Word Sense Determination from Wikipedia Data Using Neural Networks

Qiao Liu

A Project Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment of the Requirements

Of the Degree

Master of Science

By

Qiao Liu

December 2017

The Designated Project Committee Approves the Master's Project Titled

Word Sense Determination from Wikipedia Data Using Neural Networks

by

Qiao Liu

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2017

ACKNOWLEDGEMENT

I want to express my sincere appreciation to my project advisor, Dr. Christopher Pollett, for his enduring patience and guidance throughout this project. I also want to extend my thanks to the committee members, Dr. Jon Pearce and Dr. Suneuy Kim, for their suggestions and time.

Abstract

Word Sense Determination from Wikipedia Data Using Neural Networks

by Qiao Liu

Many words have multiple meanings. For example, “plant” can mean a type of living organism or a factory. Being able to determine the sense of such words is very useful in natural language processing tasks, such as speech synthesis, question answering, and machine translation. For the project described in this report, we used a modular model to classify the sense of words to be disambiguated. This model consisted of two parts: The first part was a neural-network-based language model to compute continuous vector representations of words from data sets created from Wikipedia pages. The second part classified the meaning of the given word without explicitly knowing what the meaning is. In this unsupervised word sense determination task, we did not need human-tagged training data or a dictionary of senses for each word. We tested the model with 5 randomly selected ambiguous words from 10 words that formed the test set in the related work by Schütze in 1998, and compared our experimental results with Schütze’s. The comparison showed that our model got better accuracy for 2 words, similar accuracy for 1 word, and poorer accuracy for 2 words.

Table of Contents

Figures

Tables

1. Introduction

Word sense disambiguation is the task of identifying which sense of an ambiguous word is used in a sentence. There are two variants of the word sense disambiguation task: lexical sample task and all-words task. In lexical sample task, we need to disambiguate a small set of pre-selected words. In all-words task, we need to disambiguate every word shown in the text. Word sense disambiguation task can be divided into two subtasks: sense discrimination task and sense labeling task. Sense discrimination task groups/clusters the occurrences of an ambiguous word with the same meaning. Sense labeling task assigns the sense labels to the occurrences of an ambiguous word. This project focus on the sense discrimination component of lexical sample task. The purpose of the project is “given a word to be disambiguated, to group occurrences of the word into clusters, without explicitly knowing the meaning of the occurrences”.

There are three fundamental approaches of word sense disambiguation: dictionary-based, supervised machine learning, and unsupervised learning. In this project, we used unsupervised learning to classify the senses of the target word. In the unsupervised learning approach introduced by Schütze’s in 1992 and 1998 [14, 21], he interpreted the sense of the ambiguous word as clusters of similar contexts. We used the same interpretation in our project. Unlike Schütze’s approach using co-occurrence counts to generate vectors, we used word embeddings trained by a neural statistical language model to generate word vectors. Schütze used a data set taken from the New York Times News Service. We used data sets generated from Wikipedia pages.

We experimented and evaluated the disambiguation tasks on some naturally ambiguous words that were used by Schütze (1998) [21]. We classified occurrences of these words to two/three frequent senses and compared the experimental results with Schütze’s test results [21]. For some words, our model got better accuracy. However, for some words, our model failed to classify the senses. We analyzed the result in discussions section.

In this report, Chapter 2 presents a review of existing work and describes the background and technique we used in our project. Chapter 3 describes the model architecture. Chapter 4 introduces the data sets and data preprocessing. Chapter 4 describes the key information about implementation. Chapter 6 presents the experimental results and our discussions. Chapter 7 presents the conclusion and possible future work.

2. Background

This chapter reviews existing work of word sense disambiguation and introduces the word embeddings and neural-network-based statistic language models, which are needed to understand the project.

2.1 Review of Existing Work

Three fundamental approaches of word sense disambiguation mentioned in introduction are described in detail as below:

1. Dictionary-based: Michael Lesk initially implemented this approach in 1986 [11]. Given a target word t to be disambiguated in context c. First, they retrieved all the sense definitions for t from a dictionary. Then, they selected the sense s whose definition had the most overlap with c of t. There are variant ways to measure the overlap. One example is using IDF weighted vectors. The dictionary-based approach requires a hand-built machine readable semantic sense dictionary.
2. Supervised machine learning: A set of features has been extracted from the context of the target word. Different types of features can be extracted, such as collocational features and bag-of-words features. The features were used to train classifiers that can label ambiguous words in new text. Different classifiers have been used, such as naïve Bayes, logistic regression, k-nearest neighbors, etc. Same as dictionary-based approaches, these approaches require costly large hand-built resources, because we need to label each ambiguous word in the training data. A semi-supervised approach was proposed in [20]. In this approach, they did not rely on a large hand-built data, due to using bootstrapping to generate dictionary from a very small hand-labeled seed-set [20].

3. Unsupervised machine learning: Schütze used a clustering method for word sense discrimination in [14, 21]. He interpreted the sense of the ambiguous word as clusters of similar contexts. Words were represented by high-dimensional, real-valued vectors derived from co-occurrence counts. As shown in Figure 1, the vector of word “judge” was derived from the co-occurrence counts of its neighbor words “legal” and “clothes”. Schütze used the most frequent 1,000 and 2,000 words to generate vectors, which meant the word vector dimension was 1,000 and 2,000 respectively. The centroid (or sum) of the word vectors in the context represented the context vector. The set of context vectors collected from the corpus was clustered, and then the centroid of the context vectors in the cluster was used to represent the sense of that cluster. To discriminate the sense, he measured and ranked the cosine similarity between the context vector and sense vector.

