Word embeddings for monolingual and cross-language domain-specific information retrieval

CHAYA WIGDER
Word embeddings for monolingual and cross-language domain-specific information retrieval

Chaya Wigder
wigder@kth.se

Master’s thesis in Computer Science
Date: August 8, 2018
Supervisor: Johan Boye
Examiner: Viggo Kann
Principal: Findwise AB
Supervisor at principal: Vide Karlsson
Swedish title: Ordinbäddningar för enspråkig och tvärspråklig domänspecifik informationssökning
School of Electrical Engineering and Computer Science
Abstract

Various studies have shown the usefulness of word embedding models for a wide variety of natural language processing tasks. This thesis examines how word embeddings can be incorporated into domain-specific search engines for both monolingual and cross-language search. This is done by testing various embedding model hyperparameters, as well as methods for weighting the relative importance of words to a document or query. In addition, methods for generating domain-specific bilingual embeddings are examined and tested. The system was compared to a baseline that used cosine similarity without word embeddings, and for both the monolingual and bilingual search engines the use of monolingual embedding models improved performance above the baseline. However, bilingual embeddings, especially for domain-specific terms, tended to be of too poor quality to be used directly in the search engines.

Keywords: information retrieval, domain-specific information retrieval, cross-language information retrieval, word embeddings, bilingual embeddings
Sammanfattning

Flera studier har visat att ordinbäddningsmodeller är användningsbara för många olika språkteknologiuppgifter. Denna avhandling undersöker hur ordinbäddningsmodeller kan användas i sökmotorer för både enspråkig och tvåspråklig domänspecifik sökning. Experiment gjordes för att optimera hyperparametrarna till ordinbäddningsmodellerna och för att hitta det bästa sättet att vikta ord efter hur viktiga de är i dokumentet eller sökfrågan. Dessutom undersöktes metoder för att skapa domänspecifika tvåspråkiga inbäddningar. Systemet jämfördes med en baslinje utan inbäddningar baserad på cosinuslikhet, och för både enspråkiga och tvåspråkliga sökningar var systemet som använder enspråkiga inbäddningar bättre än baslinjen. Däremot var de tvåspråkiga inbäddningarna, särskilt för domänspecifika ord, av låg kvalitet och gav för dåliga resultat för direkt användning inom sökmotorer.

Nyckelord: informationssökning, domänspecifik informationssökning, tvåspråklig informationssökning, ordinbäddningar, tvåspråkiga inbäddningar
Contents

1 Introduction ........................................... 1
  1.1 Problem ........................................... 1
  1.2 Research questions .............................. 2
  1.3 Approach ........................................ 3
  1.4 Limitations ....................................... 3
  1.5 Report overview .................................. 3

2 Background ........................................... 5
  2.1 Theoretical foundation ......................... 5
      2.1.1 Natural language processing .............. 5
      2.1.2 Vectors and matrices ...................... 7
      2.1.3 Machine learning ......................... 8
  2.2 Word vectors ..................................... 11
      2.2.1 Count-based vectors ....................... 12
      2.2.2 Word embeddings ......................... 12
      2.2.3 Bilingual word embeddings ............... 19
      2.2.4 Evaluation of word embeddings .......... 21
      2.2.5 Pretrained and domain-specific embeddings 22
  2.3 Information retrieval .......................... 24
      2.3.1 Classical methods ........................... 24
      2.3.2 Cross-language information retrieval .... 26
      2.3.3 Using word embeddings .................... 27
      2.3.4 Evaluation metrics .......................... 30

3 Experimental setup ................................. 33
  3.1 Datasets .......................................... 33
      3.1.1 IR collections .............................. 33
      3.1.2 Query translations ......................... 35
  3.2 Text preprocessing .............................. 35
3.3 Baseline search engine ........................................ 36
3.4 Embedding search engine ...................................... 37
  3.4.1 Word embedding models ................................. 37
  3.4.2 Bilingual mappings ..................................... 39
  3.4.3 Search engine ........................................... 41
3.5 Evaluation protocol ......................................... 43
3.6 Notes and limitations ....................................... 44

4 Experimental Results ........................................... 45
  4.1 Model selection & hyperparameter tuning .................. 45
    4.1.1 Monolingual baseline ................................. 45
    4.1.2 Monolingual word embeddings ....................... 46
    4.1.3 Word weights ........................................ 53
    4.1.4 Bilingual baselines ................................. 57
    4.1.5 Bilingual embedding mappings ........................ 57
  4.2 Test data .................................................. 61
  4.3 Analysis and examples ..................................... 64
    4.3.1 Monolingual .......................................... 64
    4.3.2 Bilingual ............................................. 65

5 Discussion and conclusions ..................................... 67
  5.1 Discussion of results ...................................... 67
    5.1.1 Monolingual information retrieval .................. 67
    5.1.2 Cross-language information retrieval ............... 68
  5.2 Future work .............................................. 69

Bibliography ..................................................... 70
Chapter 1

Introduction

1.1 Problem

The Internet contains hundreds of billions of pages and is growing each year [1]. In order for these pages to be useful, it is necessary to be able to automatically match relevant pages to a user’s information needs. This is the problem of information retrieval.

The most widespread solution to the information retrieval problem is the search engine: a user types a query, in words, and documents that somehow match the query are returned, in order of usefulness. Typically, documents are retrieved based on there being some overlap between the words in the query and the words in the documents.

However, relying on overlap between query words and document words is insufficient for a variety of reasons. Firstly, language is rich and one concept may be denoted by multiple words. The words used in the query might not be the same as those used in the documents. This problem can be exacerbated if a layperson user attempts to find domain-specific information; the user may not be familiar enough with the jargon of the domain to use it in the query.

More problematically, there may be no overlap at all between the query and the document, because they are not written in the same language. This is not a hypothetical problem: the 80% of the Earth’s population that does not speak English [2] may require information from the 52% of the Internet’s webpages that are written in English [3]. Moreover, 3.4% of people live outside of the countries of their birth [4], many in countries that use a different language than their native ones; they may require local information even if they do not speak the local
language.

Word embedding models are an approach to solving this problem. Word embeddings are representations of words that encode their meanings, rather than their spellings. This means that synonyms are treated as being nearly the same word, even if they are written differently. Moreover, words from different languages can be incorporated into a single word embedding model, and words with similar meanings will be treated similarly, no matter their language. Word embedding models have been applied to a wide variety of natural language processing tasks, including named-entity recognition, machine translation, and opinion mining. There has also been work incorporating word embedding models into information retrieval systems.

Previous research has shown that word embeddings are useful for the information retrieval problem, including for cross-language information retrieval and domain-specific retrieval. In addition, the effects on retrieval performance of some embedding and system configurations have been studied. The goal of this project is to more methodically and comprehensively outline the effect that various system choices have on performance, specifically in the cases of domain-specific and cross-language search. This goal is refined by the research questions.

## 1.2 Research questions

The research was guided by the following questions:

1. How should word embedding models be configured to make them useful for the information retrieval problem?

2. How should search engines be configured to best make use of word embeddings?

3. How should domain-specific bilingual mappings be generated?

Special focus was given to ways in which search engines are different from other applications of word embeddings, and in which embeddings change the usual rules of search engines. For example: Should words be weighted differently than is usual in search engines when they are represented as embeddings? Does what makes a word embedding "good" for the search engine application differ from what makes it good for other applications? These questions evolved with the study of previous literature and early experimental results.
1.3 Approach

The research questions were addressed, in order, through a series of experiments. First, word embeddings were incorporated in a monolingual search engine, and system and model parameters were configured based on tests on the development data. Using the results for the monolingual case, bilingual word embeddings were trained and incorporated into a cross-language search engine. Again, system and model parameters specific to the bilingual case were configured based on tests on the development data. Finally, the system was configured according to the findings of the experiments on development data, and the test data was evaluated using this system and configuration. These results were compared with performance using a non-embedding baseline search engine.

1.4 Limitations

Only free datasets were used in this project. This means that results cannot be directly compared to results in prior work, which are generally reported using datasets that cost money.

1.5 Report overview

The rest of the report is organized as follows:

Chapter 2 describes the ideas and algorithms that this project is based on. Alternative algorithms that were not used are also briefly mentioned so as to explain the choices made. The chapter contains the following sections:

- **Section 2.1** defines some terms used later in the chapter, and is provided as reference in case the reader comes upon some unfamiliar concept in the subsequent sections.

- **Section 2.2** explains in depth the algorithms used to generate the monolingual word vectors and bilingual mappings used in this project. It also provides a brief overview of alternative methods for generating bilingual embeddings.
Section 2.3 describes non-embedding monolingual and bilingual search engines and explains their shortcomings. It then shows how word embeddings have been used in other information retrieval systems.

Chapter 3 describes the datasets used, the setup of the baseline and the system under test, and the experiments to be performed.

Chapter 4 shows the results of experiments on the development data for hyperparameter tuning, as well as results of experiments on the test data.

Chapter 5 discusses the findings of this report and proposes some ideas for future work.
Chapter 2

Background

This chapter presents the ideas and models underlying this thesis project. The first section defines terms and equations that are used throughout the report, and should be used as reference for unfamiliar concepts. Subsequent sections present theory and related work in the fields of word representations, information retrieval, and the combination of the two.

The notation in this chapter follows the below convention:

\[ \mathbf{v} \] \quad \text{vector } \mathbf{v} \\
\[ M \] \quad \text{matrix } M \\
\[ s_{i,k} \] \quad \text{sequence } s \text{ consisting of elements } s_i \text{ to } s_k \\

To avoid excessive notation, the vector representation of an item \( e \) is denoted by the same variable printed in boldface \( \mathbf{e} \).

2.1 Theoretical foundation

Definitions and equations in this section are provided for reference, and are presented in relation to this project. Other uses or definitions may be omitted.

2.1.1 Natural language processing

Text collections

All natural language tasks rely on text collections for data. The following are some types of collections used in the related work and in this project:

Corpus \: a collection of texts.
**Parallel corpus**  a collection of texts in two or more languages.

**Sentence- or word-aligned corpus**  a parallel corpus where texts are roughly translations of each other, and a mapping is present between each sentence or word pair across languages with the same meaning.

**Comparable corpus**  a parallel corpus where texts are not translations of each other, but are aligned by topic.

**Language models**

A language model is a probability distribution over all possible sequences of words or characters in a language’s vocabulary. More formally, given a vocabulary $V$ which is a set of words $w$, the language model consists of

$$\forall w_{1..m} \in V^*: p(w_{1..m})$$

These probabilities cannot be computed for each possible sequence individually since sequence lengths, and therefore the number of sequences, may be infinite. Instead, the simplifying Markov assumption is made. That is, the probability of each word in the sequence is considered to be dependent only on the $k$ words coming before it. The probabilities can be rewritten as

$$p(w_{1..m}) = p(w_m \mid w_{1..m-1})p(w_{1..m-1})$$
$$= p(w_m \mid w_{m-k+1..m-1})p(w_{1..m-1})$$
$$= \prod_{i=1}^{m} p(w_i \mid w_{i-k+1..i-1})$$

Using this simplification, the probability of any sequence can be computed as a combination of probabilities of sequences of length $n = k + 1$. Sequences of length $n$ are called n-grams; bigrams and trigrams are n-grams of length 2 and 3 respectively.

One way to establish n-gram probabilities is to count the number of occurrences of each n-gram in a training corpus.

**Tf-idf**

Term frequency-inverse document frequency, or Tf-idf, is a metric that weights words by their importance to an individual document and to the
corpus as a whole. Words that are rare in the corpus overall but appear many times in a given document have high tf-idf scores, while words that are common in the corpus or don’t appear often in the document have lower scores. The tf-idf score for a word \( w \) in a document \( d \) and collection \( C \) is computed as follows:

\[
\text{term frequency} = \frac{\text{number of times } w \text{ appears in } d}{\text{number of documents in } C \text{ containing } w}
\]

To more evenly distribute the weights, usually the logs of the numerator and/or denominator are used instead of the original counts.

### 2.1.2 Vectors and matrices

**Cosine similarity**

Cosine similarity is a measure of the similarity between two vectors \( \mathbf{a}, \mathbf{b} \in \mathbb{R}^n \). It is a normalized form of the dot product that does not depend on the length of the two vectors, and is computed by taking the cosine of the angle between the two vectors:

\[
\text{similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}}
\]

In general, cosine similarity can range from -1, meaning completely opposite, to +1, meaning completely similar. However, in many language applications all the elements of the vectors are non-negative, leading to a similarity range between 0 and 1.

**Principal component analysis**

Given a set of data points, each consisting of multiple possibly correlated dimensions, the task of principal component analysis (PCA) is to remove the correlation between the dimensions by means of an orthogonal transformation of the data points. The uncorrelated dimensions are called principal components. PCA is often used as a first step when compressing data, since once the dimensions are uncorrelated those with low variance can be discarded without much loss of information.
**Singular value decomposition**

One method of performing principal component analysis is singular value decomposition (SVD). SVD is the factorizing of an $m \times n$ matrix into three components

$$X_{m \times n} = U_{m \times m} \Sigma_{m \times n} W_{n \times n}^T$$

(2.3)

If $X$ is real-valued, $U$ and $W$ will be orthogonal (that is, $MM^T = M^T M = I$), and $\Sigma$ will be diagonal and non-negative. The values in the diagonal of $\Sigma$ are called singular values, and are typically in descending order. If the rows of $X$ represent data points and columns represent dimensions, $U$ and $W$ are the rotation matrices that decorrelate the dimensions, and the singular values indicate the relative importance of each dimension in the new space.

