ALEXANDRU-IOAN CUZA UNIVERSITY IASI

FACULTY OF COMPUTER SCIENCE

BACHELOR'S THESIS

Medical Domain Word Embeddings: Training and Evaluation

proposed by

Florin Eugen Rotaru

Session: July, 2024

Scientific Advisor

Lect. Dr. Eugen Nicolae Croitoru

ALEXANDRU-IOAN CUZA UNIVERISTY IASI

FACULTY OF COMPUTER SCIENCE

Medical Domain Word Embeddings: Training and Evaluation

Florin Eugen Rotaru

Session: July, 2024

Scientific advisor

Lect. Dr. Eugen Nicolae Croitoru

Contents

Abstract

Word embeddings (WE) have been used extensively in the last period for representing text in the context of NLP and Machine Learning tasks. This project aims to study the potential improvements of WE trained on specific data compared to the general pretrained ones available. We have trained medical domain WE on 72.6M words worth of medical literature with a vocabulary of the most frequent 150K words. The model used was the popular Word2Vec Skip-Gram with negative sampling. We trained two versions, one with window size 2 and 4 negative samples and another with window size with 4 and 7 negative samples The empirical tests have been run using the two embedding sets trained by us and two standard general purpose WE: GloVe on 6B tokens and word2vec on 100B tokens.

For analyzing the outcome, we have opted for one visual analysis and some quantitative evaluations. Visually, we represented the embeddings of 20 drugs, symptoms and disorders in a 2-D space using PCA and inspected the word positionings. The word clusters formed and the proximity of similar words such as ”schizophrenia”, ”parkinson”, ”alzheimer” prove that our WE capture some semantic properties better than the general ones.

Quantitatively, we tested the WE on 3 different datasets of word pairs annotated by physicians with similarity scores and also on two information extraction tasks: detection of human/veterinary medicine type of text and smoker status detection based on discharge reports. Results show that our embeddings perform closer to human reasoning on 2 out of the three datasets, achieving a 0.517 correlation coefficient. On the classification tasks, our WE managed to outperform GloVe embeddings, with 0.968 accuracy and F1-score for task 1 and 0.504 accuracy and 0.51 F1-score for task 2 (vs 0.947 accuracy, 0.947 F1-score and 0.445 accuracy, 0.43 F1-score respectively), proving to handle tasks with small amount of training data better. Nevertheless, Word2Vec still performs best, scoring 0.986 accuracy and F1-score on task 1 and 0.58 accuracy, 0.53 F1-

score on task 2. Another finding is that our embeddings possess non-linear patterns which can be leveraged by non-linear classifiers (we used an SVM with 2 different kernels).

We conclude that our embeddings have managed to faithfully reflect the relatedness of medical domain concepts, in some instances better than the general WE. We have managed to ourperform one set of general WE on a classification task. Even so, more work is needed in order to obtain medical WE that outperform general WE on tasks such as IE (using more data, parameter fine-tuning), provided general WE can be outperformed at all by domain specific embeddings.

Chapter 1

Introduction

Natural Language Processing (NLP), which aims to model the human language with the aid of machine learning and statistical models, has been rapidly evolving in the past decade, finding its applications in almost all fields of human activity. Some examples are finance - analyzing market trends by processing vast amounts of financial news, entertainment and media - with recommendation systems, subtitles generation, summaries, legal - automatic contract review, assisting in legal research, etc.

Word embeddings are a popular way of representing text data in order to feed it to NLP related models and usually consist of word-level vectorization. Since the word embeddings are obtained through training on raw text, their nature depends on the training data. Depending on the data level of generality, word embeddings can be general or domain specific.

Using pretrained general word embeddings in a medical-domain specific task may have drawbacks. For example, take the words ”cell” and ”culture”, we can notice their semantic similarity with regards to the biomedical field. In the general sense, the word ”cell” is not necessarily related to medicine, it can refer to the smallest structural and functional unit of an organism or a mobile phone. Even in computer science, the term ”cell” is widely used.

Consequently, we are interested in generating pretrained domain specifc word embeddings for the medical field. That is, instead of using general literature like news, wikipedia, magazines, etc., we would like to use medical literature as dataset. We aim to obtain domain specific word that would perform competitively compared to available pretrained word embeddings trained on general text, with the constraints of having a reasonable running time (considering our limited computational resources)

and using significantly less training data - for example, using up to 80% less data. The structure of this study is as follows:

Chapter 1 consists of an overview on NLP applications in medicine and a thorough review of word representations in machine learning models, their use cases, advantages and potential drawback. We focuse our study especially on two state-of-the art methods of training word embeddings, namely `GloVe` and `word2vec` and attempted to deeply understand their particularities.

Chapter 2 describes our own implementation of a "seminal" method for training word embeddings using a shallow neural network-like mod. We present the dataset used, the preprocessing techinques, tools, frameworks and the model configuration. We also describe the training process and discuss the issues we faced along the way.

Chapter 3 first presents the metrics deemed most relevant in the context of word embeddings performance evaluation. Then, using correlation coefficient tests and domain relevant information extraction tasks, we evaluate the performance of our word embeddings with respect to standard pretrained word embeddings (`GloVe`, `word2vec`) and discuss the results.

Chapter 2

State of the Art Word Embeddings

2.1 NLP in Medicine

Healthcare is major field for NLP applications. This is due to the fact that a lot of written information is being generated in the form of electronic health records (EHR), clinical notes, scientific literature and patient feedback. All this data can be easily harvested by data specialists in order to come up tools that would streamline patient care and medical research. Some common NLP biomedical tasks currently undergoing intense research are: *Information Extraction (IE) tasks* [1] - acquiring clinical information from texts. By information we understand entities (disease, symptoms, etc.), relations (temporal relations or between medical entities, such as drugs and side effects, interaction between various drugs, etc.) or events, such as surgeries. Some examples of IE tasks include automatic identification of patient smoking-status [2], identification of medications, their dosage and reasons for administration, etc. Frequent ML models used in IE are Support Vector Machine (SVM) and Logistic Regression. *Information Retrieval (IE) tasks* - simply put, it represents a framework that allows the user (a physician in our case), to get a list of relevant documents (such as researches) in response to a specific query. As an example, a 2016 research challenge [3] consisted of developing a system that retrieves relevant biomedical articles that could answer questions about diagnoses, tests and treatments, such as "What treatment is appropriate for this patient?". For each of the questions, a real EHR note would also be given as input. *Question Answering tasks* - instead of providing a set of documents as answer to a query, it is a lot more practical to automatically analyze their contents and generate specific answers. Such systems can be achieved by extending a document retrieval model with

an answer extraction layer and a summarization technique, [4]. The answer extraction layers selects the relevant phrases from the retrieved documents using patterns for each class of answer to the question (e.g definitions of terms). The summarization layer clusters the sentences in order to select the most representative sentence from each cluster. The selected sentences are then ordered by similarity in order to form a coherent answer. Other examples of more specific medical tasks [5] are chemical-disease relation, drug-drug interaction, medical abbreviation disambiguation, medical synonym extraction, etc.

