means that no other model has been able to disambiguate the more ambiguous mentions correctly. On $X_{oneword}$ behaves a slightly worse than on X_{2k}, which makes sense as ambiguous mentions are skewed towards single word mentions.

5.4.4 Reference - DBPedia spotlight

The reference implementation behaves similarly to baseline2. Especially on the ambiguous X_{ambig} evaluation set, we observe almost the same performance. Assuming that baseline2 has an advantage by being trained on Wikipedia links, it can be assumed that the reference implementation is slightly better than baseline2 in a real-life scenario.

5.4.5 NEDardv1

The evaluation of NEDardv1 is more involved as the models have a lot of different settings to be evaluated. The following paragraphs deal with each configuration evaluated. As seen in 5.1 none of the NEDardv1 models outperforms baseline2 on any evaluation set. While this is a sign that NEDardv1 has issues performing NED, each configuration yields interesting results leading to further analysis in 5.5. Remark: NEDardv1 is the model that calculates a sense embedding for each word in the mention and each word of an entity, and then chooses the most similar candidate over all combinations as best matching entity.

Case + candidates In this setting, the model deals with different cases of words as different words (and therefore different senses). In this setting, only candidates are considered and not all known entities.

Evaluation set	Baseline2	NEDardv1	Difference (NEDardv1 - Baseline2)
X_{2k}	0.895	0.719	-0.176
X_{ambig}	0.613	0.400	-0.213
$X_{oneword}$	0.817	0.515	-0.302

NEDardv1 has significantly worse performance than the highest relevancy baseline. Even on the X_{ambig}, which should give models an advantage which consider more than the most relevant candidate.
In other words, NEDardv1 does not choose the correct candidate and therefore has issues performing disambiguation between candidates.

The next question is if there are disambiguations where NEDardv1 manages to find the correct entity and baseline2 does not?
Analyzing both models on X_{2k} (with 100,320 disambiguations) baseline2 fails at $10,492$ disambiguations. NEDardv1 fails at $28,141$ disambiguations. Out of all wrong disambiguations of baseline2, NEDardv1 had $1,656$ correct. In other words, there seems to be potential for combining them both to achieve better scores. Two examples are NEDardv1 predicting "Diagnosis procedure" and the baseline predicting "Diagnosis" with "Diagnosis procedure" being correct. This is most likely just luck of picking the correct entity with "Diagnosis" in its label. For "Gotham" the baseline suggests the TV Series whereas NEDardv1 correctly suggests the city from Batman. But this is because NEDardv1 did not know the TV Series "Gotham". Therefore it needs to be checked, whether there is a real use in combining both models.

Nocase + candidates In this setting, the model does not care about the casing of the mentions, therefore considering also the "apple" senses if asked for "Apple". Again only candidates are considered. This behaves close to to the case sensitive model. In conclusion with the given pre-trained Sensegram model, the casing of mentions has no significant impact on the NED performance of NEDardv1.

Case + all In this setting the model considers different casing of mentions. Instead of only considering the candidates, all entities are considered.

Evaluation set	Baseline2	NEDardv1	Difference (NEDardv1 - Baseline2)
X_{2k}	0.895	0.288	-0.607
X_{ambig}	0.613	0.265	-0.348
$X_{oneword}$	0.817	0.548	-0.268

These results are interesting as the random guess out of all learned entities known to the NEDard model has a chance of 1/3721342. This is because the model knows embeddings for 3,721,342 different entities. So even though it is selecting one of those 3,721,342 entities, it can get a substantial amount of disambiguations correct. This indicates that the core idea of using multi-sense word embeddings does work for the NED problem.
On the X_{2k} evaluation set the difference is the largest. NEDardv1 performs better on X_{ambig} relative to X_{2k}, again indicating that there is value in multi-sense embedding for NED. The smallest difference is achieved for $X_{oneword}$. It even performs better than the candidate only version.