**Procrustes analysis**

Procrustes superimposition is the task of superimposing shapes by means of rotation, translation, and scaling, and is one of the techniques used when generating bilingual embedding mappings. The orthogonal Procrustes problem is a special instance of this problem. Given two matrices $A$ and $B$, the goal here is to find the orthogonal matrix $T$ which best maps between the two:

$$\arg \min_T |AT - B|_F \quad \text{subject to} \quad T^T T = T T^T = I$$

(2.4)

$|.|_F$ is the Frobenius norm, defined as

$$|M_{m \times n}|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} [M_{ij}]^2}$$

(2.5)

A closed form solution for this problem was provided by Schönemann [5]. The matrix can be found by SVD of $BA^T$:

$$BA^T = U \Sigma W^T$$

(2.6)

$$T = U W^T$$

(2.7)

**2.1.3 Machine learning**

Machine learning is the computer task of generalizing from training data points to unseen data points. The focus of this project is not on
Example 2.1: An example dataset for supervised learning. Each point consists of 4 dimensions, and is associated with a real-valued label.

machine learning techniques; however, it is useful to know the following terms to understand the related work.

**Supervised and unsupervised learning**

In supervised learning, each data point is associated with a label, which is either a real number or a class. For example, the data point can be an image of a face and the label the age of the person in the image, or the data point can be a piece of text and the label can be the language of the text. The goal of the system is to learn a function that maps the input data to its label. An example dataset for supervised learning is shown in example 2.1.

In unsupervised learning, the data points are not associated with labels, but the system can still learn. For example, given a set of texts in different languages, the system can learn to cluster them into groups which have the same language or to map translations to each other, without knowing the names of the languages.

**Models and parameters**

In the machine learning models described in this report, the type of function to be learned is determined in advance by the human programmer. Types of functions can be, for example, linear functions of the input, nonlinear functions, and combinations of the two, as shown in example 2.2. The goal of learning is to set the parameters of the functions, called $\theta$.

Neural networks are models that consist of several layers of linear and nonlinear functions, as in the last function in example 2.2. As models get more complex it becomes cumbersome to list each variable,
\[ h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 \]
\[ = \theta^T x \]
\[ h_\theta(x) = \tanh(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4) \]
\[ = \tanh(\theta^T x) \]
\[ h_\theta(x) = \theta_0 \tanh(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4) + \theta_{10} \tanh(\theta_5 + \theta_6 x_1 + \theta_7 x_2 + \theta_8 x_3 + \theta_9 x_4) \]
\[ = \theta_0^T \tanh(\theta_1^T x) \]

Example 2.2: Increasingly complex functions of \( x \), parameterized by \( \theta \)

\[ l(\theta) = \frac{1}{2} (h_\theta(x) - y) \]
\[ J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \]

Objective : \( \arg \min_{\theta} J(\theta) \)

Example 2.3: A set of loss, cost, and objective functions

and they are usually represented using vectors and matrices.

**Loss, cost, and objective functions**

The loss function is the error in the system associated with a single data point, for example, the difference between the actual label and the label predicted by the system using the learned function. The cost function is the overall error in the system, and is usually the sum of the loss across all data points along with some regularization to ensure that the system does not fit too closely to the data, which may be noisy. The objective function is the overall function the system is designed to maximize or minimize, and is often simply the negation of the cost function. An example set of loss, cost, and objective functions is shown in figure 2.3.
Gradient descent

Gradient descent is a method for finding the minimum of the objective function. The system runs for several iterations, and at each iteration moves in the direction opposite to the gradient of the loss. In plain gradient descent, the total cost of the entire system is computed at every iteration. A faster variation is stochastic gradient descent, which uses a single random example at each iteration.

Expectation maximization

Expectation maximization is a method for computing the parameters of a model when some of the training data is unknown. It works by initially guessing the values for the missing data, then iteratively updating the parameters of the model to match the guesses for the data and the guesses for the data to match the parameters of the model.

For example, in an unsupervised model that clusters unlabeled data, the missing data is the label, and the model parameters are the parameters for the distributions that govern each cluster. In the first step, a label is randomly assigned to each data point. In each following iteration, the distribution parameters are first updated to most closely match the current labels for the data, and then the labels are reassigned so that each point has the label of the cluster that gives it the highest probability using the current parameters.

2.2 Word vectors

A word vector is a real-valued, d-dimensional vector representation of a word. The simplest kind of word vector is a one-hot representation, where the dimensionality is the size of the vocabulary and only the dimension corresponding to each word’s unique index in the vocabulary is non-zero. This representation has two major downsides: the vectors are high-dimensional and sparse, and the representations are meaningless - they do not encode any semantic or other information about the words.

Other implementations of word vectors take advantage of the distributional hypothesis to encode semantic meaning in the word vectors. The distributional hypothesis of linguistics is the idea that words with similar meanings appear in similar contexts, or, as popularized by J. R.
Firth, ”You shall know a word by the company it keeps!” [6]. Therefore, it is possible to represent the meaning of words by considering the documents that contain them and other words that regularly appear nearby them.

2.2.1 Count-based vectors

Distributional vectors directly encode the context of words into their vectors. They come in two major forms: term-document vectors and context vectors. Both require a corpus of texts to establish the context of a word, and differ mainly in how much context is used.

In term-document vectors, each dimension represents a document in the collection. The vector for each word encodes how many times that word appears in each document. The underlying assumption is that words with similar meanings appear in the same documents, and therefore will have similar vectors. This method leads to very high-dimensional vectors when there are many documents in the collection, and singular value decomposition (SVD; see section 2.1.2) may be used to reduce the dimensionality.

Context vectors remove the explicit meanings associated to the dimensions. Instead, each word vector is set to a combination of the vectors for all other words that occur in its context in the corpus, for example within \( n \) words. One efficient implementation uses random-indexing to compute the vectors. Each word is given a random, semi-sparse initial vector which is then used to compute the context vectors for the other words.

2.2.2 Word embeddings

Descriptive vectors explicitly count the distribution of words in the corpus. Word vectors can also be obtained as a byproduct of some predictive task on the data. These vectors are called word embeddings.

Neural network models

Word embeddings were first proposed in 2003 by Bengio et al. [7]. The authors trained a neural network to generate an \( n \)-gram language model, that is, the probability function \( p(w_t | w_{t-1..t-n+1}) \). Since words are discrete, an explicit mapping of this probability function for all possible combinations of words in the vocabulary \( V \) would require determining
Figure 2.4: The neural network language model. Note that the $C$ matrix values are shared. Bias vectors have been omitted for simplicity.

$|V|^n - 1$ free parameters. By instead representing words as real-valued vectors, the language model can be represented as a continuous probability function and the number of free variables is reduced.

A simplified version of the network architecture is described in equation 2.8 and figure 2.4$^1$.

$$p(w_t \mid w_{t-1..t-n+1}) = \frac{e^{y_{w_t}}}{\sum_{w \in V} e^{y_w}}$$

$$y_{w_t} = b + U \tanh(d + Hx)$$

$$x = (C(w_{t-1}), ..., C(w_{t-n+1}))$$

$C(w_t)$ is the row in $C$ corresponding to the index in the vocabulary of the word at location $t$ in the corpus.

The free variables here are the biases $b$ and $d$, the linear transformation matrices $U$ and $H$, and the word representation matrix $C$. The number of free variables here is linear in $|V|$ and $n$, yet the network provides as output not only a language model but also vector representations of words.

$^1$This version removes variables that were set to zero in the most successful experiments.
The language model produced by this method outperformed earlier methods, by as much as 8% and 24% on two test corpora. However, it is extremely slow to train. One reason is that the normalizing softmax layer requires computing the probabilities for every word in \( V \), which can be prohibitive for large vocabularies. Morin and Bengio [8] propose a hierarchical softmax based on a class-based softmax described in Goodman [9]. In the class-based softmax, the probability computation is split into two steps: predicting the word class and then predicting the word given the class:

\[
p(w_t \mid w_{t-1..t-n+1}) = p(w_t \mid c, w_{t-1..t-n+1})p(c \mid w_{t-1..t-n+1})
\]  

(2.9)

If the words can be optimally split into classes such that there are \( \sqrt{|V|} \) classes with \( \sqrt{|V|} \) words in each, the speedup is \( \frac{|V|}{2\sqrt{|V|}} \) times, which even for a vocabulary size as small as 10,000 word translates into a 50x speedup. Hierarchical softmax extends this idea by using not just a single intermediary layer, but rather a balanced binary tree. Each word in the vocabulary is represented as a binary string indicating its exact location in the tree, and the probability of each word becomes the product of the probabilities of each successive bit:

\[
p(b \mid w_{t-1..t-n+1}) = \prod_{b_i \in b} p(b_i \mid w_{t-1..t-n+1})
\]  

(2.10)

The bit probabilities are computed similarly to how the overall probabilities are computed in the original model - each internal node has a vector associated with it and the structure of the parameters is the same. Hence, the speedup is increased to be of order \( \frac{|V|}{\log_2|V|} \). Morin and Bengio generated the binary tree using a modified version of the WordNet IS-A hierarchy. While the speedup was significant, it did come with a considerable cost in accuracy.

The hierarchical method by Mnih and Hinton [10] was able to surpass non-hierarchical methods in accuracy by instead automatically generating the binary tree, as well as by allowing words to appear multiple times in the tree to account for multiple senses. The binary tree is generated using an expectation-maximization algorithm, where in each iteration, the tree is generated using the word vectors learned in the previous iteration.

As part of a unified model trained to solve a variety of NLP problems, Collobert and Weston [11] reframed the language modeling problem in
a way that would be used by most subsequent embedding frameworks. In previous models, the goal was to compute the probability of a word given the previous words in a sentence, and trained embeddings were obtained merely as a side effect. In this new framework, the goal of obtaining embeddings was made explicit. Removing the traditional language modeling formulation has two benefits: the middle word of the context window can be used as the target word, potentially improving the predictive power of the context, and the slow softmax layer can be removed since there is no need to compute probabilities. The loss function is designed to differentiate between real word sequences and sequences where the middle word has been replaced by some random word:

\[
J = \sum_{s \in d} \sum_{w \in d} \max(0, 1 - f(s) + f(s^w))
\]  

(2.11)

Here, \( s \) is a true sequence present in the training document \( d \), \( s^w \) is that sequence with the middle word replaced with word \( w \), and \( f(\cdot) \) is the output of the neural network. When used in downstream NLP tasks, the embeddings trained in this way produced results equivalent with the state-of-the-art.

word2vec

The adoption of word embeddings for downstream tasks was accelerated with the publishing of word2vec, a fast method for training embeddings that was first outlined in Mikolov et al. [12] and further extended in Mikolov et al. [13]. word2vec comes in two flavors: continuous bag-of-words (CBOW) and continuous skip-gram. Both models consist of an input matrix that projects the one-hot input onto the single hidden layer, a linear hidden layer of the same dimension as the word embeddings, and one or more output matrices. The structures of these networks are shown in figure 2.5.

The CBOW model is similar to the original neural network language model in that it uses context words to predict a pivot word, with two differences: the context window includes words both before and after the pivot word, and the vectors of the context words are averaged rather than concatenated. This second change makes the complexity invariant to the size of the context. In the skip-gram model, the input to the model is a single pivot word and the goal is to predict the other words in its
Figure 2.5: word2vec: CBOW on left and skip-gram on right.

context:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} \mid w_t)$$

(2.12)

Since farther words are generally less related than closer words, they are sampled less frequently during training so as to give them lower weight.

In addition to the above, there are two additional changes in word2vec that serve to speed up training: The non-linear (tanh) layer is removed, and the hierarchical tree for softmax is not balanced by number of words but instead uses Huffman codes to balance by word probability. Overall, these changes make it possible to train over billions of tokens and millions of vocabulary words, compared to, for example, the only 1 million tokens and 30,000 vocabulary words used by Collobert.

Another alternative to softmax described in [13] is negative sampling. A negative sample for a given word is another word that does not appear in its context. For each objective word, $k$ negative samples are drawn at random according to their probability in the corpus using a probability function $P_n$. The goal is to maximize the vector similarity between a word and its context words while minimizing the similarity between
that word and other words, approximated by the negative samples. The objective function is thus to maximize

$$\log \sigma(w^T w_c) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \left[ \log \sigma(-w_i^T w_c) \right]$$

(2.13)

where \(\sigma(x) = \frac{1}{1+e^{-x}}\) and \(v_1^T v_2\), the unnormalized dot product, is the measure of vector similarity.

Overall, the authors reported that word2vec outperformed earlier methods significantly, most likely at least in part because it was trained on much more data and produced higher dimensional data. Despite the additional data and dimensions, word2vec was also faster to train; training 1000-dimensional vectors on 30 billion tokens took only a day, compared to at least 7 days reported for earlier methods.