2.2 Text Representation

One of the first challenges for NLP taks mentioned above is choosing a way to represent the text such that machine learning models can make sense of it. That is, providing the text as numeric tensors (n-dimensional vectors). The first step is performing *tokenization* - splitting text into units. The units can be groups of words, words, or even individual characters. In this project we choose to tokenize at word level. The next step is converting the tokens to numerical vectors, for example:

Original Text:	”Coffee	enhances	cognitive	function.”
Standardized Text:	”coffee	enhances	cognitive	function”
Tokens:	”coffee”,	”enhances”,	”cognitive”,	”function”
Indexing:	1,	2,	3,	4
Vectorization:	$[v_1^1, v_2^1, ..., v_n^1]$	$[v_1^2, v_2^2, ..., v_n^2]$	$[v_1^3, v_2^3, ..., v_n^3]$	$[v_1^4, v_2^4, ..., v_n^4]$

There are several approaches to vectorizing the tokens.[6] The most trivial one is using **one-hot encodings**, which represents each word as a sparse vector with the value 1 corresponding to its index in the vocabulary. For example, if ”coffee” is word number 1 and ”function” is word number 4, their one-hot representations will be:

$$\mathbf{v_{coffee}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{v_{function}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

In many instances this approach can prove to be practical, since no additional computation is required to obtain the word representations. All we need is knowing the

vocabulary (i.e., all used tokens). The major weakness of this approach is the independence and uniqueness of each word representation, meaning that they hold no relationship or connection between each other (the inner product between any two such vectors is 0). Another drawback is the high dimensionality of the vectors, equal to the size of the vocabulary. The lack of semantic information in a word representation makes it hard for the model using them to generalize.

A way to add some information to this representation is counting how many times a token appears in the text corpus. Replacing the 1 on the vector index position with the word's frequency encodes more information because it helps identify terms that truly characterize the content of documents, the ones which define the overall semantic meaning. This approach is called **term frequency (TF) representation**.

Even so, there will always be words that occur very often in any document, such as articles and prepositions, that don't bring any value as features. This issue is remediated with a normalization scheme called term **frequency - inverse document frequency (TF-IDF) normalization**. [7] This approach decreases the importance of a word when that word is common to multiple documents in the overall set of documents. One way to calculate the TF-IDF value of a word is

$$\text{TF-IDF}(t) = \text{TF}(t) * \text{IDF}(t)$$

$$\text{IDF}(t) = \log \frac{|\text{D}|}{\text{DF}(t)}$$

where $\text{DF}(t)$ is the number of documents in which t appears and $\text{TF}(t)$ is the total term frequency of t. This assumes that the term appears in the corpus. The approach still has issues: let's suppose that a term appears in almost all documents. Then, $|D| \approx \text{DF}(t)$, which means $\text{TF-IDF}(t) \approx 0$. This would happen even if the term occurred everywhere 1-2 times and in one particular document 100 times. This detail is not captured by TF-IDF as it only counts the frequency in the targeted document.

There are many other techniques to represent texts that rely on counting (e.g co-occurrence matrix, etc.), but since the above described NLP tasks rely heavily on medical semantics, we will opt for something else. [8]. We are interested in choosing a representation that would help the vectors retain the semantic meaning of their corresponding token. This suggests that the representation has to be trained against some text data in order to memorize and embed the co-occurrences in the vector representation. What we are looking for in this case are word embeddings - a dense vector of N features $(f1, f2, ..., f_N)$ where each feature describes an attribute of the word within the

vector space. As a consequence, the vector becomes a mapping of the human language into a geometric space, with values like euclidean distance reflecting the *relatedness* of two words. [6]