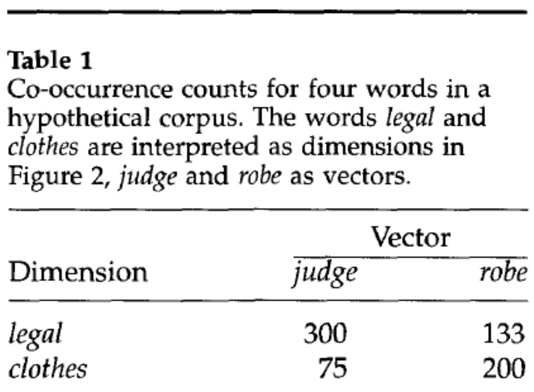


Table 1
Co-occurrence counts for four words in a hypothetical corpus. The words *legal* and *clothes* are interpreted as dimensions in Figure 2, *judge* and *robe* as vectors.

	Vector	
Dimension	*judge*	*robe*
legal	300	133
clothes	75	200

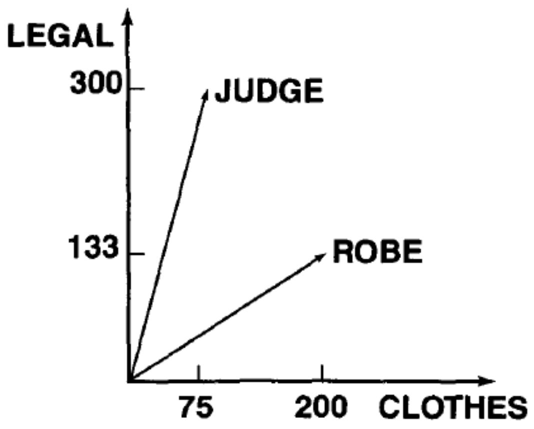


Figure 2
The derivation of word vectors. *judge* and *robe* are represented as word vectors in a two-dimensional space with the dimensions 'legal' and 'clothes.' Co-occurrence data are from Table 1.

Figure 1: Word vector in Schütze's approach

In our project, we used a modification of the unsupervised machine learning approach. Word embeddings are trained by a statistical language model based on neural networks using data from Wikipedia pages. More details are introduced in the following sections.

2.2 Word Embeddings

To represent word and contexts in vector space, we used word embeddings. Word embeddings are sometimes called word representations or word vectors. A word embedding maps each word to a dense vector W of real numbers [12]. To compute word embeddings, vectors of words are initialized by generating vectors with random real numbers. A program learns meaningful vectors by perform some tasks.

$$\text{word} \rightarrow R^n$$

$$W(\text{“plant”}) = [0.3, -0.2, 0.7, \ldots]$$

$$W(\text{“crane”}) = [0.5, 0.4\ -0.6, \ldots]$$

To see word embeddings' intuitive sense, we can visualize the representation of words in a two-dimension projection. For example, looking at Figure 2, digits are close together, and, similar words are close together [12]. It turns out, though, that much more sophisticated relationships are also encoded in this way [12]. We used word embeddings in our disambiguation task.

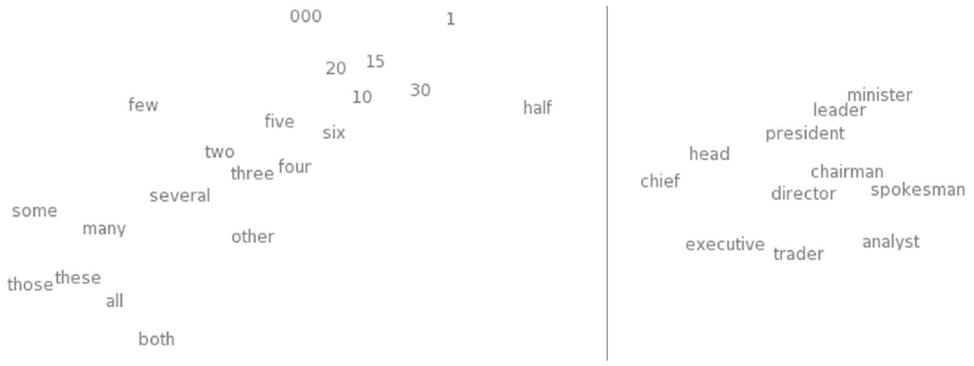


Figure 2: Two-dimension projection [12]

2.3 Statistical Language Models Based on Neural Networks

To train the embeddings, we used a statistical language model based on neural networks. This section introduces some backgrounds and histories of the neural network based statistical language models.

Many natural language processing systems represent each word in a one-hot vector, which with value 1 at the position of the word and value 0 at all other positions. The system learns the joint probability function of sequences of words in a language [1]. The one-hot vector technique faces the problem of the curse of dimensionality [1]. For example, to model the joint distribution of a sequence of *m* words in a language with a vocabulary of size $|V|$, the size of the vector is potentially $|V|^m - 1$.

Probably the most successful concept to fight the curse of dimension is to use distributed representation of words [3]. In this representation, each word is represented by a word vector of *d* real-valued parameters. *d* is usually equals to some hundreds, which is much smaller than the size of the one-hot vector.

Rumelhart, Hinton, and Williams proposed a new learning procedure for networks of neuron-like units in 1986 [3, 5]. This idea has been applied to statistical language modelling by Bengio, Ducharme, and Vincent in 2003 [1]. They proposed a very popular model architecture for estimating a neural network language model (NNLM), where the model simultaneously learns a distributed representation for each word along with the probability function for word sequences, expressed in terms of these representations [1]. Many works showed neural network based language models significantly outperform n-gram models [1, 6, 7], which use one-hot vector. Another architecture of NNLM was proposed in [2]. This model used neural networks with a single hidden layer to learn word embeddings. Several extensions that improve both the quality of the vectors and the training speed were introduced in [8].

We employed the Skip-gram model introduced in [8] in our project. It is a simply and efficient method with only one hidden layer for learning high-quality vector representations of words from large amounts of unstructured text data [3]. More detail about the model is introduced in the model architecture section.