5.4.6 NEDardv2

Analog to NEDardv1 the evaluation of NEDardv2 is more involved, because of different settings. The following paragraphs deal with each configuration evaluated. As seen in 5.1 none of the NEDardv2 models outperforms baseline2 on any evaluation set. While this is again a sign that NEDardv2 has difficulties with performing NED, each configuration yields interesting results leading to further analysis in 5.5.
Remark: NEDardv2 is the model averaging sense embeddings over the words of a mention and an entity label with *tf-idf* to get a single mention and entity embedding.

case + candidates In this setting, the case of the mention is kept, and only candidates are considered. The model improves over NEDardv1 slightly. This means that the *tf-idf* averaging works better for multi-word mentions than using an embedding per word. This seems to be particularly true for mentions consisting of only one word, which means there is value in weighting words in entity labels during training. Compared to baseline2 NEDardv2 does not improve either.
In direct comparison NEDardv2 had 26,281 disambiguations wrong on X_{2k} (with 100,320 disambiguations) whereas baseline2 had 10492 wrong. Out of those 10,492, NEDardv2 had 2004 correct. This again suggests use in combining both models, which is done later on.

case + all In this setting, the case of the mention is kept, and all entities are considered. This results of NEDardv2 with this setting are different from the one of NEDardv1.

Evaluation set	Baseline2	NEDardv2	Difference (NEDardv2 - Baseline2)
X_{2k}	0.895	0.313	-0.582
X_{ambig}	0.613	0.142	-0.471
$X_{oneword}$	0.817	0.241	-0.576

The scores are aligned with the assumed difficulty of the evaluation set. This means that NEDardv2 also suffers from more ambiguity in the words.

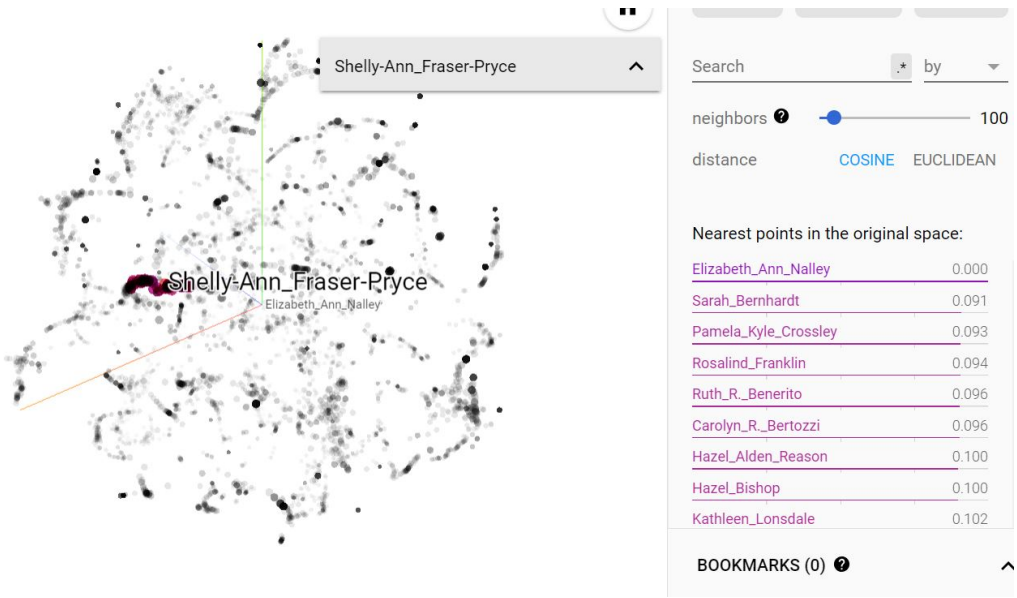

FIGURE 5.3: Cluster of famous women in NEDardv1 model. 3D t-sne representation on the left and cosine distance in original space on the right. Created with [23].

No NER evaluation set As seen in 5.1 not filtering the inter-article links for recognized named entities does not change the performance order of the models.

5.5 Error analysis

This part of the thesis provides deeper insights into why the models behave the way they behave on the evaluation sets.

Visualization A good starting point to analyze embedding models is to visualize them. For this purpose, the first 15000 entity embeddings from the model are dumped and transformed into a 3D representation using t-sne (see 4.1.1). To understand the implications of the visualization the used metric in the original space needs to be considered, too. We set the T-sne parameters learning rate of 10, perplexity of 31 and 315 iterations. For the visualization, the label of each entity is mapped from Wikidata id to Wikipedia article name. In this thesis, the visualization is performed with the tensorflow projector tool [23].