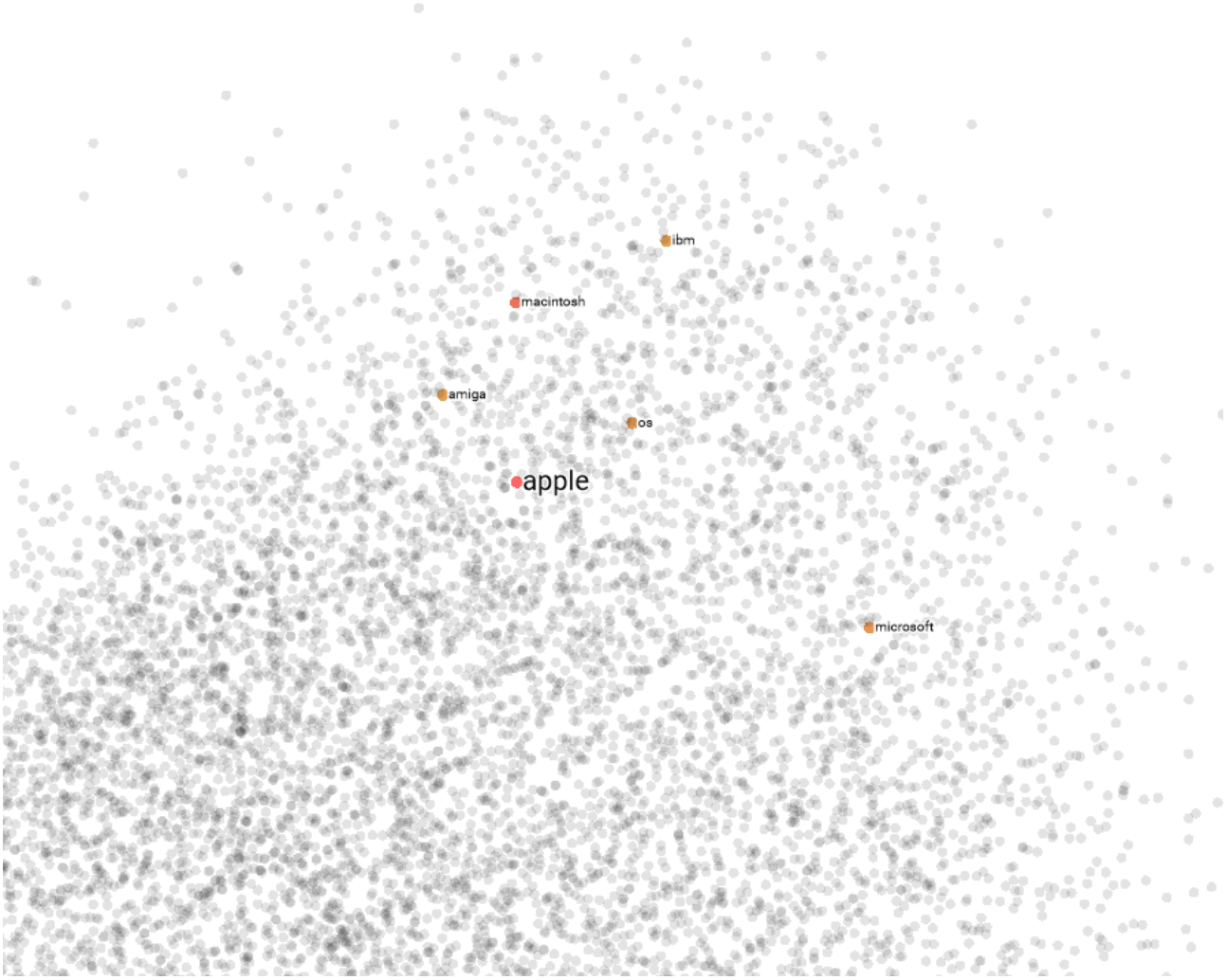


Figure 2.1: A projection of 200-dimensional Word2Vec word embeddings with Principal Component Analysis showing 5 closest words to the word "apple" in the original space. (https://projector.tensorflow.org/)

Word embeddings are obtained either by learning together with another task, such as document classification, where the embeddings are tuned jointly with the weights of the neural network via backpropagation, or using a dedicated machine learning algorithm designed to parse large volumes of text and learn general but good feature representations that can perform decently on all tasks (GloVe, Word2Vec, etc). The drawback of the first approach is the random initialization of the embeddings, which in case of little training data available makes it very ineffective.

2.3 Introduction to Word Embeddings

Word Embeddings combine semantics and connotations of words in dense vector (nonzero elements) of floating point values. Formally, we can define them as a mapping $V \mapsto \mathbb{R}^D : word \mapsto v_{word}$ of a $word$ from a vocabulary V to a real-valued vector v in a vector space of dimension D. One widely used method to evaluate the similarity (relatedness) of two words using their vector representation is the *cosine similarity* metric:

$$\text{cosine similarity}(v_{w_1}, v_{w_2}) = \frac{v_{w_1} \cdot v_{w_2}}{\|v_{w_1}\| \|v_{w_2}\|} \tag{2.1}$$

The cosine similarity belongs to the interval $[-1, 1]$. A value close to 1 indicates a very strong similarity. A value of 0 indicates unrelatedness (words that do not appear in similar contexts). A value close to -1 indicates an opposition in meaning or concept.

The linear properties of word embeddings allow for more complex similarity tasks than just grouping words by similarity, specifically using vector algebraic operations for semantic queries and analogies [9]. For example, we can retrieve the capital of France by considering the following question: "What is to France as Berlin is to Germany?", to find the answer we simply compute $X = v_{France} + v_{Germany} - v_{Berlin}$. If we search in the vector space for the word closest to X by $cosine\ similarity$, provided the vectors are well trained, we may find the correct answer $(Paris)$. Conceptually, we may interpret $(v_{Germany} - v_{Beriln})$ as the transformation "from state to capital". In the following two sections we will go over two major state of the art embedding schemes. The first one is the Word2Vec[1] algorithm, introduced by Tomas Mikolov at Google in 2013. The second one is Global Vectors (GloVe)[2], developed by Stanford researchers in 2014. Both of them come with a database of precomputed embeddings available online.

2.4 Word2Vec

Word2Vec was introduced by Mikolov et al. in 2013 [9]. It functions as an unsupervised learning task that takes a large corpus of text as input and captures semantic relationships between words based on their contexts. It operates as a shallow, two-layer neural network and employs self-supervised learning through fake prediction

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

[2] https://nlp.stanford.edu/projects/glove/

tasks. Specifically, Word2Vec parses text sequences by using some parts of the text as inputs and setting other parts as targets to predict. This approach allows it to learn vector representations of words by predicting either the context words from a given target word (Skip-gram) or the target word from its surrounding context (CBOW). After training, the embeddings essentially become the weights of the neural network's first layer. These weights, now fine-tuned, represent each word as a point in a high-dimensional space. These word vectors capture the meanings and relationships between words, allowing us to compare and manipulate them based on their contexts in meaningful ways.

2.4.1 Continuous Bag of Words (CBOW)

Bags of words are a way to represent text sequences as unordered sets of unique tokens. As a result, the general structure of the sentence is lost which leads to losing the general semantic structure. This limitation offers simplicity of use and interpretability in return.

As the CBOW model aims to predict the word (target) given its surrounding context, it relies on bags of words to represent the given context. The term *continuous* refers to generating contexts and targets using a rolling window of fixed size k that slides across the sentence.

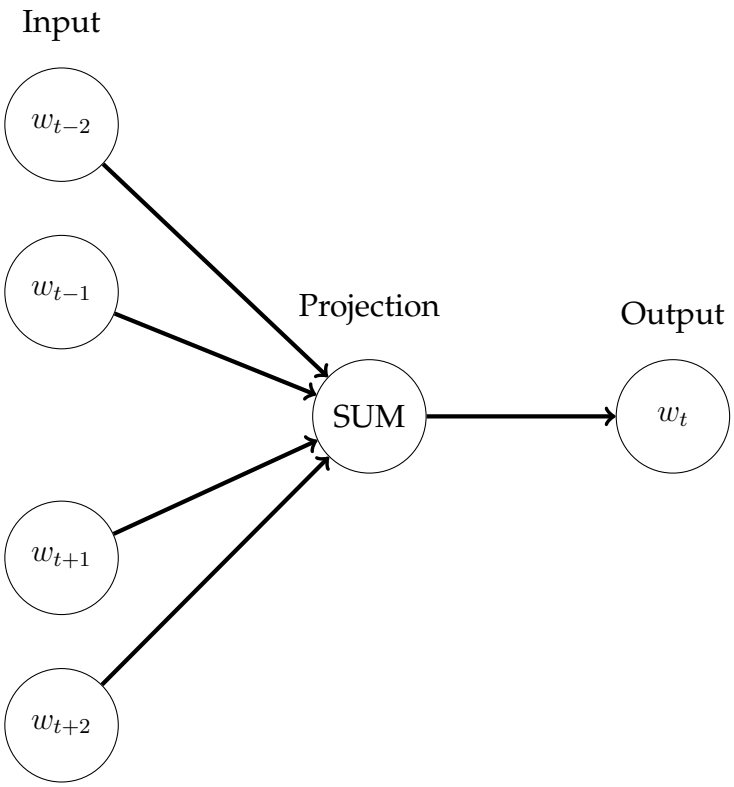


Figure 2.2: CBOW Architecture

The size k of the rolling window means we consider k history words and k future words, but we don't actually take into account their order. We present the following example:

Text Sequence: "neurons transmit signals linking thoughts actions and sensations"

Window Size = 2

context BOW (input)	target
{"transmit", "signals"}	neurons
{"neurons", "signals", "linking"}	transmit
{"neurons", "transmit", "linking", "thoughts"}	signals
{"transmit", "signals", "thoughts", "actions"}	linking
...	...
{"actions", "and"}	sensations

Table 2.1: Training samples generation with a rolling window with CBOW.

The model architecture uses two weight matrices. The first one, also called the *projection matrix*, denoted as W_{in}, holds the actual embeddings of the words - the row

i represents the vector representation for the word i in the vocabulary. This means that the size of the matrix will be $V \times N$, where V is the vocabulary size and N the embedding dimensionality. Suppose we have the context words $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$, represented as one-hot encodings (a sparse vector with 1 on their corresponding vocabulary index). In order to contract them into a single vector we first sum them and obtain the multi-hot encoding of the context $c = w_{t-2} + w_{t-1} + w_{t+1} + w_{t+2}$. Multiplying the multi-hot encoding with the projecting layer gives the sum of the context word embeddings:

$$\mathbf{v}_{in} = c^T \cdot W_{in}$$

or we can write

$$\mathbf{v}_{in} = e_{t-2} + e_{t-1} + e_{t+1} + e_{t+2}$$

where e_i is the embedding of the word i. Once we have projected the context into the first layer, we can proceed to obtaining a numerical score for the predicted word throughout the entire vocabulary. To do this we use the second weight matrix, called the output layer (having the same dimensions, but reversed). Multiplying it with the v_{in} we obtain an output vector of scores for each word.

$$\mathbf{z} = \mathbf{v}_{in}^T \cdot W_{out}$$

we may also add a bias, as it is a very popular practice in machine learning:

$$\mathbf{z} = \mathbf{v}_{in}^T \cdot W_{out} + b$$

The elements of $\mathbf{z}$ are also called *logits*, since they represent raw unnormalized results of the linear transformation. Our final output will be the normalization of $\mathbf{z}$ using the softmax function. This is done in order to convert the raw logits to a probability distribution of predicted words. In other words, we would like to obtain a probability distribution $P(X = w_i) = p_i$, with p_i being the probability of the i-th class being the correct input. Naturally, we want to maximize the p_k with k being the target word, for all training samples. The softmax activation function is as follows:

$$P(w_j|\vec{z}) = \sigma(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{V} e^{z_i}} \tag{2.2}$$

Once we have converted the output to a probability distribution, the next step is to analyze the loss with regards to the target word. To achieve this, we will use the *Cross-Entropy* loss function, which is a suitable way to quantify the predicted probability

of the target word given that softmax is used. Suppose we have the target word w_j represented as a one-hot encoding y_j. The cross entropy penalizes low probabilities for the target word. For one sample, the loss is computed in the following way:

$$CE = -\sum_{i=1}^{N} y_i \cdot \log(p_i) \tag{2.3}$$

Since we represent the target as a one-hot encoding, all members of the sum except the j-th will be zero, therefore

$$CE(j) = -\log(p_j)$$

Using the softmax formula 2.2, we have

$$CE(j) = -z_j + \log \sum_{i=1}^{V} e^{z_i} \tag{2.4}$$

2.4.2 Continuous Skip-gram

The second variation of the algorithm described in the paper by (Mikolov et al. 2013) [9] uses skip-grams. The term *skip-gram* refers to the concept of *n-grams*, which is a way to represent a text as continuous sequences of n words.

```
coffee has shown to enhance cognitive function
```

The 4-grams are:

- `"coffee has shown to"`
- `"has shown to enhance"`
- `"shown to enhance cognitive"`
- `"to enhance cognitive function"`

This time, instead of predicting the ”middle” word given a context, we will do exactly the opposite: given a word we will try to predict the context. The rationale behind this idea is that similar words appear in similar context and thus they should have similar embeddings. This fake classification task is achieved using the same model as architecture as CBOW.

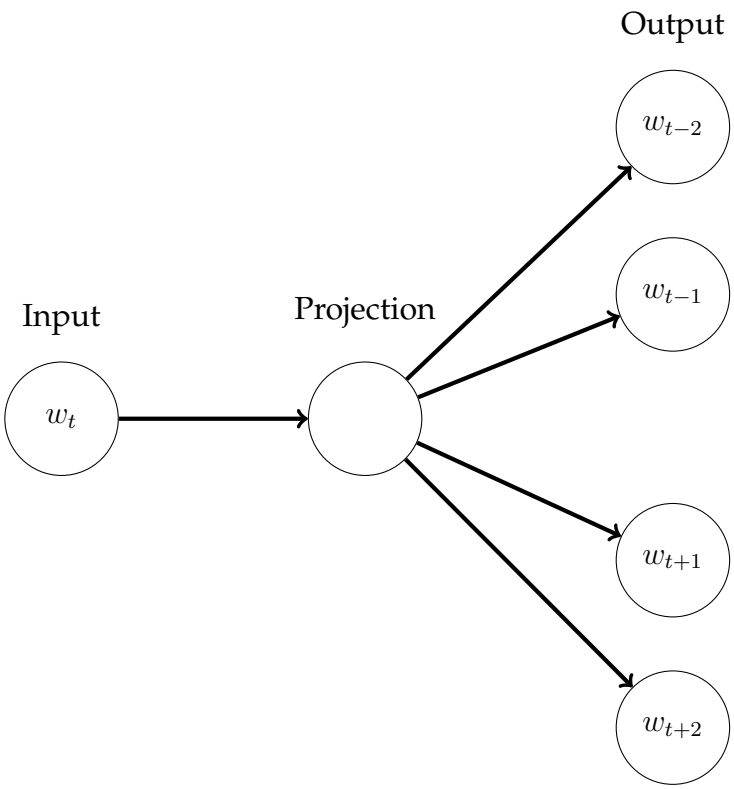


Figure 2.3: Skip-Gram Architecture

The same principle of rolling window of size k is used, with k history and future words. Only this time, the target word w_t will be fed as input as a one hot encoding. The expected output, on the other hand, will also be a one hot encoding. This means that for $k = 2$, we will have 4 context words and each of them will be considered as a separate classification:

`"coffee has shown to enhance"`

Classifications:

- `shown -> coffee`
- `shown -> has`
- `shown -> to`
- `shown -> enhance`

The workflow is identical to CBOW: the one-hot encoding of `"shown"` is projected into an embedding layer, then passed through a hidden layer in order to obtain the logits, which are the raw scores. Then, using the softmax formula 2.2 we obtain the probability for the specific true word chosen for classification. The same cross entropy

loss function is used 2.4, as you can see, the number of possible classes is equal to the total number of words in the vocabulary.