3. Model Architecture

3.1 A Modular Model to Determine Word Sense

Many natural language processing tasks take the approach of first learning a good word representation on a task and then using that representation for other tasks. We used this approach for the word sense determination task. The model architecture is shown in Figure 3. We trained a Neutral Network Language Model to get the word embeddings. Even though a single word might carry multiple meanings, one representation per word was learned. We cannot determine the word's sense by using the word's embedding itself. The distributional hypothesis indicates that similar words appear in similar contexts [9, 10]. By exploiting this hypothesis, we used a classifier on top of the word embeddings to cluster the context to determine the meaning of the target word.

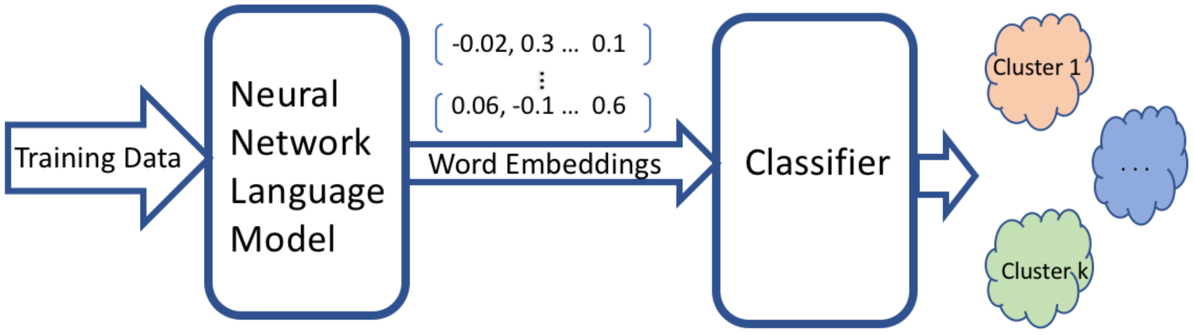


Figure 3: A modular model

3.2 Skip-gram Model

The continuous Skip-gram model is simple and efficient method for learning high-quality vector representations of words from large amounts of unstructured text data [3]. The structure of the model is showed in Figure 4 [3]. The input layer takes in a target word, the projection layer projects the probability that each word appears as the target word's context. The output layer outputs the selected context words.

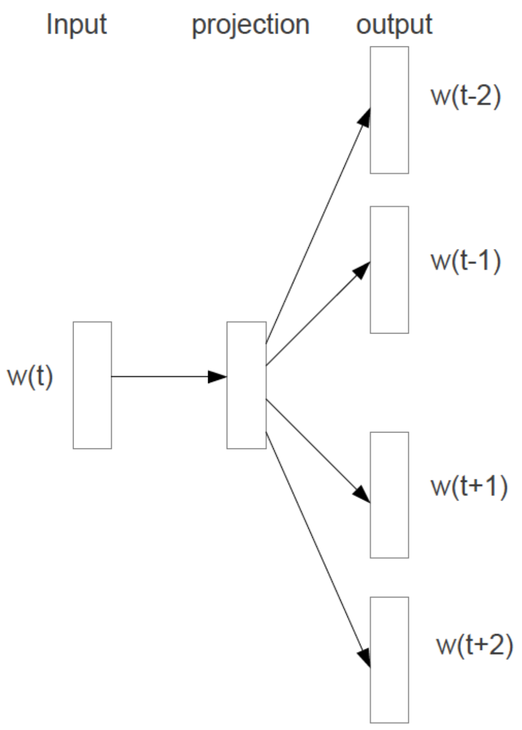


Figure 4: The skip-gram model architecture

The training objective was to learn word embeddings good at predicting the context words in a sentence. We trained the neural network by feeding it word pairs of target word and context word found in our training dataset. Rather than use all context words, we only scan through context words appear in a predefined window size. Given the sentence "natural language processing projects are fun." and the window size of 2, we get the word pairs as in Table 1.

Sentence	Training samples
natural **language processing** projects are fun	(natural, language) (natural, processing)
natural **language** **processing projects** are fun	(language, natural) (language, processing) (language, projects)
natural language **processing** **projects** are fun	(processing, natural)

	(processing, language) (processing, projects)
natural **language processing projects are fun**	(projects, language) (projects, processing) (projects, are) (projects, fun)
natural language **processing projects are fun**	(are, processing) (are, project) (are, fun)
natural language processing **projects are fun**	(fun, projects) (fun, are)

Table 1: Word pair examples generated from a sentence

More details about the model are shown in Figure 5. In this example, the vocabulary size is $|V|$. The word representation size is d. We represent the word "plant" as a one-hot vector (only the position of the word in the vocabulary is 1, otherwise, it is 0) in input layer. The hidden layer represents a matrix of size d x $|V|$. Each column in hidden layer represents the vector of one word. The output layer size is $|V|$. Each value in the output layer represents the probability that the word appears as context word of "plant". To predict the context word, first, the model looks the vector of the target word up in the hidden layer, and then calculates the similarity between the target word and each word in the vocabulary. The similarity is measured by using dot product of vectors of the target word and the context word.