Certain hyperparameter choices can affect the types of embeddings that are generated. Levy and Goldberg [14] compare window sizes of 2 and 5 and show that smaller context windows result in embeddings that cluster functionally similar words (words of the same class and similar meaning, such as ”pizza” and ”hamburger”) while larger context windows result in embeddings that also cluster topically similar words (for example ”pizza” and ”eat”). These differences are reflected in the different evaluation metrics for embeddings, which are discussed in section 2.2.4.

Subword models

Although these word embeddings help provide probability to unseen combinations of words, they still cannot handle completely out-of-vocabulary words. This is especially problematic highly inflected languages such as Finnish and Yupik, where each individual inflected form is relatively rare. It can also be a problem in languages that make heavy use of compounds, such as Swedish or even domain-specific English.

Bojanowski et al. [15] extended the skip-gram model to incorporate subword information when training. This model contains both the words in the training data as well as all character n-grams present in these words. An example is shown in figure 2.6. In this model, each character n-gram is represented by a vector in the same way as words are, and each word is represented by the sum of the vectors for the character n-grams it contains. The similarity between a word \(w\) composed of the
Training text walking the walk

With start- and end-word markers <walking> <the> <walk>

3-grams <wa wal alk lki king ng> <th the he> lk>

4-grams <wal walk alki lkin king ing> <the the> alk>

5-grams <walk walki alkin lking king> <the> walk>

6-grams <walki walkin alking lking> <walk>

Figure 2.6: An example dataset with its character n-grams. Underlined n-grams are shared between words.

Figure 2.7: Skip-gram model including subword information. To limit memory usage, n-grams are bucketed into b buckets.
set of n-grams $G_w$ and some other vector $v$ is then

$$\text{sim}(w, v) = \sum_{g \in G_w} g^T v$$  \hspace{1cm} (2.14)$$

The structure of this network is shown in figure 2.7. Training proceeds in the same way as ordinary \texttt{word2vec}, with the addition that any updates made to the word vector are also made to its constituent n-grams. This allows information to be shared between words with the same roots or compound words and their components. Rare words can get more accurate representations, and even words that weren't encountered during training can get a more informative representation than the zero-vector. This model, implemented in the \texttt{fasttext} library, outperformed both of the original \texttt{word2vec} baselines for a variety of languages. The improvement was most significant in Russian, which has six cases for nouns, and German, which has four cases for nouns and makes heavy use of compound words.

\subsection*{2.2.3 Bilingual word embeddings} 

The goal when training multilingual word embeddings is to map words from two or more languages into the same vector space so that words with similar meanings, even if they are from different languages, have similar vectors. There are two major approaches used to generate bilingual word embeddings: learning a mapping between two separately trained monolingual vector spaces, or learning a single bilingual vector space trained on some form of bilingual data.

\textbf{Mappings} 

An early example of the mapping approach followed directly from the original \texttt{word2vec} implementation. When visualizing word vectors for different languages, Mikolov, Le, and Sutskever [16] noted that related words such as numbers tended to cluster in similar geometric formations across different languages. They exploited this insight to compute, using stochastic gradient descent, a translation matrix $W$ which best maps embeddings for training words $s_i$ in the source language to their counterparts $t_i$ in the target language. Once $W$ is computed, it can be
used to translate any vector from the source space to the target space.

$$\min_W \sum_i^n \| Ws_i - t_i \|^2$$  \hspace{1cm} (2.15)$$

The authors achieved over 90% translation precision-at-5 using this method, leading to many extensions following a similar approach. For example, Xing et al. [17] find that performance is improved on similarity and translation tasks when vectors are normalized, $W$ is constrained to be orthogonal, and the objective function is amended to optimize for cosine similarity rather than Euclidean distance:

$$\max_W \sum_i^n (W s_i)^T t_i$$  \hspace{1cm} (2.16)$$

While the methods above use a training dictionary, it is also possible to compute the translation matrix in an unsupervised manner. This is useful for low-resource languages, where seed lexicons may not be available. Conneau et al. [18] used a three-step approach to train and refine a mapping given only embeddings for $n$ words in the source language and $m$ words in the target language, but no seed lexicon. First, they trained an adversarial network to generate a rough mapping matrix $\tilde{W}$. The objective of the discriminator is to distinguish between target language vectors and mapped source language vectors:

$$L(\theta_D | \tilde{W}) = - \frac{1}{n} \sum_{i=1}^n \log p_{\theta_D}(source=S | \tilde{W}s_i)$$

$$- \frac{1}{m} \sum_{i=1}^m \log p_{\theta_D}(source=T | t_i)$$  \hspace{1cm} (2.17)$$

The objective for $\tilde{W}$ is the exact opposite - to fool the discriminator, thus creating a close overall mapping. In the second step, they used $\tilde{W}$ to induce a bilingual dictionary by finding the nearest opposite-language neighbor for the most frequent words. This was done under the assumption that more frequent words would appear in more consistent contexts across the two languages, and thus be more accurately mapped. Using this lexicon, a new mapping $W$ was induced using eq 2.15 along with an orthogonality constraint on $W$ and solved using Procrustes. Finally, the two embedding spaces were rescaled to make cross-language nearest
neighbor more symmetric, that is, if the nearest translation of \( s_i \) is \( t_j \), the nearest translation of \( t_j \) should be \( s_i \).

For many language pairs this unsupervised method outperformed or performed similarly to supervised methods for training the mapping matrix. However, this method performed relatively poorly for the more distant language pairs English-Chinese and English-Russian. Specifically, for most language pairs the refinement step brought the quality of the unsupervised mappings close to their supervised counterparts, indicating that the induced bilingual lexicons were of comparable quality to the manually generated ones.

**Joint training**

An alternative to the mapping approach is to train a single model containing the vocabulary of both languages, by using either a bilingual corpus or a bilingual objective function. The first method was implemented by Vulić and Moens [19]. They generated a pseudo-bilingual training corpus by concatenating different-language Wikipedia on the same topic and trained word vectors on this corpus using ordinary word2vec. The second method was implemented by Zou et al. [20]. They updated the original monolingual loss function defined by Collobert to include a term that measured inter-language loss based on a word-aligned bilingual corpus.

Both of these methods showed promising results - Vulić’s on related European languages and Zou’s on the entirely dissimilar language pair English-Chinese. However, unlike the mapping method, these methods require an aligned bilingual corpus and cannot directly take advantage of pretrained embeddings.

**2.2.4 Evaluation of word embeddings**

Natural language models, including word embeddings, can be evaluated either intrinsically or extrinsically. Extrinsic evaluation means that the model is incorporated into some downstream task, such as document classification or part-of-speech tagging, and then the larger system is evaluated. Intrinsic evaluation, on the other hand, means that the model is tested directly using a precomputed test corpus or set of queries.

There are two popular intrinsic evaluation tasks used to test word embedding models: similarity and analogy. In word similarity, the test
collection contains pairs of words along with their similarity scores. This is then compared to the similarity between the vectors for those words. Some of the datasets used for this task are WordSim-353 [21], SimLex-999 [22], and SemEval [23]. The main difference between WordSim and SimLex is that the former gives high scores both to pairs that are semantically similar and those that are related, while the latter only considers similarity. [22] illustrate this difference with the following examples:

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>WordSim</th>
<th>SimLex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonym</td>
<td>cup-mug</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Similar</td>
<td>frog-toad</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Related</td>
<td>car-tire</td>
<td>↑</td>
<td>↓</td>
</tr>
</tbody>
</table>

SemEval also limits itself to similarity, but aims to improve on the quality of SimLex and also includes named entities and multiword phrases. Both SimLex and SemEval are available in other languages.

An alternative to the similarity test is the analogy test. An analogy is a set of two pairs of words that share a similar relationship, such as the syntactic analogy

\[ \text{big} : \text{bigger} :: \text{small} : \text{smaller} \]

To pass the analogy test, the following must hold:

\[ \text{vector(big)} - \text{vector(bigger)} + \text{vector(small)} \approx \text{vector(smaller)} \]

where \( \approx \) indicates the closest vector. An example of a semantic analogy is

\[ \text{Paris} : \text{France} :: \text{Stockholm} : \text{Sweden} \]

The test was introduced by Mikolov et al. [12] together with \texttt{word2vec}, and helped show that word vectors encode meaning in their layout beyond just putting similar words near to each other.

### 2.2.5 Pretrained and domain-specific embeddings

In general, the performance of word embeddings goes up when trained on a larger corpus. For example, in the original \texttt{word2vec} formulation, accuracy on a word relationship test increased monotonically from 23% to 46% as the number of training words was increased from 24 million to 783 million [12]. It is therefore common to train embeddings on large datasets containing billions of words, such as Wikipedia or Common
Table 2.1: Corpus, corpus size, and vocabulary size for pretrained embeddings provided with most common embedding models

<table>
<thead>
<tr>
<th>Model</th>
<th>Training corpus</th>
<th>Corpus</th>
<th>Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe²</td>
<td>Wikipedia + Gigaword</td>
<td>6B</td>
<td>400K</td>
</tr>
<tr>
<td></td>
<td>Common Crawl</td>
<td>840B</td>
<td>2.2M</td>
</tr>
<tr>
<td></td>
<td>Twitter</td>
<td>27B</td>
<td>1.2M</td>
</tr>
<tr>
<td>word2vec³</td>
<td>Google News</td>
<td>100B</td>
<td>3M</td>
</tr>
<tr>
<td>fastText⁴</td>
<td>Wikipedia</td>
<td>16B</td>
<td>1M</td>
</tr>
<tr>
<td></td>
<td>Common Crawl</td>
<td>600B</td>
<td>2M</td>
</tr>
</tbody>
</table>

Crawl, and pretrained vectors are available for the major embeddings models (see table 2.1).

While training corpus size matters, often domain-specific applications vectors trained on smaller, in-domain corpora perform better than those trained on larger, general purpose or out-of-domain datasets. For example, Joshi et al. [24] found that in-domain vectors trained on less data were more useful than general domain vectors, though using a combination of both in-domain and out-of-domain vectors gave the best results. Stenetorp et al. [25], on the other hand, found mixed results; for named-entity recognition models trained on in-domain data performed best, while for semantic category disambiguation the size of the training corpus mattered more than its domain. (The models tested here were distributional, but not specifically word vectors.) For fields with highly specialized vocabulary such as biomedicine, it is common to train domain specific embeddings, and domain-specific resources are publicly available from, among others, Moen and Ananiadou [26] and Chiu et al. [27].

Especially when in-domain corpora are sparse, it can be useful to incorporate structured domain information into the vectors. For example, for biomedicine structured information may be a mapping of diseases to their symptoms or treatments. Roy, Park, and Pan [28] proposed a method of incorporated annotations derived from structured data into the classic word2vec objective function, and showed promising results using this method within the cybersecurity domain to rank malware by

²https://nlp.stanford.edu/projects/glove/
³https://code.google.com/archive/p/word2vec/
similarity.

2.3 Information retrieval

Information retrieval is the task of retrieving relevant information from a collection of documents or other data items, given an information need. This section describes classical methods for both monolingual and cross-language information retrieval, as well as the word embedding-based methods which form the foundation for the work in this report.

2.3.1 Classical methods

Boolean retrieval

The oldest and simplest method of information retrieval is Boolean retrieval: given a query and a set of documents, find the subset of documents that is relevant to the query. This is done by retrieving those documents containing either all or some of the tokens present in the query. Although this model is elegant in its mathematical simplicity - information retrieval becomes series of set unions and intersections - it cannot differentiate between degrees of relevance. A document containing a list of all English names is deemed exactly as relevant to the query ”Shakespeare” as a biography of the playwright.

Ranked retrieval

The vector-space model instead measures similarity between the query and documents in order to perform ranked retrieval. Both the query and the documents are represented as term-vectors, where each dimension of term vector represents a word and its value corresponds to the weight, commonly tf-idf, of that word in the document or query. Documents are then ranked by their vector distance to the query.

Query expansion and relevance feedback

Because these two methods use the exact terms present in the queries and documents, they fail to account for synonyms, which are pairs of words sharing one meaning, and polysems or homographs, which are words with multiple senses or meanings, respectively. An article about roses should be returned for a query for ornamental flowers, even if
it does not contain the exact words of the query. Likewise, an article about a financial institution should not necessarily be returned for a query about the edge of a river, even if both the query and document contain the word "bank".

These issues can be handled by augmenting the original query with additional data, either by query expansion or by relevance feedback. The following descriptions are based on Schütze, Manning, and Raghavan [29].

Query expansion mainly addresses the problem of synonyms. In this method, the query is updated to include all words that are synonyms or otherwise related to the words in the original query. These synonyms can be found using a manually or automatically created thesaurus. While query expansion may improve recall, it may also cause precision to decrease due to the problems of polysemy and homography; for example, the query "river bank" may be expanded to "river bank financial institution". In addition, if the thesaurus is automatically generated from the collection being searched, even recall may not increase much since both the original terms and their synonyms will be present in the retrieved documents.

Relevance feedback addresses the problems of both synonyms and polysemes, and is generally useful when it may be difficult to formulate a query. In this method, the original search proceeds as usual. Then, the user marks some of the retrieved documents as relevant. The query is then updated using this relevance information, and the search is repeated. In the classic Rocchio implementation, the query vector is updated so that it is closer to the vectors of the relevant documents and farther from the vectors of the irrelevant documents.