2.4.3 Model Comparison & Results

The first thing we notice is that because for each training sample, the skip-gram model performs $2k$ classifications, it is more computationally expensive than CBOW.

As training corpus, Google News has been used. The corpus contains 6B tokens. In order to decide upon model hyper-parameters, a subset of the training data has been used with CBOW, with a vocabulary restricted to the most frequent 30k words [9]. The performance has been assessed using two sets of questions (semantic, syntactic). The question is considered to be answered correctly if the closest vector to the embedding computed using the relevant "word algebraic operation" corresponds to the right word required by the test pair. Some examples of such questions are shown below:

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Figure 2.4: Examples of five types of semantic and nine types of syntactic questions in the Semantic- Syntactic Word Relationship test set, Mikolov et al [9]

The main observation here was that for a significant increase of performance, we must increase both the dimensions and the amount of training data. Further testing, performed on 320M words, 82K vocabulary, 640 dimensionality of word vectors showed that the skip-gram architecture works slightly worse on syntactic tasks than CBOW, and much better on the semantic part. With a subset of Google News data, CBOW was

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Figure 2.5: Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set, one CPU, Mikolov et al [9]

trained in about a day whereas skip-gram in about three days.
The takeaway here is that "training a model on twice as much data using one epoch gives comparable or better results than iterating over the same data for three epochs". The main contribution of this methods is that they have shown the possibility to train "high quality word vectors with simple model architectures, compared to the popular neural networks models, which are more complex (feedforward or recurrent). They also make possible the use of large datasets combined with large embedding dimensionality. models (both feedforward and recurrent).

2.4.4 Negative Sampling

Going back to the basic Skip-Gram, we recall that for a target word w_t and a context word w_{t+c}, where $-k \leq c \leq k,\ c \neq 0$, the probability of a correct prediction is computed using the softmax formula. For convenience, we will denote the input word w_I, the output word w_O and their respective embeddings v_{W_I} and v_{W_O}.

$$p(w_O|w_I) = \frac{\exp(v_{w_O})}{\sum_{i=1}^{V} \exp(v_{w_i})} \tag{2.5}$$

In training, we will eventually have to compute the gradient of $\log p(w_O|w_I)$ (since we use the cross entropy loss 2.3). This computation is proportional to the size of the vocabulary V, which is of often hundreds of thousand, possibly even millions of tokens, making it inefficient.

To counter this drawback, a slightly different approach was proposed by the same researchers in a subsequent publication [10]. This time, instead of predicting a word out of $|V|$ classes, the task will be to predict the output word w_O from a relatively small set of n sampled words, called *negative samples*. The size of n depends on the size of the training data ($2 \leq n \leq 20$). The negative samples are selected according to a noise distribution which is a training parameter. The distribution proposed by the authors [10] is the unigram distribution raised to the power of $\frac{3}{4}$ normalized by the sum Z, $U(w)^{\frac{3}{4}}/Z$. This is done in order to slightly reduce the impact of very frequently occurring words and slightly increase the probability of scarce words to be sampled. The model architecture uses two embedding layers: the input layer, used for projecting the input word and the output layer, used for projecting the context (output) words. Therefore, for each word w_i, v_{w_i} will denote the input embedding and v'_{w_i} will denote the output embedding. In order to understand the objective function that measures the quality of the prediction, we consider the input word w_I, the "positive" sample w_O and n negative samples w_i, $1 \leq i \leq n$. The negative samples are sampled with respect to the selected distribution, $w_i \sim P_n(w)$. The objective function can be understood as using the logistic regression to distinguish between the positive sample and the negative one:

$$\mathcal{L}(w_O|w_I) = \log(\sigma({v'_{w_O}}^T \cdot v_{w_I})) + \sum_{1 \leq i \leq n} \log(\sigma(-{v'_{w_i}}^T \cdot v_{w_I})) \tag{2.6}$$

Intuitively, the formula above can be understood as trying to maximize the similarity (vector dot product) between the input and the positive context and minimize the similarity between the input and the negative context, hence the minus sign in front of negative samples' output vectors.

Besides sampling the negative context words, there is also the necessity of sampling the positive contexts. This may seem straightforward: just parsing the sequence and take every token as one. However, experience shows that the more frequent a word is, the less information it brings. Stop words like "a", "the", "in", etc. can occur hundreds of millions of times without any benefits. For example, consider the occurrence of "Germany" with "Berlin" and the occurrence of "Germany" with "the". This imbalance was countered using a subsampling technique [10]: a word w_i is discarded with the probability:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.7}$$

where $f(w_i)$ is the frequency of the word and t is a threshold, for example 10^-5. The

article suggest that this heuristic "accelerates learning and increases vector accuracy for rare words".

Experimental results presented in the article show that on a 1B words dataset with a vocabulary size of 692K, both the semantic and syntactic accuracies can reach 61%, being trained in under one hour. The significant improvements brought by the negative sampling modifications stand on better computational efficiency (allowing to use larger a dataset), better representation of uncommon words due to subsampling and also accurate representations for frequent words.

2.5 Global Vectors (GloVe)

2.5.1 Model Definition

A different method for training word embeddings was proposed in 2014 by Stanford researchers [11]. This method makes use of word co-occurrences statistics on which a weighted least squares model is trained. Because "certain aspects of meaning" can be extracted from co-occurrence probabilities, the connection is established between the probabilities and the inner products of the vector representations. The co-occurrences are defined by the matrix X, with X_{ij} representing the number of occurrences of the word j in the context of i and X_i being the total occurrence of the word i. Depending on how we define this relationship, X can be symmetrical or not. Specifically, the authors treat the word and the context interchangeably.