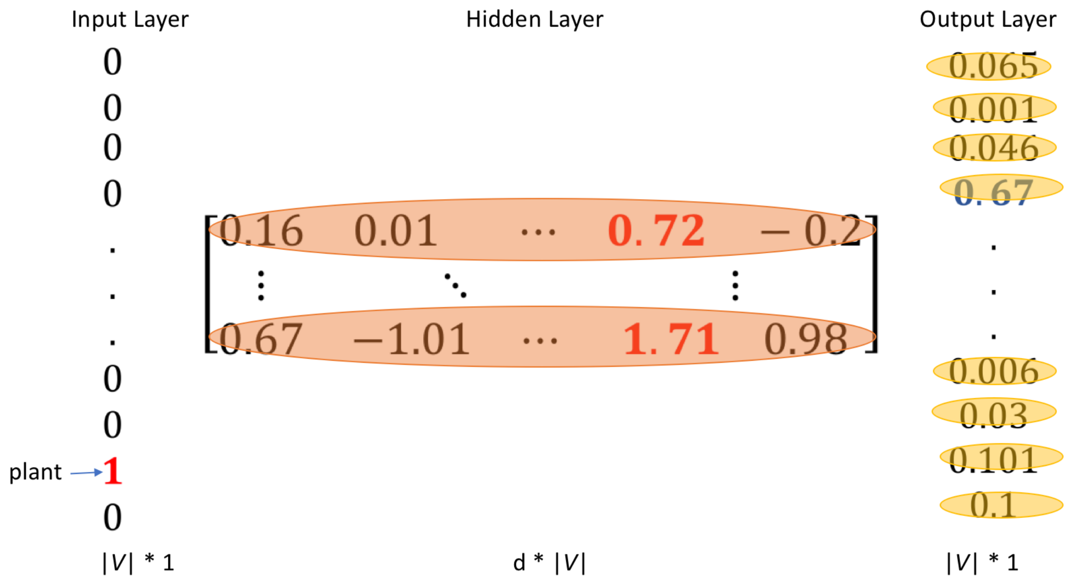


Figure 5: The skip-gram model

The objective was to maximize the probability of any context word given the target word:

$$J'(\theta) = \prod_{t=1}^{V} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} p(w_{t+j} | w_t; \theta)$$

w_t is the target word, θ is the word embeddings matrix of size $|V|$ x d, and m is the window size. If $m = 2$, as in Figure 4, we take two words before the target word and two words after the target word. $|V|$ is the vocabulary size. By taking the negative log, we can transform the problem to one where we minimize the loss $J(\theta)$

$$J(\theta) = -\frac{1}{V} \sum_{t=1}^{V} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log\left(p(w_{t+j} | w_t; \theta)\right)$$

The probability was defined as a sigmoid function:

$$p(w_o | w_t) = \frac{\exp\left({w_o}^T w_t\right)}{\sum_{j=1}^{V} \exp\left({w_j}^T w_t\right)}$$

The objective was further transformed to maximize the log probability or minimize the negative log probability.

Some models use different vector representations for each word when it is target word or context word. In the definition above and our implementation, each word has the same vector representation no matter it is target word or context word, which is widely used to improve training efficiency.

The large size of our word vocabulary (20K/50K) means that our skip-gram neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples [18]. To improve the train efficiency, hierarchical softmax or negative sampling (NEG) [2] is usually used. Negative sampling is similar as Noise Contrastive Estimation (NCE) [16]. The idea of NCE is to use logistic regression to differentiate data from noise. NEG further simplifies NCE. To use NEG in the Skip-gram model, we need to generate noise by randomly selecting words to create negative word pair samples. The experiment results in [3] show that the 2-5 negative samples are sufficient to separate signal form noise for large datasets. We set the negative sample size to 3 in our experiment.

3.3 k-means Clustering

k-means is a simple unsupervised classification algorithm. The aim of the k-means algorithm is to divide *m* points in *n* dimensions into *k* clusters so that the within-cluster sum of squares is minimize [13]. The word vector represents the position of the word in a high dimensional space. The distributional hypothesis says that similar words appear in similar contexts [9, 10]. Thus, we can use k-means to divide all vectors of context into *k* clusters. There could be many ways to create the vectors from the context. One way is getting the mean of the context, which is used in our implementation. The restriction of using k-means clustering in our project is that we need to know the number of senses of the word before perform clustering on it.

4. Data Sets and Data Preprocessing

With the model described in the Model Architecture chapter, we need to extract word pairs from a natural language to train the model. Our training data is based on a Wikipedia data dump. In our experiments, we downloaded the data from *https://dumps.wikimedia.org/enwiki/20170201/*. The `pages-articles.xml` of Wikipedia data dump contains current version of all article pages, templates, and other pages. The steps to generate training data (word pairs) from Wikipedia data included sentence extraction, dictionary and reversed dictionary creation, sentence regeneration, and word pairs creation.

4.1 Extract Sentences

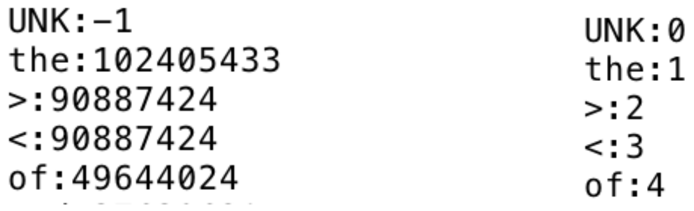
First, we extracted all sentences from article pages and cleaned the sentences. To indicate the start and end of sentences, we used *"<"* and *">"* respectively, and we also treated them as words in the following steps. We only kept the sentences of length >= 9, since we used 10 context words for each target word. Finally, we extracted 90,887,424 sentences in total.

```
< a line of light emanating from a lighthouse makes one revolution every 10 seconds >
< the lighthouse is located 4km off a straight shoreline >
< how fast does the light move along the shoreline when it forms a 45 degree angle to
a line from the lighthouse perpendicular to the shoreline >
```

Figure 6: Extracted sentence examples

4.2 Create Dictionary and Reversed Dictionary

In the input layer, we used a one-hot vector to represent the word as show in Figure 5. In the implementation, we simply used index of the *1* in the one-hot vector to represent the word. To create a dictionary mapping each word to its index, we counted word frequency, and sorted words by their frequencies. Then, the word's frequency rank in the descending order was used as the index of the word. We constrained the size of the dictionary, thus many rarely shown words were not included in the dictionary but were represented as "UNK" with index 0. Figure 7 shows the top 5 frequent words' frequency and their indices. The "<" and ">" represent the start and end of a sentence respectively.



```
UNK:-1                  UNK:0
the:102405433           the:1
>:90887424              >:2
<:90887424              <:3
of:49644024             of:4
```

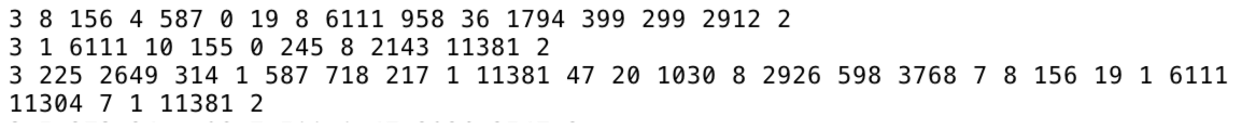
Figure 7: Word frequency and index examples

A reversed dictionary, which maps the index to the word, was created simultaneously. It was used to construct natural language sentences from lists of indices in the experiment.