5.5.1 NEDardv1

On first sight, there are many dense clusters in the visualization. 5.3 shows the whole visualization and selects one of the clusters in detail. It turns out that this cluster represents entities for famous women. As NEDardv1 learns a sense for each word in the entity label a "women names" information seems to be derived from the context. Here the first issue with the NEDardv1 model emerges: Multiple entities share embeddings. The selected entity "Shelly-Ann Fraser-Pryce" is not different from "Elizabeth Ann Nalley" as seen by cosine distance of 0. That is because this representation is the "female" sense of "Ann". The same is for "Hazel Bishop" and "Hazel Alden Reason" as the representation of "Hazel" is the same for both entities. This indicates there are too few senses to distinguish between different "Hazel"s, which does not

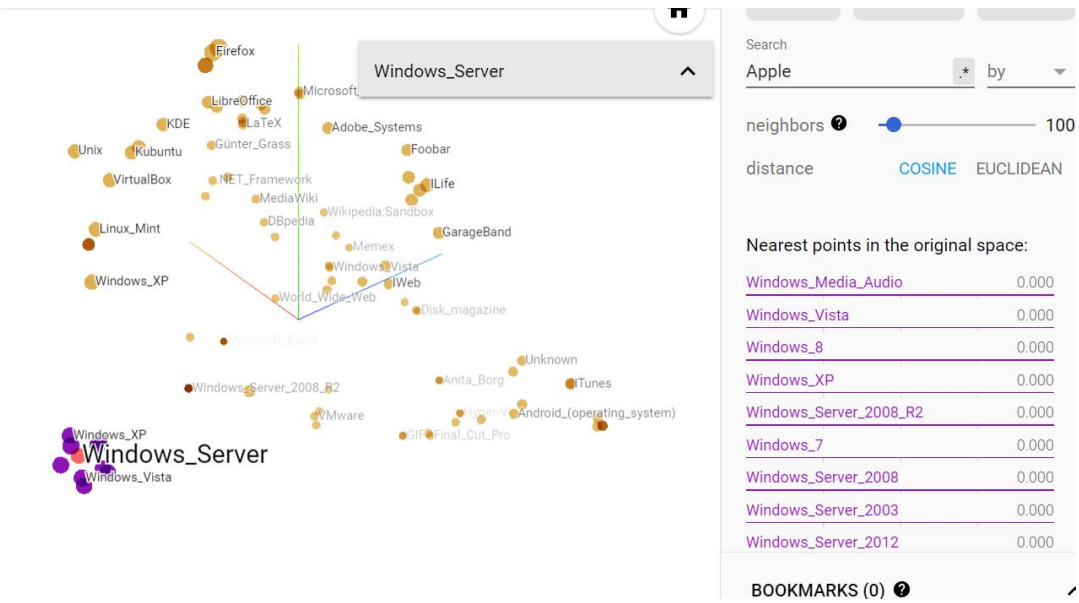

FIGURE 5.4: 3D t-sne visualization of the 100 nearest embeddings to "Apple" in NEDardv1 model on the left. Cosine distance to "Windows_Server" in original space on the right. Created with [23].

disambiguate anything.

To investigate this further the 100 neighbors of the entity "Apple" are visualized in 5.4. The clustering of the senses makes sense as we can see a split into office products, operating systems, and server related software. But looking at the "Windows" cluster, we can see that all the products have the same representation (a cosine distance of 0). This is because they all share the same "computer" sense embedding of "Windows". There is no "Vista" sense of "Windows" and no "XP" sense of "Windows".

Example query For further investigation, the following text is considered:

Apple Inc. is an American multinational technology company headquartered in **Cupertino**, **California**, that designs, develops, and sells consumer electronics, computer software, and online services.

The identified mentions are:

Start	Stop	Text
0	10	Apple Inc.
76	85	Cupertino
87	97	California

Results of NEDardv1 are:

Wikidata Id	*Similarity* $\in [0,1]$	Entity name
Q312	1.000	Apple Inc.
Q189471	0.999	Cupertino California (City)
Q3650742	0.999	California Golden Bears football team

Apple Inc. is split into the words "Apple" and "Inc." for which the matching sense embedding given the context around is calculated.
The context for Apple also includes "Inc" while the context for "Inc" also includes

"Apple". Both are restricted to all known candidates for "Apple Inc.":

$$Q312\#0, Q421253\#0, Q421253\#1$$

namely, Q312 is 'Apple' and Q421253 is 'Apple Store'.
The distances to the 'Apple' mention embedding are [0.00, 0.00, 0.884].
The distances to the 'Inc.' mention embedding are [0.581, 0.581, 0.813].
Remember, that a smaller distance is a greater similarity. Therefore the 'Inc.' sense embedding is further away from our candidates, which is what we would expect as 'Apple' should be more discriminatory. NEDardv1 is going to choose the smallest distance over all "candidate embedding"/"mention word embedding" combinations. This is one of the distances to the word 'Apple' in the mention. And here the issue can be seen: Both the 'Apple' word in the 'Apple' entity and the 'Apple' word in the 'Apple store' entity are the same sense embedding during learning. They have the same distance of 0 to our 'Apple' embedding in the mention. In other words, there is no 'Apple store' sense for word 'Apple' in the MSE model.