**Probabilistic formulation**

The information retrieval problem can be reframed as the estimation of the probability that a given document $d$ is relevant to a query $q$ composed of words $w$, or $p(d \mid q)$. The probability can be expanded by using Bayes’ rule and the assumption that the words in the query are conditionally independent of each other given the document:

$$p(d \mid q) \propto p(q \mid d)p(d) = p(d) \prod_{w \in q} p(w \mid d)$$  \hspace{1cm} (2.18)

Intuitively, $p(q \mid d)$ can be read as the probability that a user wishing to retrieve document $d$ will include $q$ in the query, and the conditional
independence implies that including any word in the query is dependent only on the document itself, and not on the other words in the query. The probabilities $p(d \mid q)$ can be used to rank documents for a query -documents with higher probability are ranked higher than those with low probability. Practically, $p(w \mid d)$ can be computed by sampling $w_q$ from $d$; that is

$$p(w \mid d) \propto \frac{\text{count}(w, d)}{\# \text{words in } d}$$  \hspace{1cm} (2.19)

This formulation can be used to motivate the use of term-frequency in relevance ranking, and will be used below to provide the theoretic basis for latent semantic analysis as well as using word embeddings in information retrieval.

**Latent semantic analysis**

Latent semantic analysis (LSA) is a method for extracting the topics present in the document collection. Each topic is a combination of words, and each word may appear in more than one topic. For example, in a gardening corpus some topics may be

$\text{topic}_A = .28 * flowers + .13 * gardening + .12 * roses + .11 * tulips...$

$\text{topic}_B = ...$

Topics are computed by running SVD on a term-document matrix - the same method used to generate distributional word vectors. In this case, however, the dimensions corresponding to words, not documents, are compressed. In the transformed matrix, each document is represented as a vector whose dimensions are topics and values are the relative importance of each topic to the document. LSA can be used to measure the similarity between documents by comparing their vectors, as well as similarity of a query to a document by transforming the query into the same topic space as the documents using the factored matrices from SVD.

### 2.3.2 Cross-language information retrieval

The methods for information retrieval listed so far rely at least to some extent on there being an overlap in the words used in the query and words present in the relevant documents. In the world of cross-language
information retrieval, this assumption is no longer valid. For some language pairs there may be some shared terms, for example proper nouns. Other language pairs may have different scripts or writing systems and not share any terms at all.

Because of this, cross-language information retrieval traditionally relies on translation as a basic component. Oard and Dorr [30] describe three major approaches to cross-language text retrieval: text translation, dictionaries, and statistical methods.

The first approach, text translation, relies on translating (using automatic machine translation) either the documents or the queries. In general, it is more efficient to translate the queries, as this can be done at query-time and does not require additional storage of translations. However, because queries are relatively short, they often lack the context needed to make an accurate translation and a single mistake can have high impact (for example, if the wrong meaning for a homograph is chosen).

The second approach uses a bilingual dictionary to convert the query terms from the query language to the document language. This is often done as a form of query expansion - all possible translations are used, which addresses the problem of single translation at the possible expense of precision. The problem with this approach is that dictionaries are difficult to manually create and to maintain as language evolves.

The third approach takes advantage of statistics in bilingual or parallel texts. Either the texts are used to automatically generate a bilingual dictionary, and search is performed as in the second approach. Alternatively, if part of the collection to be searched consists of parallel texts while the rest consists of monolingual texts, the search can first be performed on the parallel texts, and the top matches can be used for query expansion based on relevance feedback to search the monolingual texts.

### 2.3.3 Using word embeddings

Ganguly et al. [31] use the probabilistic formulation of information retrieval to provide the theoretical basis for incorporating word embeddings in this task. As discussed, the probabilistic formulation ranks documents based on their probability given the query:

\[
p(d \mid q) \propto p(q \mid d)p(d) = p(d) \prod_{w_q \in q} p(w_q \mid d) \quad (2.18 \text{ revisited})
\]
The authors extend this by considering the probability that a word \( w_d \) is generated from a document \( d \) and then transformed into the word \( w_q \) in the query, denoted by \( p(w_d, w_q \mid d) \). Using the rules of conditional probabilities this is expanded to

\[
p(w_d, w_q \mid d) = p(w_q \mid w_d, d)p(w_d \mid d)
\] (2.20)

The probability \( p(w_q \mid w_d, d) \) can be seen as the probability that a word \( w_d \) in document \( d \) is transformed into a different word, \( w_q \), in the query. One method to compute this probability is to use the similarity between the two words:

\[
p(w_q \mid w_d, d) = \frac{\text{sim}(w_q, w_d)}{\sum_{w \in d} \text{sim}(w_q, w)}
\] (2.21)

Using word similarity, \( p(w_q \mid d) \) in equation 2.18 can now be expanded to account for not only the query word itself, but also the count of other words in the document weighted by their similarity to the query word.

In order to implement this efficiently, the nearest neighbors for each word were stored in memory, and only these were used when computing the probability. Overall, the authors reported that this method outperformed other language model methods.

Vulić and Moens [32] also use word embeddings to improve information retrieval, but instead of explicitly computing the similarity of each term in the query, they compose the embeddings of constituent terms to get the document and query vectors, and then compute the cosine similarity between the two. They present two methods of composing word vectors. The first, ADD-BASIC, is simple vector addition:

\[
d = \sum_{w \in d} w
\] (2.22)

The second, called ADD-SI (for self information), uses a variation of idf weighting. Each word is weighted according to its frequency in the corpus:

\[
si(w) = -\log \frac{\text{# times } w \text{ appears in collection}}{\text{# tokens in collection}}
\] (2.23)

\[
d = \sum_{w \in d} \text{si}(w) w
\] (2.24)

The authors generated bilingual embeddings using pseudo-bilingual texts, and tested both monolingual and bilingual retrieval. In the monolingual case, a simple language model-based retrieval system outperformed the embedding-based retrieval, but combining the embedding
and language models using linear interpolation had the best results. All these models outperformed simple LDA. In the multilingual setting the results were mixed based on the language, but overall a combined system using a linear combination of the embedding-based scores and LDA+language model scores performed the best.

Most applications of word2vec, including the IR applications listed above, use the input layer of the neural network to get the word embeddings. However, Nalisnick et al. [33] found that information retrieval results improved if a combination of the input and output layers were used. Their best performing IR system used input layer vectors for the query terms and output layer vectors for the document terms. Similarity was defined as the dot product of the query and document vectors, which were in turn defined as the average of the normalized vectors of their constituent terms.

\[
\text{sim}(q, d) = q_{IN}^T d_{OUT}
\]

\[
q_{IN} = \frac{1}{|q|} \sum_{q_i \in q} \frac{q_{IN,i}}{\|q_{IN,i}\|}
\]

\[
d_{OUT} = \frac{1}{|d|} \sum_{d_i \in d} \frac{d_{OUT,i}}{\|d_{OUT,i}\|}
\]

The authors provided a likely reason for this: words with high IN-IN similarity were more typically similar, that is, they fill the same function in a sentence, while words with high IN-OUT similarity were more topically similar. For example, the word Yale is typically similar to the names of other American universities such as Harvard or NYU, but topically similar to general academic terms such as faculty or graduate. A high concentration of topically similar words in a document indicates that the document is about the topic, rather than merely mentions it. This may indicate that for information retrieval high WordSim scores, which measure both similarity and relatedness, are more important than high SimLex scores, which measure only similarity.

Diaz, Mitra, and Craswell [34] studied the impact of using domain-specific rather than general purpose embeddings for query expansion. They found that embeddings trained only on topically related documents resulted in better performance, even if the number of training documents was smaller.

Schakel and Wilson [35] suggest that the length of a word vector (that is, the l2-norm) can be indicative of the word’s significance. Auxiliary
Table 2.2: Human judgment vs system judgment

<table>
<thead>
<tr>
<th></th>
<th>Relevant</th>
<th>Irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Not retrieved</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

words that appear in many different contexts, such as “and”, will have short vectors, words that appear frequently but in similar contexts will have longer vectors, and words that appear infrequently and in similar contexts will have longer vectors still. This means that vector length can be used to determine the important words in a text. Although the authors do not apply this to information retrieval, if the relationship between word length and significance hold, it may be useful as a complement to or replacement for inverse document frequency.

2.3.4 Evaluation metrics

For a given query, each document in the collection can be judged relevant or irrelevant to the query by a human evaluator. In addition, the system makes its own relevance judgments, retrieving documents it deems relevant and not retrieving others. The combination of human and system judgments leads to four classes of documents:

**True positive (TP)** relevant document that is retrieved by the system

**False positive (FP)** irrelevant document that is retrieved by the system

**False negative (FN)** relevant document that is not retrieved by the system

**True negative (TN)** irrelevant document that is not retrieved by the system

The goal of an IR system is to match the human judgment as closely as possible, so that relevant documents are retrieved and irrelevant documents are not retrieved. The simple measure of this is accuracy, or the percentage of the total where human and system judgments match:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \quad (2.26)$$

Because there tend to be far more irrelevant documents for each query than relevant ones, it’s possible for a system to get a high accuracy score
by simply not retrieving any documents. Measuring precision and recall separately avoids this problem.

**Precision (P)** percentage of retrieved documents that are relevant

**Recall (R)** percentage of relevant documents that are retrieved

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{2.27}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{2.28}
\]

In cases where it is desirable to have a single number representing the system, precision and recall can be combined using the harmonic mean, which tends toward the lower of the two values. This score is called the F-score. The balanced F-score, known as the F1-score, is computed as

\[
\text{F1-score} = \frac{2PR}{P + R} \tag{2.29}
\]

It is also possible to adjust this measure to give higher weight to either precision or recall.

In ranked retrieval systems, earlier results are more important than later ones. Two ways of measuring this are precision-at-k and average precision. Precision-at-k (P@k) measures precision after the the first k results; for example, on a typical web search P@10 may be used to measure the quality of the first page of results. While precision-at-k is easy to compute, since it requires only the first k results to be examined, it still gives equal weight to the top k results, and no weight to the remaining results. Average precision instead monotonically decreases the weight given to each result. It is computed as

\[
\text{Average precision} = \frac{\sum_{k=1}^{\text{# retrieved documents}} \text{rel}(k) \times \text{P@k}}{\text{# relevant documents}} \tag{2.30}
\]

\[
\text{rel}(k) = \begin{cases} 
1, & \text{if document at rank } k \text{ is relevant} \\
0, & \text{otherwise} 
\end{cases} \tag{2.31}
\]

For example, if there is one relevant document, returning it at 1st, 5th, and 10th place will lead to MAP@10 of 1.0, 0.2, and 0.1 respectively. If there are 10 relevant documents, returning one relevant document
at 1st, 5th, and 10th place will lead to MAP@10 of 0.1, 0.02, and 0.01 respectively; if the top half of documents are relevant the MAP@10 will be 0.5 while if the bottom half are relevant MAP@10 will be 0.18.

For an entire system tested on multiple queries, the mean average precision (MAP), that is, average precision averaged over all queries, is used.
Chapter 3

Experimental setup

This chapter describes the datasets used for development and testing, gives an overview of the implementation of both the baseline and embedding search engines, and describes how the experiments were run and evaluated. It also lists the third-party libraries and pretrained models used in the embedding search engine. The specific outcomes of hyperparameter tuning and other configuration that was determined experimentally is described in chapter 4.

3.1 Datasets

3.1.1 IR collections

The information retrieval collections were obtained from the Information Retrieval group at University of Glasgow\(^1\) and the Text Retrieval Conference hosted by the National Institute of Standards and Technology (TREC-NIST)\(^2\). Each collection contains a set of documents, a set of queries, and a mapping of queries to relevant documents.

Below is a description of the datasets used. For development, ADI, Time, and OHSU development was used, and for test, CACM, NPL, Medline, and OHSU test were used.

\(^1\)http://ir.dcs.gla.ac.uk/resources/test_collections/

\(^2\)http://trec.nist.gov/data/t9_filtering.html
ADI

<table>
<thead>
<tr>
<th>Source</th>
<th>University of Glasgow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Information retrieval</td>
</tr>
<tr>
<td>Documents</td>
<td>82 article abstracts</td>
</tr>
<tr>
<td>Queries</td>
<td>35, often as one or multiple longer questions</td>
</tr>
<tr>
<td>Notes</td>
<td>Small collection size limits usefulness</td>
</tr>
</tbody>
</table>
| Sample query | *What possibilities are there for automatic grammatical and contextual analysis of articles for inclusion in an information retrieval system?*

Time

<table>
<thead>
<tr>
<th>Source</th>
<th>University of Glasgow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>News</td>
</tr>
<tr>
<td>Documents</td>
<td>423 news stories</td>
</tr>
<tr>
<td>Queries</td>
<td>83 topics, usually fairly detailed</td>
</tr>
<tr>
<td>Notes</td>
<td>Dated topics and language decrease effectiveness of pre-trained embeddings, for example &quot;Viet Nam&quot; instead of &quot;Vietnam&quot;. In addition, certain relevance judgments are mislabeled and had to be fixed manually.</td>
</tr>
<tr>
<td>Sample query</td>
<td><em>EFFORTS OF AMBASSADOR HENRY CABOT LODGE TO GET VIET NAM’S PRESIDENT DIEM TO CHANGE HIS POLICIES OF POLITICAL REPRESSION.</em></td>
</tr>
</tbody>
</table>

OHSU (development and test)

<table>
<thead>
<tr>
<th>Source</th>
<th>TREC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Medicine</td>
</tr>
<tr>
<td>Documents</td>
<td>54,710 development and 293,851 test, titles and/or abstracts from medical journals, sourced from MEDLINE, an on-line medical information database</td>
</tr>
<tr>
<td>Queries</td>
<td>63 descriptions of patients and queries about their medical issues</td>
</tr>
<tr>
<td>Notes</td>
<td>This dataset is split into development and test, which use different document collections but the same queries</td>
</tr>
<tr>
<td>Sample query</td>
<td><em>60 yo male with disseminated intravascular coagulation; pathophysiology and treatment of disseminated intravascular coagulation</em></td>
</tr>
</tbody>
</table>
Medline

Source: University of Glasgow
Domain: Medicine
Documents: 1,033 titles and abstracts from medical texts
Queries: 30 descriptions of information needs, in various lengths
Sample query: the crystalline lens in vertebrates, including humans.