With different vocabulary size, we will get different dictionary. In our implementation, we experimented two vocabulary sizes of 20k and 50k.

4.3 Regenerate Sentences

Using the dictionary, we translated each text sentence to a sequence of indices. The examples shown in Figure 6 was translated to sentences show in Figure 8 with vocabulary size of 20k.



```
3 8 156 4 587 0 19 8 6111 958 36 1794 399 299 2912 2
3 1 6111 10 155 0 245 8 2143 11381 2
3 225 2649 314 1 587 718 217 1 11381 47 20 1030 8 2926 598 3768 7 8 156 19 1 6111
11304 7 1 11381 2
```

Figure 8: Sentences with words representing in indices

4.4 Create Word Pairs

To generate word pairs, we need to set the window size of the context and the number of context words to be used. For example, when the window size is 3, and the number of context words is 4, we need to randomly select 4 context words out of 6 (3 x 2) neighbors to generate 4 pairs for each target words. In our implementation, we set the window size to 5 and the number of context words to 10. We processed 545,107,494 target words, which means around 5 billion word pairs have been generated.

5. Implementation

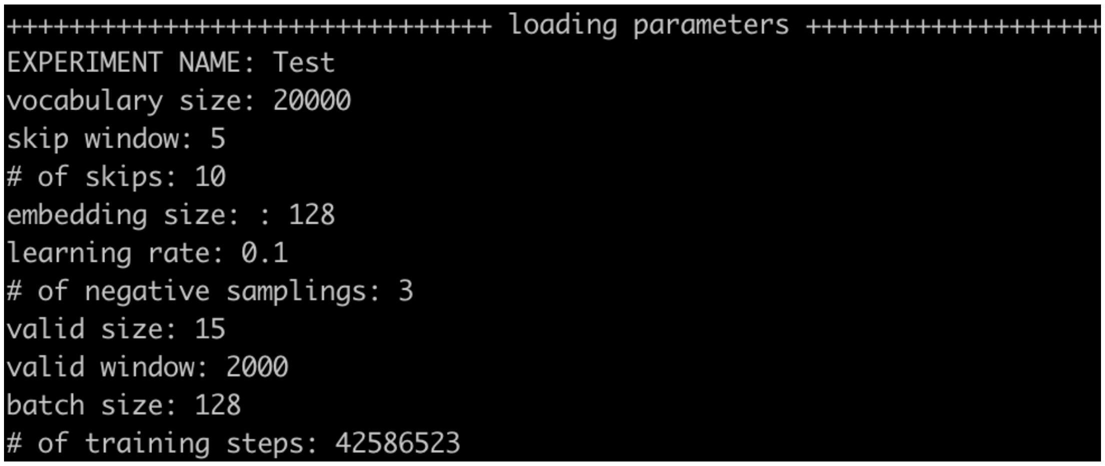
We used TensorFlow as the framework to build the Skip-gram model to train word embeddings. The Skip-gram model has many parameters. In the following section, we summarize the meaning of the parameters and introduce the optimizer used in our implementation. After the Skip-gram model was implemented to train the word embeddings, the python science package sklearn [15] was used to perform k-means clustering on the top of the embeddings.

5.1 Parameters of Skip-gram Model

Table 2 shows the parameters and meanings for the Skip-gram model. We experimented with different parameters. We show the results in the Experiments chapter. One example setting to the parameters is shown in Figure 9.

Parameters	**Meaning**
VOC_SIZE	The vocabulary size with all rare words represented as ‘UNK’ in vocabulary.
SKIP_WINDOW	The window size of text words around target word.
NUM_SKIPS	The number of context words. Those number of context words in the window will be randomly took to generate word pairs.
EMBEDDING_SIZE	The number of parameters in the word embedding. The size of the word vector.
LR	The learning rate of gradient descent.
BATCH_SIZE	The size of each batch in stochastic gradient descent. Running one batch takes one step.
NUM_STEPS	The number of training steps.
NUM_SAMPLE	The number of negative samples. It can be as small as 2-5 for large datasets [3].

Table 2: Parameters for the skip-gram model



```
++++++++++++++++++++++++++++++ loading parameters ++++++++++++++++++
EXPERIMENT NAME: Test
vocabulary size: 20000
skip window: 5
# of skips: 10
embedding size: : 128
learning rate: 0.1
# of negative samplings: 3
valid size: 15
valid window: 2000
batch size: 128
# of training steps: 42586523
```

Figure 9: Example of parameters

Since we are using negative sampling, the noise distribution is a parameter for the model as well. The *tf.nn.nce_loss* uses a log-uniform (Zipfian) distribution for sampling by default, so our labels must be sorted in the descending frequency order to achieve good results [18].

5.2 Optimizer of Skip-gram Model

Gradient descent finds the minimum of a function by taking steps proportional to the positive of the gradient. In each iteration of gradient descent, we need to calculate all examples. Instead of computing the gradient of the whole training set, each iteration of stochastic gradient descent only estimates this gradient based on a batch of randomly picked examples [17]. We used stochastic gradient descent to optimize the vector representation during training.

5.3 Tools and Packages

The amount of data we processed is too large to load into memory at one time. Throughout the implementation, the data was read line by line to reduce the memory requirements. The tools and packages we used are listed below:

TensorFlow r1.4

TensorBoard 0.1.6

Python 2.7.10

Wikipedia Extractor v2.55

sklearn.cluster [15]

numpy

6. Experiments

We experimented the Skip-gram model with different parameters and selected one word embedding for clustering.