In the MSE model 'Apple' has 3 senses with a similar probability for each. The following are these senses and their most similar senses in the MSE model:

'Apple#1' (technology):
`[('Apple\_iPod#1', 0.912), ('PalmPilot#1', 0.910), ('ibooks#1', 0.910)]`

'Apple#2' (fruit):
`[('APPLE#2', 0.995), ('Strawberry#1', 0.971), ('Peach#2', 0.969)]`

'Apple#3' (other):
```
[('apple#3', 0.999), ('APPLE#3', 0.999), ('level#1', 0.983),
('LEVEL#1', 0.983), ('Level#1', 0.983), ('evolution#2', 0.983),
('mouse#2', 0.983), ('Window#3', 0.982)]
```

There is one technology, one fruit and one sense for the rest, which is what we would expect for the word "Apple". But there is no "store" sense of "Apple" indicating "Apple Store".
As one can see the system now chooses one arbitrary entity as the result because the candidates for 'Apple' and 'Apple store' have the same distance. They are in fact the same vector as described. In this case, 'Apple' (Q312) is chosen as it is the first item in the list.

Next up is **'Cupertino'**. NEDardv1 retrieves a sense embedding for 'Cupertino' given its context. Cupertino has the following candidate entities:

$$Q189471\#0, Q110739\#0, Q110739\#1, Q110739\#2, Q52133\#0$$

Namely 'Cupertino' (City), three times 'Santa Clara County', and 'Copertino' (Italy). The distances to the embedding of the mention word are [0.00, 0.4, 0.75, 0.433, 0.776]. Here the disambiguation succeeds as the distance to the desired entity is much smaller than to other candidates. This is reflected by the sense embeddings that are most similar to the given mention embedding. All of them are Californian cities.

```
[('Cupertino#1', 1.0), ('Rocklin#1', 0.981),
('Glendora#1', 0.981), ('Poway#1', 0.979)].
```

So here we have a "Californian City" sense of "Cupertino", which made the disambiguation succeed.

Next up is **'California'**. The mention 'California' has 254 candidates. Same as for 'Apple Inc.' some of the candidates are the same representation of the word 'California' in the candidate entity. This means that they identify 'California' as a country, but are not specific enough to identify the entity. In this case, the first candidate 'Q3650742#0' is the 'California' in 'California Golden Bears football', and it is chosen as the suggestion for the mention.
The word 'California' has three senses in the MSE model. One for Californian cities, one for the state of California and one for the rest (history, western, etc.). Again, there is no 'California Golden Bears football' or 'football team' sense of 'California'. Again, an arbitrary candidate with this sense embedding is chosen because it has the highest similarity (in fact same embedding). Again, multiple entities are represented by the same embedding as there is no specific sense for the entity meaning of their compound words.
Given the new example: "Berlin is a world city of culture, politics, media, and science." we want Berlin to be the city of Berlin and not something else. But the resulting entity is the "Humboldt University of Berlin" and the returned sense vector for the mention is close to "Berlin#1" and "Munich#2". "Frankfurt#1" and "Dresden#2". It can be assumed that this is not the "University" sense of those cities. Therefore, again a sense is missing or cannot be found and an arbitrary entity is chosen. A more narrow MSE model should learn a "University" sense of "Berlin".

Conclusion: In the NEDardv1 model senses are shared across entities. There are too few senses in the MSE model to represent a 1:1 entity sense matching. The senses of the MSE model are too broad. During disambiguation "Windows XP" and "Windows Vista" or not different if the "Windows" component is the same for both and has the most similar matching. This results in an arbitrary entity with this sense of "Windows" to be chosen. Multi-word mentions, therefore, are not resolved correctly by NEDardv1. Same applies for multi-word entities if the mention is "Windows", but a specific Windows is meant as all "Windows" entity representations share the same entity embedding.
For names, just the information that they are names is learned from the context. While this is disambiguating a human entity from other non-human entities with human-like names, it is not narrow enough to identify one person, even the last name.