CACM

Source: University of Glasgow
Domain: Computing
Documents: 3,204 titles, some with abstracts
Queries: 64 descriptions of information needs, often as would be presented to another human
Sample query: What articles exist which deal with TSS (Time Sharing System), an operating system for IBM computers?

NPL

Source: University of Glasgow
Domain: Electronics / physics
Documents: 11,429 journal abstracts
Queries: 93 queries, mostly search engine format
Sample query: MEASUREMENT OF DIELECTRIC CONSTANT OF LIQUIDS BY THE USE OF MICROWAVE TECHNIQUES

3.1.2 Query translations

For the bilingual search engine, a subset of the queries were manually translated into Swedish. For Time, 20 queries were translated, and for OHSU and Medline, all queries were translated.

3.2 Text preprocessing

The text of queries and documents were preprocessed as follows:

- The text was lowercased. This was especially necessary because the Time collection is in all caps.
• Spaces were added before and after non-alphanumeric characters.
• The text was tokenized by splitting on spaces.
• Words appearing in at least 80% of documents were considered stopwords and removed. The cutoff of 80% was chosen by validating on the development datasets, although values between 60% and 90% all had the same overall performance.
• In the OHSU collection, most queries start with ’<age> yo’ or equivalent. In the translated queries, these were treated as stopwords and removed rather than translated.

### 3.3 Baseline search engine

The baseline search engine used tf-idf and cosine similarity. Specifically, each document and query was considered to be a bag of words with stopwords removed:

\[
d = \text{multiset}(w \not\in \text{stopwords})
\]

\[
q = \text{multiset}(w \not\in \text{stopwords})
\]

Each document and query was assigned a length based on the Euclidean length:

\[
\text{len}(d) = \sqrt{\sum_{w \in d} \text{count}(w, d)^2}
\]

A document was considered a match for a query if it contains at least one word present in the query:

\[
\text{match}(d, q) = d \cap q \neq \emptyset
\]

Each word in the vocabulary was weighted using its document frequency:

\[
df(w) = -\log_{10} \frac{\sum_{d \in C \mid w \in d} 1}{|C|}
\]
Each document-word pair was also weighted, this time using term frequency:

\[
\text{tf}(w, d) = \begin{cases} 
1 + \log_{10} \text{count}(w, d) & w \in d \\
0 & w \notin d
\end{cases}
\]  

(3.6)

The matching documents and queries were scored according to their weighted cosine-similarity:

\[
\text{score}(d, q) = \frac{\sum_{w \in d \cap q} \text{df}(w) \text{tf}(w, d) \text{tf}(w, q)}{\text{len}(d) \text{len}(q)}
\]  

(3.7)

**Bilingual search engine**

In the bilingual search engine, queries were translated into the document language using Google Translate, and then the search was performed as in the monolingual case.

### 3.4 Embedding search engine

The embedding search engine consists of a word vector model, an index that stores the vector representation of documents, and a search functionality that converts queries into vectors and finds the closest documents. The rest of this section will describe these elements.

#### 3.4.1 Word embedding models

**Libraries**

Monolingual word embeddings were trained using the fasttext library \(^3\), which implements word2vec with subword information as described in Bojanowski et al. [15] and in section 2.2.2. This method was chosen since both the medical texts of OHSU as well as the Swedish texts in general seemed likely to benefit from training using character-level data. The extent of this benefit was also tested, as described below.

Bilingual word embeddings were generated by mapping the monolingual embeddings into a single space using the supervised method.

---

\(^3\)https://fasttext.cc
in the MUSE library \(^4\), which is described in Conneau et al. [18] and in section 2.2.3. The mapping method was chosen because it could be used in conjunction with domain-specific data and does not require any aligned data beyond a small bilingual dictionary.

**Pretrained and domain-specific models**

It can be impractical to train embedding models from scratch for every application, and pretrained models are available for many languages and domains. Below are the pretrained models tested in this project.

**Wiki** English model trained using fasttext on Wikipedia, UMBC’s web crawl, and statmt.org news corpus \(^5\). General purpose, with possible bias towards news (Time).

**Crawl** English model trained using fasttext on the Common Crawl\(^6\).

**Pubmed** English model trained using word2vec on the biomedical databases PubMed and PMD [26]\(^7\). In-domain for OHSU.

**Mapped** English and Swedish models trained using fasttext on Wikipedia, and then mapped into a common space using MUSE \(^8\).

In addition to pretrained models, two types of collection-specific models were tested. The first type is a fasttext model trained only on the document collection. Models of this type most closely reflect their document collections, since they contain no outside data. The second type is a hybrid - a fasttext model trained on the document collection, but using a pretrained model as starting weights for the vectors. These hybrid models require fewer epochs to train, since vectors are initialized to reasonable values. They also have a larger vocabulary, since they contain not only the words in the document collection but also those in the pretrained model.

\(^4\)https://github.com/facebookresearch/MUSE  
\(^6\)https://fasttext.cc/docs/en/english-vectors.html  
\(^7\)http://bio.nlplab.org/  
\(^8\)https://github.com/facebookresearch/MUSE
Hyperparameter tuning

When training embeddings, it is necessary to configure a variety of hyperparameters, including the number of training epochs, the window size, and the minimum and maximum lengths for character n-grams. These parameters were tuned by running experiments on the development datasets, and the results of these experiments are listed in chapter 4. In order to understand where hyperparameters affect each other, the hyperparameters were tested using a grid search - that is, each hyperparameter was tested with all combinations of values for the other hyperparameters. Any hyperparameters not mentioned in this report were set to the fasttext defaults.

There is inherent randomness when training word embeddings, and different models trained with the same hyperparameters can show up to 2 percentage point difference in performance. Therefore, the goal of the grid search was to find general trends and find hyperparameters that generally worked well, rather than blindly picking the specific combination that led to the best performance.

3.4.2 Bilingual mappings

To get bilingual word embeddings, two separately trained monolingual embedding models were mapped to each other using the supervised method MUSE⁹ framework. In the supervised method, two monolingual embeddings are mapped into a single space using a bilingual dictionary and Procrustes alignment. A bilingual Swedish-English dictionary containing 5000 words is included with the MUSE framework.

Several mappings were tested:

Premapped  Aligned English and Swedish embeddings provided along with the MUSE framework.

Pretrained  Pretrained Crawl models for English and Swedish mapped locally using the MUSE framework.

Domain-specific  Mappings between domain-specific embedding models and / or trained using domain-specific dictionaries.

Initial experiments for the bilingual search engine using non-domain specific mappings were done using the Time and OHSU collections.

⁹https://github.com/facebookresearch/MUSE
Time was used to approximate a domain that closely matches available pretrained embeddings, while OHSU was used to approximate a very specific domain that does not match available pretrained vectors. Domain is more significant in the bilingual case than in the monolingual case for two reasons: the training data for the query language embedding and the mapping dictionary.

In the monolingual case, the queries and documents are expected to have significant overlap in vocabulary, so embeddings trained on the collection documents could be directly used to encode queries. In the bilingual case, however, there is little or no overlap between query and document terms, and since query terms are not known in advance, query-language embeddings must be trained on external data or pretrained embeddings must be used. To examine the impact of domain-specific training data for the query-language embeddings, two types of query-language embeddings were tested: pretrained embeddings trained on all of Wikipedia and Common Crawl, and smaller embedding models trained on only relevant pages from Wikipedia. For OHSU, relevant pages were those that fell under the Medicine (Medicin) or Anatomy (Anatomi) categories on Wikipedia.

Similarly, the mapping dictionary can be either general-purpose or domain-specific. General purpose dictionaries are provided together with MUSE, and mappings were generated using these. In addition, domain-specific dictionaries were compiled for the medical domain. These were generated using English MeSH terms and their Swedish translations. MeSH\(^{10}\), or Medical Subject Headings, is a hierarchical list of terms used for indexing and cataloging biomedical texts. It is distributed by the U.S. National Library of Medicine, National Institutes of Health. Many of the terms have been translated to Swedish by Karolinska Institutet Universitetsbiblioteket\(^{11}\) and are available as part of the standard MeSH distribution. For the purposes of these experiments, the dictionary consisted of 4428 pairs, and included only such pairs where the original and translation both contained only one word, and the original was present in the OHSU documents. These choices were made to simplify the generation of the dictionary.

\(^{10}\) [https://www.nlm.nih.gov/mesh/](https://www.nlm.nih.gov/mesh/)
\(^{11}\) [https://mesh.kib.ki.se/](https://mesh.kib.ki.se/)
3.4.3 Search engine

Document and query vectors

The vector representation for each document or query was set to be the weighted sums of the vectors for its constituent words.

\[ d = \sum_{w \in d} weight_w \cdot w \]  

(3.8)

Note that \( d \) is treated as a multiset, with words able to appear more than once. The weight for each word was based on its inverse document frequency, in the same way as in the baseline:

\[ weight_w = \log_{10} \frac{\text{# of documents in collection}}{\text{# of documents in collection containing } w} \]  

(3.9)

Word vectors can be of different lengths. If used unnormalized, i.e., without their lengths set to 1, words with longer vectors will have higher weight than shorter vectors, even without considering tf-idf. This is only desirable if vector length is a reasonable indicator of word significance. Experiments were run both with and without normalizing word vectors; the results are reported in chapter 4.

Pretrained embedding models and hybrid collection-pretrained models can contain vectors for words not present in the document collection. Because of this, the embedding search engine can take advantage of query words that the baseline search engine would have to discard. To do this, it is necessary to assign a weight to words not in the document collection that does not depend on their document frequency. The following approaches were tested: setting the weight of these words to zero (essentially discarding them), using the minimum or average weight of all in-vocabulary words for out-of-vocabulary words, and using Wikipedia to compute the document frequencies, since it has a larger vocabulary that is more similar to the vocabulary of the pretrained embeddings. The results of these experiments are reported in the next chapter.

Indexing and search

At startup, each document was converted into a vector as described above. The document vectors were indexed using Spotify’s annoy frame-
work\textsuperscript{12}, which allows for fast approximate nearest neighbor search of high-dimensional vectors.

At query time, the query string was converted into a vector using the same method as for the documents. A nearest neighbor search was performed to find the document vectors in the index with the highest cosine similarity to the query vector. The rank for each document was set to the cosine similarity between the query and the document; higher ranked results had higher cosine similarity. Since embeddings can contain negative values, the ranking ranged from -1, most dissimilar, to +1, most similar. Note that the annoy framework returns the Euclidean distance between the normalized vectors, a strictly monotonic function of cosine similarity. The Euclidean distance can be converted to cosine similarity using the Law of Cosines for finding the third side of a triangle, 
\[ c^2 = a^2 + b^2 - 2ab \cos \theta, \]
with \( a = b = 1 \) because the vectors are normalized:
\[ \text{cosine similarity}(u, v) = 1 - \frac{\text{distance}(u, v)^2}{2} \] (3.10)

**Bilingual search engine**

In the bilingual search engine, the overall process was similar to that of the monolingual search engine. However, instead of using a single embedding model to convert both the documents and the queries into vectors, two mapped embedding models were used - one for the query language and one for the document language.

Queries and documents were converted into vectors using their own language embeddings. For query tokens that were out-of-vocabulary in the query-language model, the document-language embedding model was used as a backup. This was done to account for proper nouns and similar.

Document words were weighted using tf-idf, just as in the monolingual case. However, because document-frequency is not available for words in the query language, all words were given equal weight. In addition, preliminary testing was done with using document frequencies obtained from Wikipedia.

When mapping two different embedding models into the same space, the MUSE framework discards any subword information included in the models. The subword embeddings were reconstructed by computing

\textsuperscript{12}https://github.com/spotify/annoy
the character n-grams for all words in the model and setting each n-gram to the normalized sum of all the words in the model containing that n-gram.

\[
c = \frac{\sum w}{\sqrt{\left( \sum w \right)^2}}
\]  \hspace{1cm} (3.11)

Preliminary testing was done using these reconstructed subword embeddings for Swedish-language embeddings, either for all words or only as a backup for out-of-vocabulary words.

### 3.5 Evaluation protocol

Each experiment was evaluated on each of the datasets separately. Results were reported using the mean average precision of the first ten results for each query, or MAP@10. Because experiments with word embeddings involve some inherent randomness, in certain cases results are reported as the average of multiple runs.

The experiments were organized as follows:

**Baseline** Baseline experiments were run one time per development and test collection, using the system as described in section 3.3.