Consider the word w_i and the context word w_j. With the aid of linear algebra reasoning (see details in the article), the authors came to the conclusion that

$$w_i^T \cdot w_j = \log(P_{ij}) \tag{2.8}$$

where P_{ij} denotes the probability of the two word co-occurring and is defined as

$$P_{ij} = \frac{X_{ij}}{X_i} \tag{2.9}$$

Then, from 2.8 we obtain

$$w_i^T \cdot w_j = \log(X_{ij}) - \log(X_i) \tag{2.10}$$

In order to really make the formula above "symmetrical" with respect to the word and the target word, X_i is replaced by a bias b_i. A bias b_j is also added for the context word.

The weighted least squares regression objective function used is, then:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \cdot w_j + b_i + b_j - \log(X_{ij}))^2 \tag{2.11}$$

where V is the size of the vocabulary. The objective function is then minimized with gradient descent. The weighting function f serves the purpose of discarding pairs for which $X_{ij} = 0$, avoiding the undefined logarithm. It also makes sure that frequent co-occurrences are not overweighted and neither are the rare ones. One possible such function is

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

with $x_{max} = 100$ and $\alpha = 3/4$.

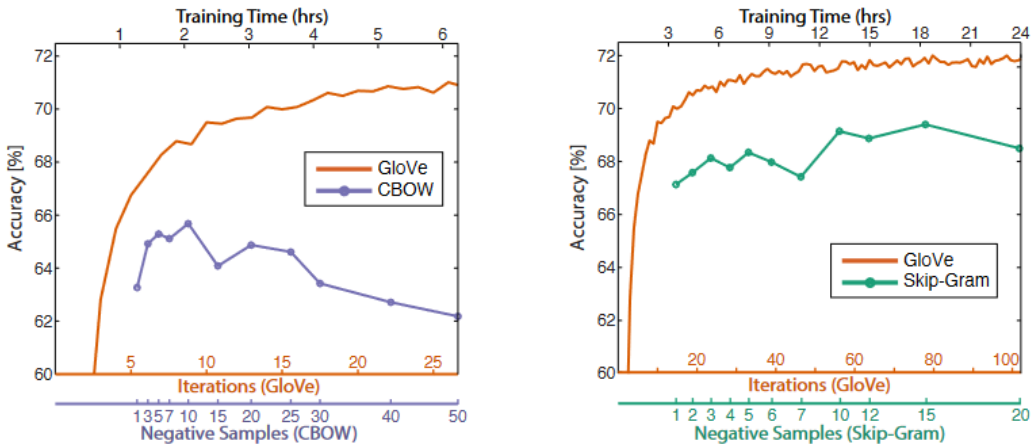
Looking at the objective formula 2.11, it is clear that in the worst case the complexity scales to $O(|V|^2)$ as opposed to shallow window-based models like `word2vec`, which depend on the corpus size $|C|$. Since we are only interested in the nonzero co-occurrences, the matrix X is significantly reduced, which helps prove that the overall complexity of the model is much better than $O(|V|^2)$, which for large vocabularies can reach up to hundreds of billions (larger than most corpora). Moreover, they perform a little bit better than the window-based approaches as well [11].

2.5.2 Training & Results

Several training datasets have been used, with sizes ranging from 1B to 42B tokens. The vocabulary size was 400k most frequent words. When constructing the matrix X, the context size d was considered as 10 words to the left of the word and 10 words to the right. As we saw in the previous section, during training, two sets of embeddings are learned. Since X is symmetric, the final embeddings represent the sum of the two trained embedding matrices.

The performance is compared to several embedding methods, among which the previously discussed `word2vec` method (skip-gram and CBOW on 6B tokens, 400K vocabulary, 10 window size and 10 negative samples). On the word analogy task described in the previous section, GloVe outperforms both skip-gram and CBOW, on all dimensions (100, 300, 1000). Another testsets used were word similarity tests and the corresponding spearman rank correlation coefficient. Again, GloVe has shown to outperform `word2vec`.

Another important aspect is the training time. For GloVe, it depends mostly on the number of iterations of the minimization algorithm, whereas the skip-gram and CBOW depend on the number of negative samples. GloVe not only outperforms `word2vec` but also achieves the results faster.



(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

Figure 2.6: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus with the same 400,000 word vocabulary, and use a symmetric context window of size 10. (Pennington et al 2014 [11])

2.6 Why Medical Word Embeddings?

In order to get an idea of how these pretrained word embeddings capture the semantics of medical terms, we analyzed the 10 most similar words for some terms that can be found both in medical contexts and also general contexts:

heart	**cell**	**tissue**	**organ**	**muscle**
cancer - 0.724	cells - 0.826	tissues - 0.88	organs - 0.689	muscles - 0.821
blood - 0.7	cellular - 0.71	bone - 0.755	harpsichord - 0.616	bone - 0.676
brain - 0.693	brain - 0.627	cells - 0.753	liver - 0.611	abdominal - 0.674
pain - 0.691	tissue - 0.619	brain - 0.749	transplantation - 0.606	muscular - 0.665
surgery - 0.686	embryonic - 0.612	skin - 0.729	transplants - 0.595	tendon - 0.662
cardiac - 0.67	phones - 0.601	tumor - 0.724	transplant - 0.583	stiffness - 0.66
chest - 0.66	dna - 0.6	connective - 0.7	instrument - 0.567	stress - 0.646
stomach - 0.657	phone - 0.596	marrow - 0.697	piano - 0.566	pain - 0.643
hospital - 0.652	stem - 0.59	tumors - 0.686	kidney - 0.558	flexing - 0.639
failure - 0.648	tumor - 0.586	cartilage - 0.671	marrow - 0.541	strain - 0.638

Table 2.2: 10 Most similar words according to GloVe trained on 6B words with 100d vectors.

For GloVe trained vectors, most of the similar words are related to the medical field, but the connections of some words to the target word remains somewhat general. For example ”heart: and ”cancer” are not usually directly related in the medical literature. Same holds for ”heart” and ”brain”, etc. Among the retrieved words we also notice non-medical words, such as ”phone” for ”cell”, ”instrument” and ”piano” for ”organ”.