5.5.2 NEDardv2

Now, we have a look at the NEDardv2 model, which is designed to deal with multi-word mentions differently. For each mention and each entity, a single embedding is computed using the *tf-idf* weighted average of its words sense embeddings.

Visualization The visualized embedding model is shown in 5.5. We can easily see that office products have their cluster, operating systems have their cluster (with the server systems being different from the personal ones), search engines and hardware manufacturers each have their cluster. Also the entities "Microsoft", "Microsoft

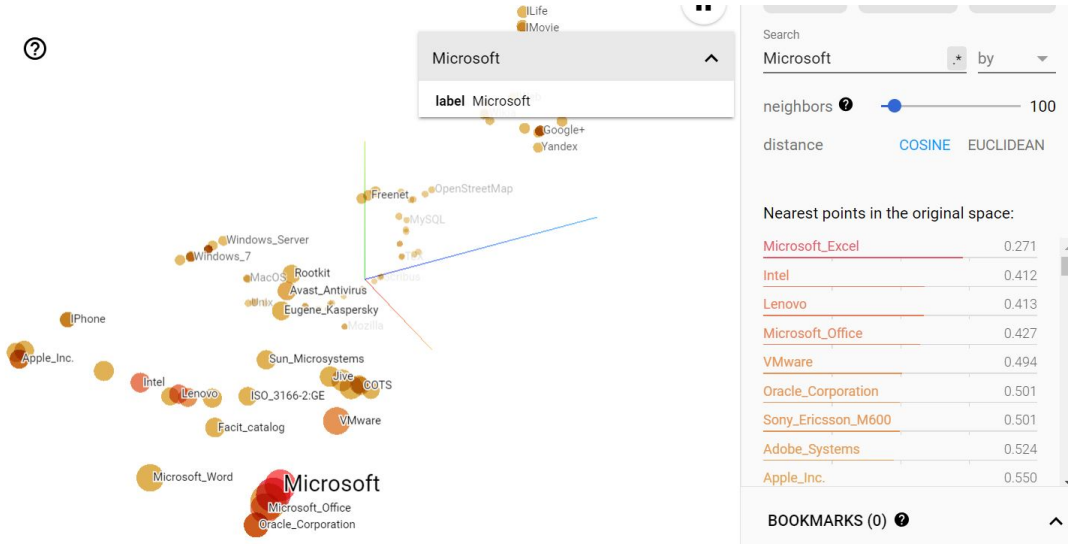

FIGURE 5.5: Cluster of the 100 nearest embedding to "Microsoft" in NEDardv2 model. 3D t-sne representation on the left and cosine distance in original space on the right. Created with [23].

Office" and "Microsoft Excel" are not the same in the original space. This means as expected NEDardv2 does not generate the same embedding for a word shared over mentions like NEDardv1. T-sne visualized the entities as we would expect, but the closest embeddings by cosine distance are fuzzy. Intel and Lenovo are quite similar to "Microsoft", more similar than Microsoft products. This behavior is what makes the results of embedding based approaches fuzzy. Furthermore the low cosine similarity between "Microsoft" and "Microsoft Office" can be explained with the combination of "Office" and "Microsoft" not being close to "Microsoft" anymore.

Example query For further investigation the following text is considered:

> **Apple Inc.** is an American multinational technology company headquartered in **Cupertino**, **California**, that designs, develops, and sells consumer electronics, computer software, and online services.

The identified mentions are:

Start	Stop	Text
0	10	Apple Inc.
76	85	Cupertino
87	97	California

Results are

Wikidata Id	*Similarity* $\in [0,1]$	Entity name
Q312	1.000	Apple Inc.
Q189471	0.999	Cupertino California (City)
Q1134176	0.999	California,_Pennsylvania

Apple Inc. has the candidates `['Q312', 'Q421253']` namely "Apple" and "Apple Store". They only have one entity embedding according to the NEDardv2 model. The distances to our mention "Apple Inc." are `[0.211 0.260]`. The NEDardv2 model correctly identifies "Apple Inc." being closer to the computer representation of "Apple", than to the meaning of "Apple" and "Store".