**Hyperparameter and system setup - monolingual** These experiments were performed on the development collections. A grid search was run across all embedding hyperparameter and system setup choices, except where otherwise noted. Results for each hyperparameter or system option are reported in two ways: as the average of the top results for each value, to show where a given hyperparameter choice places an upper limit on performance, and as the change in performance when all hyperparameters but one were kept constant.

**Hyperparameter and system setup - cross-language** Same as in monolingual case.

**Test** Experiments were run on each test collection using the optimal system setup found in the tests on the development data. Each test was repeated three times with the same hyperparameters and system setup, and results were reported as the average of the runs.
3.6 Notes and limitations

This project used only collections provided at no cost. Some of these collections were relatively small compared to real-world datasets, both in the number of documents as well as the documents’ lengths. Furthermore, the number of queries was small, meaning that performance was highly sensitive to which queries happened to be in the datasets.

Query translations were done manually as best-effort by the author, who has Swedish as a second language and limited medical knowledge. Many medical terms have two types of possible translations - more technical Latin-derived terms as well as less technical Swedish-derived terms, for example, hepatit or leverinflammation for hepatitis. In general, an effort was made to match the style of the original query, and this determined which translation was chosen. In many cases, the choice of translation term could have large impact on the results, especially if one of the terms was not present in the model.

Because the aim of this project was to investigate unsupervised models, no language-specific preprocessing such as stemming or compound-splitting was done. In a real-world implementation, language-specific preprocessing should be done to improve performance. This is especially true for languages where less training data is available, or languages where words can have many forms.
Chapter 4

Experimental Results

This chapter lists the results of all experiments done on the embedding search engine. Section 4.1 shows the results of the experiments that were run on the development data in order to configure the search engine, including hyperparameter tuning for training the embeddings. The findings of these experiments are then put to test in section 4.2, which details the overall performance of the system on the test data. To gain some insight into why embedding models work the way they do, the results were analyzed with the help of examples in section 4.3.

4.1 Model selection & hyperparameter tuning

Three datasets were used for development and hyperparameter tuning: the tiny ADI collection, the news-domain Time collection, and the medical-domain OHSU collection. The collections are described in detail in section 3.1.1.

4.1.1 Monolingual baseline

The baseline results for the monolingual search engine are shown in table 4.1. These and the rest of the results in this chapter are listed using the mean average precision of the top ten results, or MAP@10, as described in 2.3.4.

OHSU in particular was plagued with short documents (containing titles) that tended to be ranked more highly even if they were irrelevant, which partly accounts for the low scores.
Table 4.1: MAP@10 for the baseline search engine

<table>
<thead>
<tr>
<th></th>
<th>MAP@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.29</td>
</tr>
<tr>
<td>Time</td>
<td>0.48</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.09</td>
</tr>
</tbody>
</table>

Table 4.2: MAP@10 across pretrained, hybrid, and collection-trained embeddings. Collection and hybrid embeddings shown are the average of top 10 results.

<table>
<thead>
<tr>
<th></th>
<th>Collection</th>
<th>Crawl</th>
<th>Wiki</th>
<th>Pubmed</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.28</td>
<td><strong>0.35</strong></td>
<td>0.29</td>
<td>0.27</td>
<td>0.29</td>
</tr>
<tr>
<td>Time</td>
<td><strong>0.60</strong></td>
<td>0.50</td>
<td>0.33</td>
<td>0.13</td>
<td>0.58</td>
</tr>
<tr>
<td>OHSU</td>
<td><strong>0.21</strong></td>
<td>0.14</td>
<td>0.10</td>
<td>0.14</td>
<td>0.20</td>
</tr>
</tbody>
</table>

4.1.2 Monolingual word embeddings

Domain specific embeddings

To test the importance of using embeddings trained specifically for the domain or on the documents of a given IR collection, MAP@10 was measured using pretrained, collection-trained, and hybrid (i.e., collection-trained using pretrained as starting weights) embeddings.

The results are shown in table 4.2. For all collections, the Crawl pretrained embeddings outperformed the Wiki embeddings, with MAP@10 between 0.04 and 0.16 higher when using Crawl. The medical-domain Pubmed embeddings performed similar to the Crawl embeddings for OHSU (for which it was in-domain), though more poorly than the Wiki embeddings for the other two collections. For ADI, the Crawl model, which gave a performance of 0.3479, outperformed all other pretrained, hybrid, and collection-only models, which all had performance ranging between 0.27-0.29. For both the Time and OHSU collections, collection-only embeddings performed best, outperforming the closest pretrained embeddings by 0.10 and 0.07 respectively. The hybrid embeddings performed slightly worse than the collection-only embeddings, by 0.02 and 0.01 respectively.

Although the higher performing embeddings typically also showed
Table 4.3: Out-of-vocabulary rates across embeddings

<table>
<thead>
<tr>
<th></th>
<th>Coll</th>
<th>Crawl</th>
<th>Wiki</th>
<th>Pubmed</th>
<th>Crawl+Coll</th>
<th>Wiki+Coll</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
</tr>
<tr>
<td>Time</td>
<td>0.17</td>
<td>0.014</td>
<td>0.018</td>
<td>0.042</td>
<td>0.002</td>
<td>0.001</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.22</td>
<td>0.007</td>
<td>0.013</td>
<td>0.005</td>
<td>0.004</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Table 4.4: New words added during hybrid training

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>Crawl</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Time</td>
<td>655</td>
<td>475</td>
</tr>
<tr>
<td>OHSU</td>
<td>14077</td>
<td>9492</td>
</tr>
</tbody>
</table>

A lower out-of-vocabulary rate (see table 4.3), OOV rates cannot entirely explain the difference in performance. For both the Time and OHSU collections, the collection-only embeddings had both the highest OOV rate and the best performance. Table 4.4 shows the number of unique words that appeared at least twice in each collection’s documents but never in the pretrained embeddings, and thus were added in the hybrid models. The larger the collection the more words were added: ADI, Time, and OHSU added 11, 655, and 14077 words to the Wiki embedding respectively. The reason for the large number of words added by OHSU may not only be due to its large collection size; OHSU contains many medical-domain words that may not be common enough to be contained in general-purpose models.

Training epochs

In the grid search, 5, 20, and 40 training epochs were tested. The results are shown in figure 4.1. On average, increasing the number of epochs from 5 to 40 while keeping other hyperparameters constant led to increased performance for all collections and model types (collection-only or hybrid). For ADI the average increase in MAP@10 was 0.06 and 0.01 for collection-only and hybrid mappings respectively, for Time it was 0.24 and 0.15, and for OHSU it was 0.02 and 0.03. Except for in ADI collection-only models, most of this gain happened when increasing from 5 to 20 epochs.
(a) Change in MAP@10 with change in number of epochs. Each point represents a pair of experiments with all hyperparameters equal except for the number of epochs.

(b) MAP@10 vs epochs

Figure 4.1: Impact of number of epochs on MAP@10, split by model type. Each point represents a single experiment in the grid search.
Table 4.5: Window size vs performance for locally trained Wikipedia embeddings

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>20</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.32</td>
<td>0.32</td>
<td>0%</td>
</tr>
<tr>
<td>Time</td>
<td>0.56</td>
<td>0.54</td>
<td>-5%</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.032</td>
<td>0.027</td>
<td>-18%</td>
</tr>
</tbody>
</table>

**Window size**

Four different window sizes were tested in the grid search: 5, 10, 20, and 40. The results are shown in figure 4.2. Also here, on average, increasing the window size from 5 to 40 while keeping other hyperparameters constant led to an increase in performance: 0.03 and 0.01 for ADI collection-only and hybrid models, 0.17 and 0.09 for Time models, and 0.02 and 0.04 for OHSU models. In this case, each incremental window size increase, from 5 to 10 to 20 to 40 led to a similar sized gain in performance.

However, not all hyperparameter settings benefit equally from increasing the window size. For both Time and OHSU, the largest gains when increasing the window size were when the number of epochs was 5, with increases of 0.31 and 0.05 respectively. At 20 epochs this fell to 0.06 and 0.02; and at 40 the increase was only 0.03 and 0.02. This trend held for both collection-only and hybrid models, though it was more pronounced for the collection-only models. For ADI the relationship was less clear-cut. For the hybrid models the trend was the same, although increasing the window size had a slight negative effect for 20 and 40 epochs (less than 0.01). For the collection-only models, no clear pattern emerged - at 5 epochs the ideal window size was 10 or 20, at 20 epochs it was 5 and 40, and at 40 epochs performance increased with window size until 20, and then leveled off.

While increased window size generally corresponded with higher performance for collection-only and hybrid models, this was not the case when using a model trained on Wikipedia, as shown in table 4.5. Here performance fell across all collections when the window size was increased from 5 to 20 - by less than 0.01 for ADI and OHSU and by 0.03 for Time.

Just like when increasing the number of training epochs, increasing the window size also increased training time. In general, models with
(a) Change in MAP@10 with change in window size, split by model type

(b) Window size vs performance, split by model type

(c) Window size vs performance for collection-only models, split by epoch

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>( \Delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.28</td>
<td>0.27</td>
<td>0.28</td>
<td>0.29</td>
<td>+5%</td>
</tr>
<tr>
<td>Time</td>
<td>0.56</td>
<td>0.58</td>
<td>0.59</td>
<td>0.60</td>
<td>+7%</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.19</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
<td>+5%</td>
</tr>
</tbody>
</table>

(d) Average performance of 10 top models for each collection and window size, and change in MAP@10 from window size 5 to 40.

Figure 4.2: Impact of window size on MAP@10
Table 4.6: Minimum subword size vs MAP@10, as average of 10 top-performing models

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adi</td>
<td>0.28</td>
<td><strong>0.28</strong></td>
<td>0.28</td>
<td>0.28</td>
<td>0.28</td>
</tr>
<tr>
<td>Time</td>
<td><strong>0.59</strong></td>
<td>0.59</td>
<td>0.59</td>
<td>0.59</td>
<td>0.58</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.20</td>
<td>0.20</td>
<td><strong>0.20</strong></td>
<td>0.20</td>
<td>0.19</td>
</tr>
</tbody>
</table>

window size 40 took around four times as long to train as those with window size 5.

**Subword embeddings**

Two tests were run to evaluate the impact of differing amounts of subword information, one at training time and one at test time. At training time, the minimum character n-gram was varied from 3 to 6, or no subword information was used. A minimum character n-gram length of \( n \) means that all subword n-grams of size \( n, n + 1 \), up to size 6 were included in the model. At test time, either out-of-vocabulary terms were discarded, or they were set to be the sum of their constituent n-grams.

The results for the training-time experiments are shown in table 4.6 and figure 4.3, and results for the test-time experiments are shown in figure 4.4. All collections, both for hybrid and collection-only models, were able to achieve their highest performance when using some amount of subword data during training, though the ideal minimum subword length and whether the performance increase persisted if OOV terms were set to zero was different for different collections. For OHSU, minimum subword lengths of 4-6 improved performance by around 0.01; for Time and ADI the improvements were smaller but consistent. In general, models trained for more epochs were more likely to benefit from using subword information, while those trained for only 5 epochs were more likely to see performance drop when using subword information.

For hybrid models, the change when using subword information for OOV models was always small (up to 0.01), most likely since there were few OOV terms in these models. For collection models the change was small on average, but varied more: most configurations saw a small increase - up to 0.04 and 0.01 for ADI and OHSU respectively, while some saw a larger decrease - up to 0.08 and 0.02 respectively. (For Time,
Figure 4.3: Change in MAP@10 when training with subword information as compared to no subword information, split by number of training epochs.

Figure 4.4: Change in MAP@10 when OOV terms were treated as sums of subwords

the change was less than 0.01 in almost all cases.) For both ADI and OHSU, the poorer performance when using subword information for OOV terms occurred mainly when smaller minimum n-gram lengths were used - 3 or 4 for ADI and 3 for OHSU. In addition, for ADI lower performance when using subword information for OOV terms also corresponded to fewer training epochs and smaller window sizes.

There are two practical considerations when using subword embeddings: time and space. Training with character n-grams takes more time than training with just word unigrams, and training with more subword information, for example, by having a lower threshold for the minimum n-gram length, takes more time than training the same number of epochs with less subword information (however it is possible that training vectors using subword information requires fewer epochs to get the same results). For example, training OHSU using subgrams
Table 4.7: Embedding dimensionality vs MAP@10, collection-only. Percent change 50d to 300d.

<table>
<thead>
<tr>
<th></th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.26</td>
<td>0.26</td>
<td>0.26</td>
<td>0.25</td>
<td>-3%</td>
</tr>
<tr>
<td>Time</td>
<td>0.53</td>
<td>0.57</td>
<td>0.57</td>
<td>0.58</td>
<td>+10%</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.14</td>
<td>0.17</td>
<td>0.19</td>
<td>0.20</td>
<td>+45%</td>
</tr>
</tbody>
</table>

of at least size 3, subgrams of at least size 4, and no subgrams took 276, 245, and 175 minutes respectively. In addition, the models containing subword information are much larger - over 2 gigabytes for subword models compared to less than 200 megabytes for the word-only models.