The experimental results were compared with Schütze's unsupervised learning approach in 1998 [21]. He used a data set (435M) taken from the New York Times News Service. We used the data set extracted from Wikipedia pages (12G). In his approach, co-occurrence counts were used to generate vectors, which had large numbers of vector dimension (1,000/2,000), therefore, he applied singular-value decomposition to the vectors. We used the Skip-gram model to learn a distributed word representation with a dimension of 250. Taking advantage of a smaller number of dimension, we did not need to perform any matrix decomposition.

6.1 Experiment with Skip-gram Model

To decide which learning rate to use, we experimented with a group of tests with 3 different learning rates (LR) as shown in Table 3. We found the model converged to similar average losses after around 20M steps. We selected the learning rate as 0.3 for future experiments.

Parameters	VOC_SIZE	SKIP_WINDOW	NUM_SKIPS	EMBEDDING_SIZE
Value	20K	5	10	128
Parameters	LR	BATCH_SIZE	NUM_STEPS	NUM_SAMPLE
Value	0.03/0.3/1.0	128	Set to 1 epoch	3

Table 3: Parameters of test group 1

To decide the number of the training epochs, we experimented with 2 tests with different NUM_STEPS values as shown in Table 4. We observed lower loss using 2 epochs. By referring to the experiment in [2], we decided to train the model with 6 epochs.

Parameters	VOC_SIZE	SKIP_WINDOW	NUM_SKIPS	EMBEDDING_SIZE
Value	20K	5	10	128
Parameters	LR	BATCH_SIZE	NUM_STEPS	NUM_SAMPLE

Value	0.3	128	Set to 1 epoch / 2 epochs	3

Table 4: Parameters of test group 2

To select the vocabulary size, we experimented with 2 tests with parameters as shown in Table 5. Increasing the vocabulary size from 20K to 50K helped to decrease the loss slightly.

Parameters	VOC_SIZE	SKIP_WINDOW	NUM_SKIPS	EMBEDDING_SIZE
Value	20K/50K	5	10	128/250
Parameters	LR	BATCH_SIZE	NUM_STEPS	NUM_SAMPLE
Value	0.3	128	Set to run dataset 1 time / 2 times	3

Table 5: Parameters of test group 3

To monitor the loss of the model during training, we used average loss to estimate the loss over every 100K batches. The loss had shown the same pattern during training with different parameters. The loss of the training in first group with learning rate 0.3 is shown in Figure 10. While the loss keep decreasing, the model found words closer to the target word. The nearest words of sample words at training step 10M is shown in Figure 11, and the result at training step 40M is shown in Figure 12. Even from intuition, we could observe better performance at step 40M.

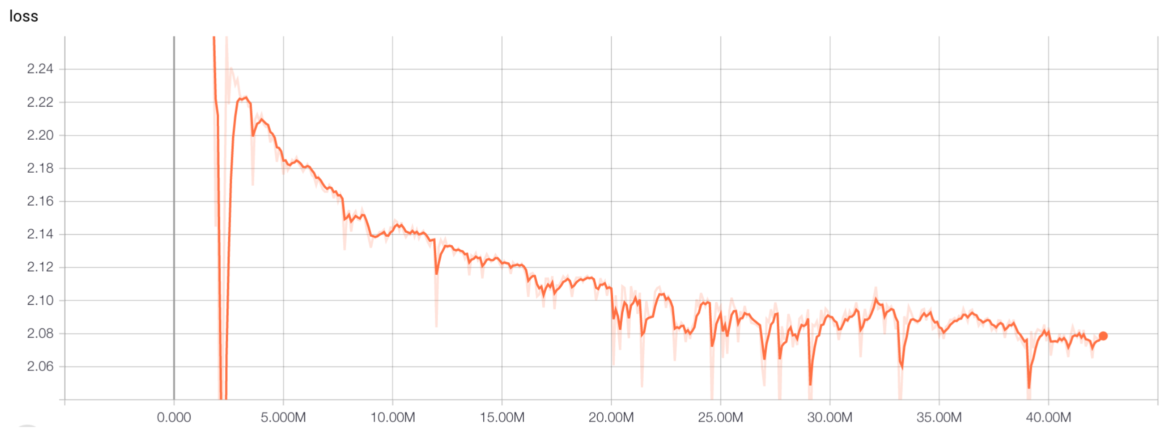
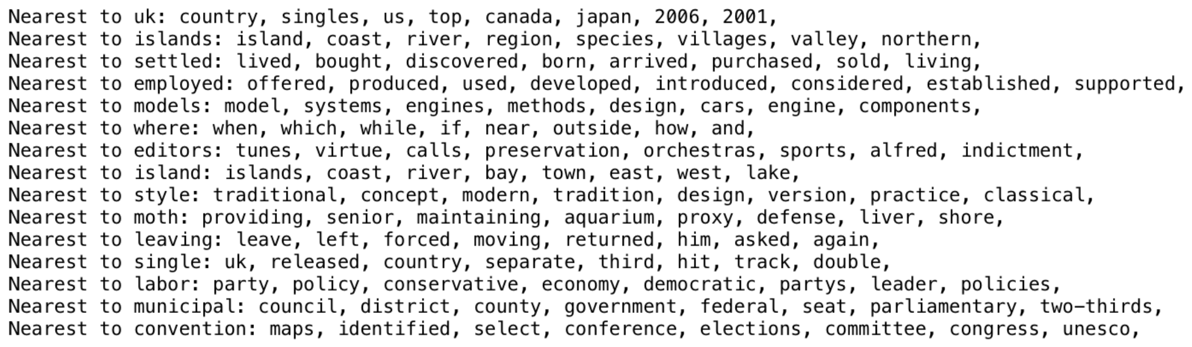
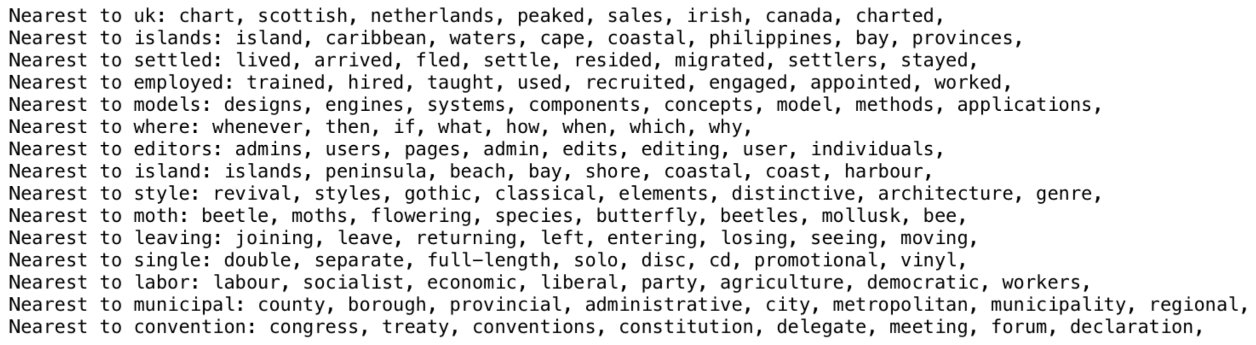