Cupertino has candidates `['Q189471', 'Q110739', 'Q52133']` namely "Cupertino", "Santa Clara County" and "Copertino". Distances to the mention "Cupertino" are `[0.00 0.500 0.776]`. This means that the model correctly identifies the meaning of "Cupertino" and knows that "Cupertino" is related to Santa Clara County (located in), even though "Cupertino" is not in the entity name. It also knows that "Cupertino" is different from "Copertino".

California has 254 candidates. Instead of predicting Q99 ("California" state), Q1134176 ("California" borough in Pennsylvania) is predicted. Here the real entity has a lower similarity than the wrong one. This is because the context of the mention makes the model resolve to the wrong sense of "California".

Conclusion Combination results of NEDardv2 are fuzzy. Single-word mentions of multi-word entities can fail because partial mentions are not similar to the entity embedding anymore. E.g., the partial "computer"-sense mention "Microsoft" is not anymore near the entity "Microsoft Windows" using cosine similarity, because of "Windows" in the entity label. However, it still can be closer than other candidates. NEDardv2 solves the issue NEDardv1 has with too few senses per word and shared embeddings across entities.

Different types of mentions During analysis, it became clear, that there are multiple types of named entity disambiguation on the Wikipedia corpus.

- The same word with different meanings. E.g. "Apple" as fruit vs. "Apple" as a company. This is not an issue for the proposed model as the different senses can be derived from context.
- The same real-world object in a different context. E.g., "Berlin" can mean the entity for "Berlin" or the entity for "Alt-Berlin", which is the medieval version of Berlin. This is an issue for the proposed model as an extra sense for the medieval entity needs to exist. If the MSE model only knows one "city" representation for Berlin the disambiguation fails.
- Mention with same name and context but a different entity. E.g. "French revolution" can either be the documentary series or the real revolution. The proposed model will fail as it cannot derive a different sense for both.
- Too specific entities. E.g., the mention "Jews" can be linked to the Wikidata entity "Jews during the first and second world war" in the evaluation set. This entity is way too specific for the proposed model to perform disambiguation.
- Words of mention are not featured in entity title. E.g., the mention "Holy see" is linked to "Vatican". Due to the similarity in embedding space, the proposed models should be able to identify the correct entity. However especially NEDardv1 struggles with this task as another candidate with a similar "Holy" or "see" in its title will be suggested first.

Multi word mentions Both proposed models used a different way of dealing with multiple words in mentions and entities. Both models fail in this task. A core issue of using sense embeddings for NED, therefore, is extracting a mention embedding for multi-word mentions.

Noise in knowledge base Another point of failure is too much noise in the candidate list and the Wikidata entities. The candidate list contains entities that are rarely used and most likely not the correct candidate.
The Wikidata knowledge graph contains too specific entities such as "Jews during the first and second world war". Specially NEDardv1 is sensitive to unlikely entities with many words in their labels. In both cases, the models would perform better if such noise is reduced. Either by filtering out less relevant candidates or by filtering the entities learned, limiting the model to a subset of the NED problem.

Too broad senses As seen the proposed models rely on narrow senses for each word to improve NED performance. The semantic resolution can be increased to split senses more often during training of the MSE model. Increasing the semantic resolution might not solve this issue as there are not enough different contexts to distinguish the senses.

Missing words during learning Missing words in the MSE model leads to missing information during entity learning and therefore more ambiguity. Ignoring entities with too many missing words might be a good idea to tackle this issue.

5.6 Potential of combination

As it has been shown both models fail to solve the NED problem on their own, the following combinations with baseline2 are evaluated.

- Truncating the candidate list based on the relevance score, ignoring all candidates for a mention with $relevance \leq x$. This tackles the noise issue introduced by the candidate list.
- Combining NEDard similarity score and relevance score. This is done by learning a linear combination of both scores based on the evaluation set to predict the correct entity. In the end after the linear combination is applied for all candidates again the candidate with the highest score is chosen. For training the evaluation set needs to be split into a training set and a test set to evaluate the generalization performance.

Minimum relevance As minimum candidate relevance 0.08 is set. Only considering candidates with $relevance > 0.08$:

Accuracy	Evaluation set #disambig.			
Model	X_{2k} 100,320	X_{ambig} 20,626	$X_{oneword}$ 37,732	X_{noner} 151,215
Baseline2 (Highest relevance)	**0.895**	0.613	**0.817**	**0.876**
NEDardv1 (case + candidates)	0.719	0.400	0.515	0.720
NEDardv1 (case + candidates > 0.08)	0.889	**0.615**	0.810	-
NEDardv2 (case + candidates)	0.738	0.402	0.567	0.731
NEDardv2 (case + candidates > 0.08)	0.883	0.606	0.797	0.861

The accuracy jumps close to the results of baseline2, but still there is no significant advantage over the baseline2 model.