**Dimensions**

Outside of the grid search, an experiment varying the number of dimensions was performed, with all other parameters held constant. The number of dimensions used in `word2vec` embeddings typically ranges from 50 to 300, and this experiment tested four values: 50, 100, 200, and 300. Since the dimensionality of hybrid models is restricted to that of the pretrained vectors, only collection-only models were tested. The results for this experiment are shown in table 4.7. For ADI, performance decreased somewhat as dimensionality increased, going down 2.88% between 50d and 300d. For both Time and OHSU, performance increased significantly with more dimensions, going up 10.39% and 45% respectively between 50d and 300d.

**4.1.3 Word weights**

**Vector normalization**

Document and query vectors are generated by taking the weighted sum of the vectors for their constituent words. Word embeddings can be of different lengths, which means that without normalizing the word vectors first, words with longer vectors will have higher weights, assuming equivalent tf-idf.

Table 4.8 and figure 4.5 show the impact of normalizing the word vectors before using them to generate document and query vectors. The impact of normalization was always negative for pretrained vectors,
Table 4.8: MAP@10 with and without word vector normalization, and percent change after normalization

(a) Pretrained embeddings

<table>
<thead>
<tr>
<th></th>
<th>Crawl</th>
<th>Wiki</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Plain</td>
<td>Norm</td>
</tr>
<tr>
<td>ADI</td>
<td>0.35</td>
<td>0.31</td>
</tr>
<tr>
<td>Time</td>
<td>0.37</td>
<td>0.34</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.14</td>
<td>0.11</td>
</tr>
</tbody>
</table>

(b) Collection and hybrid embeddings

<table>
<thead>
<tr>
<th></th>
<th>Collection</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Plain</td>
<td>Norm</td>
</tr>
<tr>
<td>ADI</td>
<td>0.28</td>
<td>0.25</td>
</tr>
<tr>
<td>Time</td>
<td>0.59</td>
<td>0.60</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.21</td>
<td>0.19</td>
</tr>
</tbody>
</table>

Figure 4.5: Change in MAP@10 after normalization
Table 4.9: Variance in lengths of vectors for words in collection documents for Wiki and collection embeddings.

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>0.15</td>
<td>23.66</td>
</tr>
<tr>
<td>Time</td>
<td>0.21</td>
<td>5.43</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.26</td>
<td>4.21</td>
</tr>
</tbody>
</table>

and with only one exception, was on average negative for hybrid and collection-only vectors. Pretrained vectors suffered the most from normalization, with an average decrease in performance of 14% for Crawl embeddings and 5% for Wiki embeddings. For collection-only and hybrid models, the average impact was usually slightly negative, at less than 0.02, except for ADI hybrid models. The impact of normalization varied somewhat with the number of training epochs - models that trained for longer tended to have lower negative impact from normalization. Overall, however, higher performance was able to be achieved when no normalization was applied.

Examining the queries showed that vector length did tend to align with word importance in both the pretrained and collection-only embeddings. Although the overall variance in lengths in the pretrained embeddings was higher, for words present in the documents, the variance in the collection-trained embeddings was higher, as is shown in table 4.9. This may indicate that some damping of the lengths may be beneficial for collection-trained models, although no such experiments were performed.

An example of the lengths of words in a Time query is shown in figure 4.6. While the lengths for low important words are similar, at around 1.5 for on, to, and the, important words like buddhist have lengths of around 3 in the Wiki embeddings but 4 or more in the Time embeddings.

Document frequency weighting

To handle query terms that were present in the embedding models but not in the document collection, four different options were tested. Either those terms were removed, or they were given a default weight (either the average or the max of the weights for terms existing in the document collection), or Wikipedia was used to compute the document
Wiki embedding  
\textit{kennedy} 2.6442 \textit{administration} 1.9365 \textit{pressure} 2.1381 
on 1.6372 \textit{ngo} 2.5320 \textit{dinh} 1.5183 \textit{diem} 3.0449 to 1.5830 \textit{stop} 2.0691 
suppressing 2.2744 \textit{the} 1.4465 \textit{buddhists} 2.6721

Collection embedding  
\textit{kennedy} 3.5954 \textit{administration} 4.6633 \textit{pressure} 5.1063 on 1.5033 \textit{ngo} 4.7964 \textit{dinh} 4.8384 \textit{diem} 4.0914 to 1.2938 \textit{stop} 4.9156 suppressing \textit{OOV} \textit{the} 1.2473 \textit{buddhists} 4.8198

Figure 4.6: Lengths of vectors in the tokens of a Time query.

Table 4.10: MAP@10 by default document-frequency weighting

<table>
<thead>
<tr>
<th></th>
<th>Zero</th>
<th>Average</th>
<th>Max</th>
<th>Wiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adi</td>
<td>\textbf{0.31}</td>
<td>0.29</td>
<td>0.29</td>
<td>0.26</td>
</tr>
<tr>
<td>Time</td>
<td>0.36</td>
<td>\textbf{0.40}</td>
<td>0.39</td>
<td>0.33</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.10</td>
<td>0.10</td>
<td>0.09</td>
<td>\textbf{0.11}</td>
</tr>
</tbody>
</table>

frequencies. This experiment was run outside of the grid search.

The results using the Wiki embeddings are shown in table 4.10. (Results for the Crawl embedding, not shown, were similar.) The results are not conclusive - what worked best differed for each collection, and Wikipedia weights worked the worst for the ADI and Time and best for OHSU.

Overall setup

Overall, certain training setups worked well across all collections: training for at least 20 epochs, using a window size of at least 20, and using some amount of subword information. ADI, due to its small size, required at least 40 training epochs, for good performance. However, for both Time and OHSU, training for 20 epochs, a window size of 20, and minimum subword size of 4 worked well, and balanced training time and performance. Therefore, this setup was chosen for further tests. Furthermore, across all collections using subword information for OOV terms and not normalizing word vectors worked well, and this setup continued to be used.
Table 4.11: MAP@10 baselines for untranslated queries

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Embed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.58</td>
<td>0.52</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.09</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Table 4.12: MAP@10 for general-purpose mapped embeddings

<table>
<thead>
<tr>
<th></th>
<th>Premapped</th>
<th>Pretrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.004</td>
<td>0.002</td>
</tr>
</tbody>
</table>

4.1.4 Bilingual baselines

For the bilingual baselines, the translated Time and OHSU queries were untranslated using Google Translate. Two baselines are reported: using the cosine similarity search engine and using the embedding search engine. Results for the untranslated queries, shown in table 4.11, were very similar to those with the original queries, reflecting the high quality of Google Translate for Swedish-English and also indicating that the quality of the translations into Swedish was acceptable.

4.1.5 Bilingual embedding mappings

General domain embeddings

Initial tests for the bilingual search engine used mappings based in part or wholly on pretrained models. Two versions of mappings were tested:

Premapped  Aligned English and Swedish embeddings provided along with the MUSE framework.

Pretrained  Pretrained Crawl models for English and Swedish mapped locally using the MUSE framework.

The results are shown in table 4.12. Overall, the performance for the premapped and pretrained models were much lower than both the monolingual case and the baseline - Time performance of 0.31 was only half as good as its cosine similarity baseline of 0.58, and OHSU’s 0.004 was just very slightly above random chance.
**Time** efforts ansträngningar of denna ambassador ambassadör henry henry cabot cabot lodge ordenshus to att get få viet vietnameserna nam’s. president president diem diem to att change ändra his hans policies policies of denna political politisk repression förtryck

**OHSU** 60. yo yo male manliga with med disseminated spreds intravascular muskelleceller coagulation; pathophysiology neurofysiologi and samt treatment behandling of denna disseminated spreds intravascular muskelleceller coagulation kalciumjoner

Figure 4.7: Nearest neighbors for each word in sample queries from Time and OHSU.

Examining the embeddings showed that the mappings for medical-domain words were of much lower quality than those for general-domain words. While the embeddings for English medical-domain terms would generally map to Swedish medical-domain terms, differentiation within the domain was low. For example, the nearest Swedish neighbor to *intravascular*, meaning within the blood cells, is *muskelleceller*, meaning muscle cells. By comparison, the terms used by Time tended to have reasonable explanation. Figure 4.7 shows a comparison between the translation of a Time query and an OHSU query.

Because of this, the rest of the experiments focused on understanding how to improve the results for the medical-domain OHSU collection by improving the mappings.

**Domain-specific mapping dictionary**

To test whether using a a domain specific dictionary was sufficient to obtain high-quality domain-specific mappings for general purpose embedding models, the pretrained vectors were mapped using domain-specific dictionaries generated from MESH translations. Two dictionaries were tested: a combination of the general-domain and domain-specific dictionaries, and a dictionary with only domain-specific terms. The results are shown in table 4.13. Although these mappings led to slightly higher performance than mappings generated using only general-purpose dictionaries, performance was still very low, at 0.0031 for the hybrid dictionary and 0.0026 for the domain-specific dictionary.
Table 4.13: MAP@10 for mappings between pretrained general-domain embeddings using domain-specific dictionaries

<table>
<thead>
<tr>
<th></th>
<th>General+Med</th>
<th>Med</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHSU</td>
<td>0.0031</td>
<td>0.0026</td>
</tr>
</tbody>
</table>

Table 4.14: Corpus size and query OOV rates for different query-language embedding training corpora

<table>
<thead>
<tr>
<th></th>
<th>Tokens</th>
<th>Unique words</th>
<th>OOV count</th>
<th>OOV rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medicine-Anatomy-4</td>
<td>7143251</td>
<td>348506</td>
<td>50</td>
<td>0.061</td>
</tr>
<tr>
<td>Medicine-3</td>
<td>3670974</td>
<td>200950</td>
<td>50</td>
<td>0.061</td>
</tr>
<tr>
<td>Medicine-4</td>
<td>6039142</td>
<td>303784</td>
<td>50</td>
<td>0.061</td>
</tr>
<tr>
<td>Medicine-5</td>
<td>9413348</td>
<td>430049</td>
<td>48</td>
<td>0.059</td>
</tr>
</tbody>
</table>

**Domain-specific corpus for query-language embeddings**

Due to the poor performance using general-purpose Swedish embeddings, the rest of the tests for tuning hyperparameters were performed using Swedish embeddings trained on domain-specific data.

Swedish training data was sourced from Wikipedia, by taking all pages in a given category and all its subcategories up to some depth. The Medicine (medicine) category was used, and depths of 3, 4, and 5 were tested. Increasing the depth increased the amount of training text, although it lead to an increased number of irrelevant documents; for example, the category Blinda musiker (blind musicians) is at a depth 4 from the category Medicine.

Table 4.14 shows the number of tokens, vocabulary size, and query out-of-vocabulary rates for the different category depths. Although increasing the depth added many more tokens and unique words, the OOV rate stayed essentially the same, decreasing with only two words between depth 3 and depth 5. This indicates that any increases in performance with more training data were not due decreases in OOV rates.

On average, keeping all hyperparameters equal but increasing the depth from 3 to 5 led to a MAP@10 increase of 0.004.
Table 4.15: MAP@10 where subword embeddings were used for OOV or all terms

<table>
<thead>
<tr>
<th></th>
<th>None</th>
<th>English</th>
<th>Swedish</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOV</td>
<td>0.055</td>
<td>0.053</td>
<td>0.055</td>
<td>0.054</td>
</tr>
<tr>
<td>All</td>
<td>0.055</td>
<td>0.038</td>
<td>0.011</td>
<td>0.017</td>
</tr>
</tbody>
</table>

Hyperparameters for query-language embeddings

A grid search was performed on the minimum subword size and window size hyperparameters when training the Swedish embeddings, with the number of epochs held constant at 20. Each Swedish embedding model was then mapped to two different high performing English models: one collection-only and one hybrid.

The results were in line with the findings for the English embeddings, namely that a subword size of 4 and larger window sizes lead to better performance. A minimum subword size of 4 lead to an average increase of 0.001 over a minimum subword size of 3, and 0.005 over not using subword information. Increasing the window size from 5 to 40 improved MAP@10 by 0.013, with most of the improvement gain occurring between 5 and 20.

Keeping hyperparameters constant, mapping to the collection-only embedding had an average MAP@10 improvement of 0.002, though the change varied between a 0.017 increase and 0.015 decrease in performance. Interestingly, the Swedish models trained on more data benefited more from being mapped to the collection-only OHSU models.

Subword information

For one pair of mapped embeddings, the subword embeddings were recovered. These subword embeddings were then used to generate word embeddings, either for all words or for OOV words. The results are shown in Table 4.15.

Generally, models using subword embeddings performed very slightly worse if the subword embeddings were used for only OOV terms, and significantly worse if the subword embeddings were used for all terms. Examining the vectors generated for OOV terms showed that the problem was not that the vectors for the OOV words were low quality in
Table 4.16: MAP@10 for different term weighting schemes

<table>
<thead>
<tr>
<th></th>
<th>None</th>
<th>Stopword</th>
<th>Idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSHU</td>
<td>0.055</td>
<td>0.055</td>
<td>0.040</td>
</tr>
</tbody>
</table>

their original language, but rather, that their translations tended to be of low quality. For example, the OOV term *hormonersättningstherapi*, meaning hormone replacement therapy, mapped most closely in Swedish to the in-vocabulary term *hormonersättningsbehandling*, or hormone replacement therapy. However, its closest term in English was *chirurgical*, meaning surgical - in-domain, but with very different meaning. While OOV terms tended to get high-quality vectors that were close to similar in-vocabulary terms, in-vocabulary vectors tended to get worse when using the subword embeddings. This is not surprising, since many of the words sharing n-grams with each other may not have any meaning in common.