Figure 10: The average loss during training



```
Nearest to uk: country, singles, us, top, canada, japan, 2006, 2001,
Nearest to islands: island, coast, river, region, species, villages, valley, northern,
Nearest to settled: lived, bought, discovered, born, arrived, purchased, sold, living,
Nearest to employed: offered, produced, used, developed, introduced, considered, established, supported,
Nearest to models: model, systems, engines, methods, design, cars, engine, components,
Nearest to where: when, which, while, if, near, outside, how, and,
Nearest to editors: tunes, virtue, calls, preservation, orchestras, sports, alfred, indictment,
Nearest to island: islands, coast, river, bay, town, east, west, lake,
Nearest to style: traditional, concept, modern, tradition, design, version, practice, classical,
Nearest to moth: providing, senior, maintaining, aquarium, proxy, defense, liver, shore,
Nearest to leaving: leave, left, forced, moving, returned, him, asked, again,
Nearest to single: uk, released, country, separate, third, hit, track, double,
Nearest to labor: party, policy, conservative, economy, democratic, partys, leader, policies,
Nearest to municipal: council, district, county, government, federal, seat, parliamentary, two-thirds,
Nearest to convention: maps, identified, select, conference, elections, committee, congress, unesco,
```

Figure 11: Nearest words 1



```
Nearest to uk: chart, scottish, netherlands, peaked, sales, irish, canada, charted,
Nearest to islands: island, caribbean, waters, cape, coastal, philippines, bay, provinces,
Nearest to settled: lived, arrived, fled, settle, resided, migrated, settlers, stayed,
Nearest to employed: trained, hired, taught, used, recruited, engaged, appointed, worked,
Nearest to models: designs, engines, systems, components, concepts, model, methods, applications,
Nearest to where: whenever, then, if, what, how, when, which, why,
Nearest to editors: admins, users, pages, admin, edits, editing, user, individuals,
Nearest to island: islands, peninsula, beach, bay, shore, coastal, coast, harbour,
Nearest to style: revival, styles, gothic, classical, elements, distinctive, architecture, genre,
Nearest to moth: beetle, moths, flowering, species, butterfly, beetles, mollusk, bee,
Nearest to leaving: joining, leave, returning, left, entering, losing, seeing, moving,
Nearest to single: double, separate, full-length, solo, disc, cd, promotional, vinyl,
Nearest to labor: labour, socialist, economic, liberal, party, agriculture, democratic, workers,
Nearest to municipal: county, borough, provincial, administrative, city, metropolitan, municipality, regional,
Nearest to convention: congress, treaty, conventions, constitution, delegate, meeting, forum, declaration,
```

Figure 12: Nearest words 2

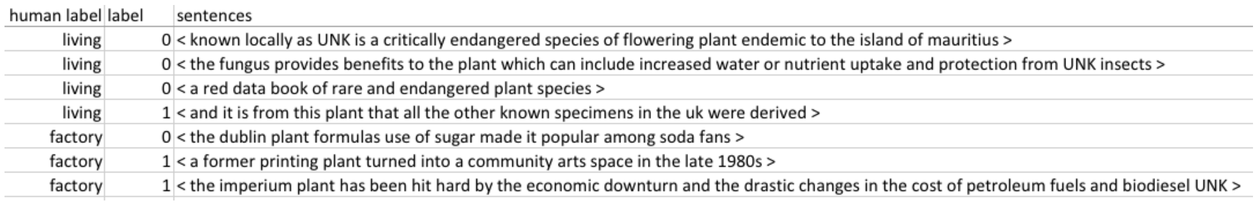
The final parameters we selected is show as in Table 6.

Parameters	VOC_SIZE	SKIP_WINDOW	NUM_SKIPS	EMBEDDING_SIZE
Value	50K	5	10	250
Parameters	LR	BATCH_SIZE	NUM_STEPS	NUM_SAMPLE
Value	0.3	256	Set to 6 epochs	3

Table 6: Parameters for our skip-gram model

6.2 Experiment with Classifying Word Senses

To train the classifier, we clustered the contexts of the occurrences of given ambiguous word into two/three coherent groups. To evaluate the classifier, we manually assigned labels to the occurrences of ambiguous words in the test corpus, and compared them with machine learned labels to calculate accuracy (to be described below). As shown in Figure 13, the column "human label" gives the human-tagged meanings and the column "label" gives the machine learned labels.



human label	label	sentences
living	0	< known locally as UNK is a critically endangered species of flowering plant endemic to the island of mauritius >
living	0	< the fungus provides benefits to the plant which can include increased water or nutrient uptake and protection from UNK insects >
living	0	< a red data book of rare and endangered plant species >
living	1	< and it is from this plant that all the other known specimens in the uk were derived >
factory	0	< the dublin plant formulas use of sugar made it popular among soda fans >
factory	1	< a former printing plant turned into a community arts space in the late 1980s >
factory	1	< the imperium plant has been hit hard by the economic downturn and the drastic changes in the cost of petroleum fuels and biodiesel UNK >

Figure 13: Test data examples

6.2.1 Metric

After we labelled the test sentences by hand, we could find the fraction of occurrences with the most frequent meaning. Before word sense determination, we assigned all occurrences to the most frequent meaning, and used the fraction as the baseline.