Linear combination For the combination, logistic regression is used on all candidates of a disambiguation. Each candidate is labeled 0 if it is the wrong and 1 if it is the correct entity for its disambiguation. The task is learning to predict the likelihood for a candidate to be correct given its relevance and similarity score, therefore acting as a simple baseline of combining both models. For evaluation each evaluation set is randomly split into 70% training and 30% test disambiguations.

Evaluation set	Baseline2	NEDardv2	Combined
X_{ambig}	0.616	0.409	0.650
$X_{oneword}$	0.822	0.583	0.839
X_{2k}	0.904	0.753	0.913

It can be seen that the NEDardv2 combination gives slight improvements for $X_{oneword}$ and X_{2k}. It gives a larger improvement on X_{ambig}. This is indicating that the additional information helps to disambiguate more ambiguous mentions.

5.7 Future work

Investigate performance on entity subset Noise from Wikidata entities influences the performance of both NEDard models. In future work, it can be investigated how the performance changes when this noise is removed. In an attempt to remove the noise introduced by Wikidata entities, restrict them during training. A reasonable starting point would be to remove all Wikidata meta entities like categories, concepts or topic groups.
Furthermore, it can be investigated how the performance changes when the NEDard models are learned only on entities with a single-word label or on entities with only (proper) nouns in their label. This has the chance of making the senses match to entities more unambiguously. Another idea is to remove entities that have too few links from other entities inside Wikidata. These approaches grant insights on the performance of NEDard on a subset of Wikidata. However, they do not solve the NED task to disambiguate all entities of Wikidata.

Build an advanced hybrid model Baseline2 and other NED models can be combined with the NEDard models to achieve better performance. For example, choose the model to use based on the mentions length and its candidates. It is possible to choose a threshold for the maximum candidate relevance to chose the disambiguation model. The NEDard models can be used in case the mention is ambiguous by candidate relevance. An advanced hybrid model can be trained by utilizing information about the candidates and mentions as features in addition to the score provided by the models. Furthermore removing unlikely candidates with low relevance should be considered.

Use another method for multi-word entities This thesis proposed two ways of dealing with multi-word mentions and entities. Besides, other techniques can be investigated: For NEDardv1 this means another way to select a candidate over all candidate-word/mention-word combinations. For NEDardv2 this means another way of combining embeddings than averaging with *tf-idf*.

New embedding model The precision of the NEDard models might be increased by altering the multi-sense embedding model. For example:

- Train a new Sensegram model with different parameters.
- Increasing the semantic resolution (n parameter) to narrow down words with too broad sense embeddings. Increasing the semantic resolution makes the MSE models learn more senses for a word. However, it is unclear if there are enough different contexts to learn from.
- Normalize words during training. E.g., remove non-ASCII characters. Normalizing the words makes the model robust to different spellings and character sets and therefore reducing noise. E.g., the German word "Feuerlöscher" can be written as "Feuerloescher", but should be learned as the same word.
- Learn representations for bi-grams (tuples of successive words) too. A lot of entities names consist of more than one word, and a single word embedding is not incorporating this information, even if the two words frequently occur together. Learning representations for such bi-grams might make the model more robust towards multi-word mentions, e.g., "Microsoft Windows".
- Use the AdaGram MSE model instead. Embeddings might behave differently due to their different training process.

5.8 Conclusion

This thesis evaluated the use of multi-sense word embeddings on the NED task. Insights are gained on the pitfalls of using multi-sense embeddings for this task. It has been shown that there is no 1:1 mapping for entities. There are way fewer senses than entity candidates for a word. Senses are too broad, especially names. Mentions and entities with multiple words need a well thought out strategy. Parts of the NED problem are not solvable as the entities are too specific versions of an entity. Both Wikidata entities and the Wikipedia based candidate list introduce noise, and it needs to be cleaned from irrelevant entries. The two proposed models do not advance on the NED task on their own. They might improve a specific part of the task and therefore are useful in combination with other techniques. It is advised to use them in combination with other NED models to tackle specific parts of the problem such as entities that are distinct real-world entities with the same name. E.g. "Apple" as fruit and company. The best accuracy obtained on an evaluation set (N_{2k}) is 0.738, having 74,039 out of 100,320 disambiguations correct. Combining the score obtained by the proposed models with the baseline increased the accuracy on more ambiguous evaluation sets. The best combination accuracy on the test set of a linear combination is 0.650 in comparison to 0.616 of the baseline alone.