**Word weights**

Because query terms are not present in the document collections, they cannot directly be weighted by idf. Therefore, document frequency information was computed using the domain-specific text corpora downloaded from Wikipedia (depth 5). Two experiments using document frequency were run: stopword removal based on high df-values, and query term weighting based on Wikipedia idf. For stopword removal, the cutoff was any word appearing in more than 40% of all documents. The results are shown in table 4.16.

Using the Wikipedia weights for stopwords had no impact. Using it for idf decreased the MAP@10 by 0.015. This is inline with the monolingual tests, which for 2 out of the 3 datasets (though not for OHSU) showed decreased performance when using Wikipedia idf weights for OOV terms.

### 4.2 Test data

Four datasets were used for testing: the computing-domain CACM, the electronics-domain NPL, and the medical-domain Medline and test portion of OHSU. These datasets are described in detail in section 3.1.1.
Table 4.17: MAP@10 for test datasets. Collection-only and hybrid results shown as average of 3 runs, with range of results in parentheses.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Crawl</th>
<th>Collection-only</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACM</td>
<td>0.14</td>
<td>0.11</td>
<td><strong>0.21</strong></td>
<td>0.17</td>
</tr>
<tr>
<td>NPL</td>
<td>0.07</td>
<td>0.08</td>
<td>0.10</td>
<td><strong>0.13</strong></td>
</tr>
<tr>
<td>Medline</td>
<td>0.23</td>
<td>0.22</td>
<td><strong>0.29</strong></td>
<td><strong>0.29</strong></td>
</tr>
<tr>
<td>OHSU</td>
<td>0.029</td>
<td>0.051</td>
<td>0.054</td>
<td><strong>0.057</strong></td>
</tr>
</tbody>
</table>

Table 4.18: Difference in MAP@10 for hybrid embeddings as compared to collection-only embeddings, and OOV rates for queries with respect to document collection. Tokens that appear only once in the document collection are not included in the vocabulary.

<table>
<thead>
<tr>
<th></th>
<th>Difference</th>
<th>OOV rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACM</td>
<td>-0.03</td>
<td>0.17</td>
</tr>
<tr>
<td>NPL</td>
<td>+0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>Medline</td>
<td>+0.002</td>
<td>0.10</td>
</tr>
<tr>
<td>OHSU</td>
<td>+0.004</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 4.17 shows the baseline results for these datasets, as well as results using the Crawl pretrained vectors and collection-only vectors. Overall, the results were similar to those of the development datasets - depending on the dataset the baseline or pretrained embeddings might have higher performance, but the highest MAP@10 was always achieved using collection-only or hybrid embeddings.

Whether the hybrid embeddings or collection-only embeddings did better varied among the collections. CACM saw a 0.03 drop using hybrid embeddings instead of collection-only, NPL saw a 0.03 increase, and Medline and OHSU stayed the same. Although it seems expected that hybrid models would do better for collections where the query OOV rate with regard to the documents was high, this was not the case, as can be seen in table 4.18.

The two medical-domain collections, Medline and OHSU (test), were also tested using the pretrained medical-domain Pubmed embeddings, as well as the embeddings trained on OHSU development data. The results are shown in table 4.19. For both data sets, the Pubmed embeddings lead to higher MAP@10 than the baselines, by 0.01. For OHSU,
Table 4.19: MAP@10 for medical test datasets, using pretrained Pubmed embeddings and embeddings trained on OHSU development documents, and change in MAP@10 using OHSU development embeddings as compared to using embedding trained on own dataset.

<table>
<thead>
<tr>
<th></th>
<th>Pubmed</th>
<th>OHSU (dev)</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medline</td>
<td>0.24</td>
<td>0.26</td>
<td>-0.03</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.044</td>
<td>0.055</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.20: MAP@10 for bilingual search engine for Medline and Pubmed.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Baseline embed</th>
<th>Bilingual mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medline</td>
<td>0.18</td>
<td>0.25</td>
<td>0.047</td>
</tr>
<tr>
<td>OHSU</td>
<td>0.029</td>
<td>0.057</td>
<td>0.023</td>
</tr>
</tbody>
</table>

However, the Pubmed embeddings performed worse than the Crawl embeddings, by 0.01.

Both datasets also had positive results using the OHSU development embeddings. For OHSU, the OHSU development embeddings worked just as well as the embeddings trained on the test document collection. For Medline, the OHSU development embeddings performed worse than the collection trained embedding, by 0.03, but still better than the baseline and the Crawl embeddings, by 0.03 and 0.04 respectively.

Table 4.20 shows the results for the bilingual search engine. For OHSU, the results for the untranslated queries were essentially the same as for the original queries. For Medline, the untranslated queries performed 0.05 worse than the original queries in the cosine similarity search engine, and 0.04 worse in the embedding search engine. Note that for Medline the bilingual mappings were generated using hybrid embeddings trained on Medline, due to the small size of the Medline-only embedding model.
4.3 Analysis and examples

4.3.1 Monolingual

It is clear that using embedding models will improve recall, because relevant documents that don’t contain any of the words in the query will be retrieved. However, embedding models also improve ranking performance, since every word in the document contributes to whether it is relevant to the query or not, not only overlapping words.

This is illustrated by an example from the Time collection:

**Query**  spain’s relaxation of controls over some of its african territories .

**Top result (baseline)**  <article about Spain> spain more news, more money spaniards who tuned in on news broadcasts last week got the surprise of a quarter-century . since francisco franco installed himself as spain’s dictator in 1938, every newscast had unfailingly ended with a ponderous salute to his falangist party and a martial rendition of the falangist anthem ...

**Top result (embedding)**  <article about Portugal giving up African colonies, mentioning Spain’s policy> portugal too late in the day portugal’s ascetic dictator antonio de oliveira salazar made one of his rare tv appearances last week (on film) to answer african demands that portugal abandon its colonies ... spain’s francisco franco, was bending slightly more with the winds, announced plans to grant a measure of autonomy to spanish guinea

The baseline top ranked (irrelevant) article includes the word Spain 8 times, and is therefore ranked highly by the baseline search engine, despite the rest of the article being irrelevant and never mentioning Africa or Spanish colonies. The embedding top ranked (relevant) article uses the word Spain only once, African only twice, and territories only once, but the rest of the article contains many words that are related to the three concepts, such as the names of countries in Africa. In an embedding model, these can contribute to moving the document vector closer to the query.

Because all words can contribute to relevance, the embedding model does not show the same level of preference for short documents that mention only the words of the query. This affected the OHSU collection
most, since many of the documents in the collection consisted only of a headline.

**Query**  55 yo female, postmenopausal does estrogen replacement therapy cause breast cancer

**First result (both)**  postmenopausal estrogen use and heart disease

**Second result (both)**  postmenopausal estrogen and cardiovascular disease

**Third result (baseline)**  breast cancer screening

**Third result (embedding)**  sex steroids and cancer, hormones, particularly estrogens, have been suspected for many years of being carcinogens. Retrospective studies from the mid-1970s indicate that unopposed estrogen replacement therapy increases the risk for endometrial cancer... there is increasing evidence that progestogen added to estrogen replacement may decrease the risk for carcinoma of the breast in some women.

While the first two results were the same, and irrelevant, in both search engines, the baseline search engine continued to return such results, while the embedding search engine returned longer documents that contained many related words such as *progestogen* and *carcinogens*.

### 4.3.2 Bilingual

Using mapped embeddings trained only on in-domain data had a large impact on the performance of the bilingual search engine, even if performance was not as good as when using Google translate.

The reason for the improvement can be illustrated by example.

**Swedish query**  patient med dysuri. urinvägsinfektion kriterier för behandling och intagning

**English query**  patient with dysuria. urinary tract infection criteria for treatment and admission

**Translations (premapped)**  *patient patient med with dysuri OOV urinvägsinfektion infections kriterier criteria för for behandling treatment och and intagning hospitalization*
Translations (domain-specific)  patient  patient med  the dysuri dysuria  urinvägsinfektion  urosepsis kriterier criteria för the behandling  treatment och the intagning  paraprofessional

The pretrained embeddings often had better mappings for more general words, such as intagning, which it correctly mapped to hospitalization, compared to the domain specific embedding’s translation to paraprofessional. However, for more important, specific words, the domain-specific translations were better. One of the most important words in the query, dysuri, was not present at all in the general embeddings. The other keyword, urinvägsinfektion, meaning urinary tract infection, was translated to the general infections. While this is more correct than the domain-specific translation to urosepsis, which is a complication of urinary tract infection, it is a less useful for finding relevant documents. Indeed, the search engine using domain-specific embeddings found 3 relevant documents, including two in the top 2 results, while the one using general embeddings found none.

While the domain-specific mappings tended to be of higher quality for many in-domain terms, as the example above shows, there are still cases where in-domain terms had better mappings in the general embeddings. This is likely a sign that not enough data was used to train the domain-specific mappings.
Chapter 5

Discussion and conclusions

This final chapter discusses the results of the project in a broader context. Section 5.1 reviews the results, including their potential usefulness and possible reasons for why the systems behaved the way they did. Section 5.2 proposes some areas for future research.

5.1 Discussion of results

5.1.1 Monolingual information retrieval

Although using word embeddings was useful for all three development collections, the ideal model configurations often varied between them. This reflects the fact that the collections are each unique in some way: ADI is very small, Time is fairly general-purpose, and OHSU is large and domain-specific.

Pretrained models vary widely, both in their general quality as well as their suitability for a particular domain. All collections performed better with the Crawl model, which is trained on more data and is likely more general-purpose than the Wiki model. For OHSU, the best performance with a pretrained model was using the in-domain Pubmed model; Time especially did more poorly using this model, despite the fact that it was trained not only on medical texts but also Wikipedia. The large variance in performance when using different pretrained models highlights the importance of picking a high-quality and domain-appropriate pretrained model, even when no further tuning will be done.

Both Time and OHSU did better with hybrid models than with
pretrained models, and better still with collection only models. ADI, on
the other hand, showed the opposite effect, and did best with pretrained
models and worst with collection-only models. For ADI, this is likely
explained by the small collection size - there is not enough training data
to create high-quality, generalizable vectors, and the vocabulary size
is limited and does not contain enough of the query terms. OHSU, on
the other hand, showed the largest improvement when incorporating
collection data. This is likely because the general-purpose vectors cannot
sufficiently distinguish between different in-domain terms.

5.1.2 Cross-language information retrieval

Modern machine translation works extremely well, especially between
two high-resource languages such as English and Swedish, and where
the output of translation does not necessarily have to be grammatically
correct and natural, such as with a search query. This was reflected
by the high performance in the baseline when using automatically
translated search queries.

However, machine translation systems require large amounts of
parallel data to train, and may therefore not be of available and of high
quality for all language pairs and domains. Word embedding mappings
require less data, and especially less parallel data. The general-domain
Time dataset achieved a MAP@10 of 0.31 using off-the-shelf mapped
embeddings. Although this was much lower than in the monolingual
baseline, it is still high enough to be useful.

For the domain-specific OHSU collection, pretrained embeddings re-
sulted in extremely low performance. However, using a domain-specific
dictionary and domain-specific training data improved performance
significantly for it, even if overall results were still quite low. The same
datasets were also useful for the Medline collection.

One reason for the poor performance in the bilingual search on the
medical-domain collection was that many of the high-information terms
were out-of-vocabulary. Although subword information is generally
useful for OOV terms, incorporating it in the mapped embeddings had
little impact, despite the fact that some of the OOV term appeared in
different forms (for example plural or with different spelling) or were
compounds containing in-vocabulary terms. Because the goal of this
project was to examine unsupervised and language-agnostic methods,
minimal preprocessing was done on the text, including no stemming or
compound splitting. However, in a real-world application, language-specific preprocessing will likely prove useful.

5.2 Future work

There are several areas for further development, both in understanding the results and in improving them.

Although the baseline used in this project was a tf-idf cosine similarity search engine, the embedding search engine has more in common with other methods that do not rely as heavily on the actual words present in the document, such as LSA or search engines that do query expansion. Comparing results to such systems would be interesting.

When training embeddings, larger window sizes led to higher performance, but it is unclear why this is the case. While one explanation is that embeddings trained with larger window sizes have higher topical, rather than typical, similarity, other explanations may be that larger window sizes allowed the embeddings to encode entire documents rather than individual words, or that larger window sizes meant that more training data was available per word. To understand the causes, the experiments could be repeated with larger collections containing longer documents, or intrinsic evaluation could be used to determine whether the larger-window embedding models display higher topical, rather than typical, similarity.

Domain-specific bilingual embeddings were often, but not always, better than their general-purpose counterparts. Significantly more training data would probably be required to generate embeddings that always perform better. One way to improve this might be to train domain-specific embeddings using pretrained embeddings as starting weights, as was done with the monolingual hybrid models. Other methods for training high quality domain-specific vectors using low amounts of in-domain data are also worth investigating.

While high quality monolingual subword embeddings were able to be reconstructed from embedding models without subword information, their cross-language mappings tended to be poor. Incorporating subword information directly while training mappings might improve the mappings for all words, just as in monolingual training.
Bibliography