We calculated the disambiguation accuracy of the model as:

$$\text{accuracy} = \frac{\mathit{Number\ of\ instances\ with\ correct\ machine\ learned\ sense\ label}}{\mathit{The\ total\ number\ of\ test\ instances}}$$

6.2.2 Results

Table 7 show results of our word sense determination experiments and related results of Schütze's experiments.

word	senses	Training	Test	Schütze's baseline	Schütze's accuracy	Baseline	Accuracy
capital	Stock of goods/ Seat of government	179,793	100	64%	71%	59%	79%
plant	living/factory	164,858	100	54%	64%	59%	76%
ruling	an authoritative decision to exert control / influence			60%	84%	63%	70%
crane	bird/ machine/ person name	6,655	100	-	-	46%	68%
interest	A feeling of special attention/ A charge for borrowed money	112,903	100	58%	90%	86%	49%
train	benefit/drink	9,290	100	74%	69%	91%	57%

Table 7: Experiment results

"Schütze's baseline" column gives the fraction of the most frequent sense in his data sets. "Schütze's accuracy" column gives the results of his disambiguation experiments with local terms frequency if applicable. We got better accuracy out of experiments with “capital” and “plant”. However, the model cannot determine the senses of word “interest” and “sake”, which has a baseline over 85% in our data sets.

6.2.3 Discussions

Our data sets (12G) are much larger than Schütze’s data sets (435M). For example, the size of his training set for word “capital” is 13,015, and ours is 179,793. The larger data sets might have helped to increase the accuracy for some words.

We also observed that when the baseline is high (>= 85%), the model cannot determine the senses of the word. The performance of unsupervised learning relies on sufficient information from the training data. However, the model didn’t get trained with sufficient data carrying rare meaning. The size of the training data, and the distribution of the senses of the target word has significant influent to the performance.

7. Conclusion and Future Work

In this project, we utilized the distributional word representation and the distributional hypothesis to build a modular model to classify the senses of ambiguous words. We used the Skip-gram model to train word embeddings using 5 billion word pairs generated from Wikipedia pages. On the top of the word embeddings, we created vector to present context, and determined the sense of given ambiguous word by clustering the context vectors. Our experiments showed our model performed well when an ambiguous word had each sense accounts for than 20% of occurrences in the training data set.

In future work, we can try to optimize the classifier. One possible approach might be using weighted sum of contexts by taking IDF into account. We can also extend and experiment this approach to other models with different classifiers. The classifier which works well when occurrences are skewed to one cluster might improve the accuracy for words with large portion of occurrences are using the most frequent sense. In this project's implementation, we didn't tokenize words such as "worked" to "work", so the Skip-gram model could learn that "worked" is closer to "employed" than "employ". Learning such type of relationship is very useful in other tasks, such as translation. In disambiguation task, "He worked in a power plant" and "He is working in a power plant" doesn't make difference for the meaning of "plant", thus, we do not need both "worded" and "working" in vocabulary. By tokenize the corpus, we could reduce the time cost of training.

References

[1] Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155, 2003.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.

[3] G.E. Hinton, J.L. McClelland, D.E. Rumelhart. Distributed representations. In: Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations, MIT Press, 1986.

[4] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Language Learning, 2007.

[5] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by backpropagating errors. Nature, 323(6088):533–536, 1986.

[6] H. Schwenk. Continuous space language models. Computer Speech and Language, vol. 21, 2007.

[7] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, J. Cˇ ernocky´. Empirical Evaluation and Combination of Advanced Language Modeling Techniques, In: Proceedings of Interspeech, 2011.

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems, 2013a.

[9] James R. Curran and Marc Moens. Improvements in automatic thesaurus extraction. In Proceedings of the ACL-02 workshop on Unsupervised lexical acquisition, pages 59–66. 2002.

[10] Patrick Pantel and Dekang Lin. Discovering word senses from text. In Proc. Of SIGKDD-02, pages 613–619, New York, NY, USA. ACM. 2002.

[11] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In Proceedings of SIGDOC, pages 24-26, 1986.

[12] Olah, Christopher. Deep Learning, NLP, and Representations. Retrieved from http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/. 2014

[13] Hartigan, J. A. and Wong, M. A. Algorithm AS 136: A K-Means Clustering Algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics). 28 (1): pages 100–108, 1979.

[14] Schütze, Hinrich. Dimensions of meaning. In Proceedings of Supercomputing'92, pages 787-796, 1992.

[15] Pedregosa *et al.*, Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[16] Michael U Gutmann and Aapo Hyv¨arinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. The Journal ofMachine Learning Research, 13:307–361, 2012.

[17] Bottou L. (2010) Large-Scale Machine Learning with Stochastic Gradient Descent. In: Lechevallier Y., Saporta G. (eds) Proceedings of COMPSTAT'2010. Physica-Verlag HD

[18] TensorFlow Tutorial, tf.nn.nce_loss. Retriveved from https://www.tensorflow.org/api_docs/python/tf/nn/nce_loss. 2017

[19] McCormick, C, Word2Vec Tutorial Part 2 - Negative Sampling. Retrieved from http://www.mccormickml.com, 2017, January 11.

[20] D. Yarowsky, Unsupervised word sense disambiguation rivaling supervised methods, Proc. 33rd Annual meeting of the ACL, Cambridge, MA, USA, pp 189-196, 1995.

[21] Schütze, Hinrich, Automatic word sense discrimination, Computational Linguistics, v.24 n.1, March 1998.