In the case of `word2vec` trained words, the medical focus is stronger for all words except ”organ”, which has no medical similar words in the top 10. We also notice the presence of phrases. This is due to the fact of another improvement brought to the algorithm which allows words that co-occur very frequently to be concatenated with ”_” and be included in the vocabulary. We note the significant difference of the training corpus between the two models, specifically, for a good performance on medical semantics, 100B tokens worth of data was necessary.

heart	**cell**	**tissue**
hearts - 0.591	cells - 0.818	tissues - 0.827
Heart - 0.559	Cells - 0.615	Tissue - 0.639
heartbeat - 0.531	cells-hESC - 0.579	tissue-grafts - 0.637
congestive-heart-failure - 0.516	cells-hESCs - 0.563	amniotic-membrane - 0.629
cardiac - 0.515	Ovonic-fuel - 0.559	corneal-cells - 0.624
myocardial-infraction - 0.503	cell-hESC-derived - 0.552	muscle-tissue - 0.624
coronary-artery-clots - 0.503	booster-EPO - 0.543	stem-cells - 0.621
lion-Frese - 0.498	HSCs - 0.543	dermal-tissue - 0.611
empowers-CSPs - 0.497	neural-progenitor - 0.542	mesothelial-cells - 0.607
coronary - 0.495	neural-progenitor-cells - 0.542	corneal-tissue - 0.604
organ	**muscle**	
organs - 0.758	muscles - 0.74	
Organ - 0.69	faceoff-flexing - 0.691	
pipe-organ - 0.622	muscular - 0.623	
Wurlitzer-pipe - 0.621	muscle-tissue - 0.59	
Casavant-pipe - 0.599	protruding-Belo - 0.583	
Organs - 0.562	muscle-fibers - 0.563	
Aeolian-Skinner - 0.553	quadriceps-muscles - 0.56	
Kokaew-detention - 0.544	calf-muscles - 0.557	
Casavant-Freres - 0.53	abdominal-muscles - 0.552	
Anne-Paschke-spokeswoman - 0.527	buttock-muscles - 0.551	

Table 2.3: 10 Most similar words according to word2vec trained on 100B words with 300d vectors.

Different attempts of training and evaluating word embeddings are there in the literature. One example is discussed by Wang et al [5], in which medical word embeddings were trained using Mikolov's skip-gram approach with negative sampling. Two different training datas have been used: a corpus of electronic health records of 113K patients, with 103k vocabulary size and a set of 1.25M medical articles with 2M words as vocabulary. The main goal was comparing the performance of the obtained embeddings to the performance of pretrained GloVe and `word2vec`. Various tests (which we will go through in depth in Chapter 3) revealed that firstly, embeddings trained on EHR seem to be superior to the one trained on medical literature. EHR embeddings are more focused on clinical terminology whereas medical lit. embeddings focuses on medical scientifical terminology. Secondly, word embeddings trained on medical specific text capture the semantics of medical terms better than GloVe and Google News, being able to map relevant medical terms better.

Another 2019 study [12] compared two sets of embeddings also trained on medical records (both 1.5M words, one with 50 vector size, another with 300), belonging to a specific hospital and another one trained on general text (including wikipedia), with 300 vector size. After some information extraction tests performed with various known classifiers, the results have shown that the pre-trained general model has achieved the highest score. However, the authors note the significant difference between the sizes of the corpora: 788M vs 1.5M. The relative difference in the F1 scores obtained is 0.02 -0.03 in the worst case, a somewhat small difference considering the huge difference in corpus sizes. This can only indicate the possibility of domain specific embeddings outperforming the general ones had they been trained on more data. These examples, and not only these ones, are encouraging and motivate us to try and train our own version of medical word embeddings.

Chapter 3

Training Word Embeddings

As mentioned in the introduction, our ultimate goals was to train a version of word embeddings of our own. Since we are especially interested in shallow window-based methods, we chose the **Skip-Gram** model with **negative sampling**, as it has shown to be a fast and robust algorithm suitable for this task. In this chapter we describe the dataset used, the preprocessing steps, the algorithm specifications, fine-tuning and ultimately we provide some training results.

3.1 Dataset and text preprocessing

As dataset, we opted for medical literature, as we found that obtaining Electronic Medical Records is not particularly easy because of issues regarding personal data. The medical literature used is freely available online [1]. Specifically, it is a small partition of the "PubMed Central ® (PMC) - a free full-text archive of biomedical and life sciences journal literature at the U.S. National Institutes of Health's National Library of Medicine (NIH/NLM)". Our partition consists of 500MB worth of medical raw literature, of which 450MB have been used for training and 50MB for testing. This accounts for 72.6M tokens (16,088 articles) used for training.

As preprocessing, we first removed the `Front` and `Refs` from the corpus. The `Front` contains general details about the publications, copy rights, etc. The refs is the bibliography. Following, we removed all punctuations, digits and symbols. One element we kept though is the '-' symbol, because we were interested in keeping the combined tokens, some retrieved examples are:

[1] https://ftp.ncbi.nlm.nih.gov/pub/pmc/oa_bulk/oa_comm/txt/

- protein-coupled
- non-clinical
- virus-infected
- virus-infected
- ligand-binding
- etc.

The vocabulary used consists of the most frequent 150K words, which manages to retain most of the significant words compared to larger vocabularies such as 200K size, considering the percentage of irrelevant terms in the last 40-50k words (word fragments, protein chains, etc.)

3.2 Model Configuration

Following the skip-gram with negative sampling architecture, we used two embedding layers: one for the target (input) word and another one for the context words (output). Even though the original model uses a dedicated objective function (eq. 2.6), we noticed that in practice it performs worse than using a softmax with a cross-entropy over the sampled context (the positive samples plus the negative samples). Therefore, in our model we used the latter. This means that we performed softmax as activation function for the logits obtained from the dot product of the target and the context.

In order to subsample frequent words when creating the positive pairs, we used an approximation of frequency in the proposed approach (eq. 2.7), we used the ordered list of tokens by their frequency and approximated the frequency with

$$\frac{1}{\text{rank} \cdot (\log(\text{rank}) + \gamma) + \frac{1}{2} - \frac{1}{12 \cdot \text{rank}}} \tag{3.1}$$

where γ is the Euler-Mascheroni constant and rank is the index of the token in the sorted list.