Bibliography

[1] URL: https://en.wikipedia.org/.

[2] URL: https://wikidata.org/.

[3] URL: https://www.wikidata.org/wiki/Help:Items.

[4] URL: https://www.mediawiki.org/wiki/Help:Formatting/.

[5] URL: https://spacy.io/.

[6] URL: https://dumps.wikimedia.org/.

[7] URL: https://www.mediawiki.org/wiki/Alternative_parsers.

[8] URL: https://radimrehurek.com/gensim/scripts/segment_wiki.html.

[9] URL: https://github.com/earwig/mwparserfromhell/.

[10] URL: https://github.com/attardi/wikiextractor.

[11] URL: http://qlever.informatik.uni-freiburg.de/.

[12] URL: https://www.w3.org/TR/rdf-sparql-query/.

[13] URL: https://stackoverflow.com/questions/38287772/cbow-v-s-skip-gram-why-invert-context-and-target-words.

[14] URL: https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/.

[15] URL: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/.

[16] URL: http://ruder.io/word-embeddings-1/.

[17] URL: https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.

[18] URL: https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/.

[19] URL: https://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html.

[20] URL: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[21] URL: https://www.slideshare.net/ChristopherMoody3/word2vec-lda-and-introducing-a-new-hybrid-algorithm-lda2vec-57135994.

[22] URL: https://radimrehurek.com/gensim/models/keyedvectors.html.

[23] URL: https://projector.tensorflow.org.

[24] Yamada et. al. "Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation". In: (2016). URL: https://arxiv.org/pdf/1601.01343.pdf.

[25] Sergey Bartunov. "Breaking Sticks and Ambiguities with Adaptive Skip-gram". In: (2015). URL: https://arxiv.org/abs/1502.07257.

[26] Niklas Baumert. "Web-scalable Named-entity Recognition and Linking with a Wikipedia-backed Knowledge Base". In: (2018). URL: http://ad-publications.informatik.uni-freiburg.de/theses/Bachelor_Niklas_Baumert_2018.pdf.

[27] Christian Biemann. "Chinese Whispers - An Efficient Graph Clustering Algorithm And Its Application To Natural Language Processing Problems". In: 2006.

[28] *How to Use t-SNE Effectively*. URL: https://distill.pub/2016/misread-tsne/.

[29] Olivier Raiman Jonathan Raiman. "DeepType: Multilingual Entity Linking by Neural Type System Evolution". In: (2018). URL: https://blog.openai.com/discovering-types-for-entity-disambiguation/.

[30] Thomas Mikolov. "Efficient Estimation of Word Representations in Vector Space". In: (2013). URL: https://arxiv.org/abs/1301.3781.

[31] Ragavan Natarajan. "Entity Disambiguation using Freebase and Wikipedia". In: (2014). URL: http://ad-publications.informatik.uni-freiburg.de/theses/Master_Ragavan_Natarajan_2014.pdf.

[32] Andrés García-Silva Christian Bizer Pablo N. Mendes Max Jakob. "DBpedia Spotlight: Shedding Light on the Web of Documents". In: (2011). URL: https://www.dbpedia-spotlight.org/docs/spotlight.pdf.

[33] Maria Pelevina. "Making Sense of Word Embeddings". In: (2016). URL: http://aclweb.org/anthology/W/W16/W16-1620.pdf.

[34] Fei Tian. "A Probabilistic Model for Learning Multi-Prototype Word Embeddings". In: (2014). URL: http://www.aclweb.org/anthology/C14-1016.

[35] Andrew Trask. "sense2vec - A Fast and Accurate Method for Word Sense Disambiguation In Neural Word Embeddings". In: (2015). URL: https://arxiv.org/abs/1511.06388.

[36] *Wikipedia2vec*. URL: https://wikipedia2vec.github.io/wikipedia2vec/.

[37] *word2vec Parameter Learning Explained*. URL: https://arxiv.org/pdf/1411.2738.pdf.