For sampling the negative context words, we used a unigram distribution (i.e, statistical distribution of words probability with respect to their frequency). However, for efficiency purposes, this unigram distribution is learned on the go, as positive samples are generated. This can be seen as an improvement because of its working principle:

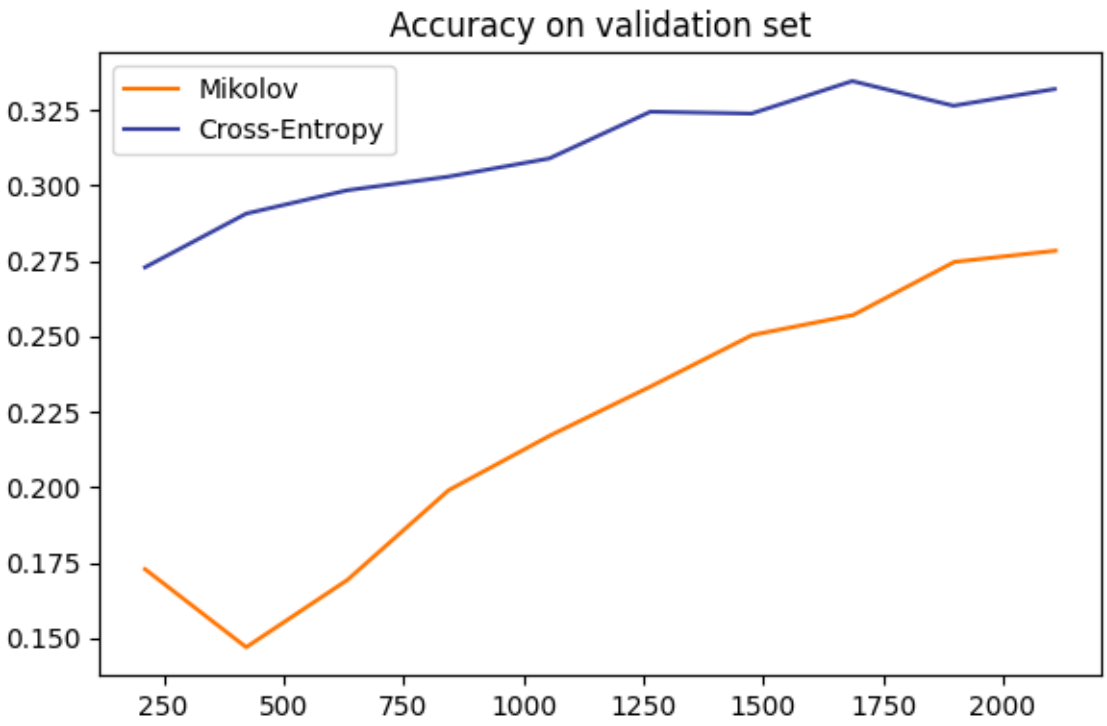


Figure 3.1: Comparison between two loss functions on a test subset of 307K words and vocabulary of 21K tokens

for each positive context sampled, the weight of that word is incremented. Initially, all words start with a sample of 1. Since the positive samples are sampled with a bias towards the less frequent tokens, they will have higher probabilities in the unigram distribution and therefore the frequent words will also be subsampled for the negative samples as well.

Since the model is trained on a batched dataset, we exemplify the working logic: for example, suppose we work with a batch of 5 samples with 2 negative samples and 10 embedding size. This mean we have 5 target embeddings and 5 sets of contexts, each with one positive sample and 2 negative ones. The matrix multiplication is illustrated below:

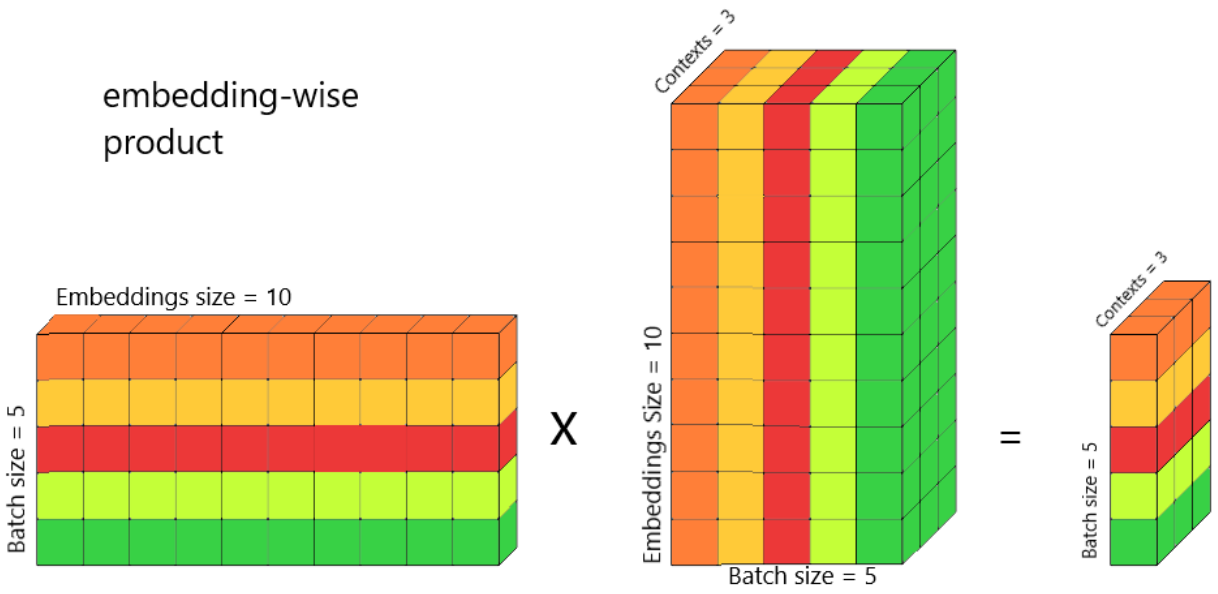


Figure 3.2: Tensor operations with word embedding batches. Three context words, of each one positive and 2 negatives.

The resulting matrix contains n lines of logits outputs, where n is the batch size. For each row, the softmax is applied with respect to the values and the cross entropy is applied (eq. 2.4).

3.3 Training

Using the same algorithm, dataset and vocabulary, we trained two different versions slightly changing some parameters. We present the two variations in the table 3.1.

For a window of size k, up to $2k$ classifications are performed for each token. Since the second variations uses a window of double size, the training has been performed on a double amount of sampled, each of them being a 8 binary prediction instead of 5. We used one epoch, since as established by Mikolov et al 2013 [9], one epoch with more data is better than several epochs with less data. The constant learning rate proved to be a better choice than the linearly decreasing one. We tested both of them in combination with Adam optimizer.

Training on 72.6M tokens with constant learning rate gives similar or better re-

Named as	WS2-NEG4	WS4-NEG7
Window Size	2	4
Negative Samples	4	7
Batch Size	1024	2048
Learning Rate	0.001	0.001
Epochs	1	1
Embedding Dimension	100	100
Vocabulary Size	150K	150K
Corpus Tokens	72.6M	72.6M

Table 3.1: Model specifications for the 2 trained variations

Embeddings	WS2-NEG4	WS4-NEG7
Accuracy	0.51	0.366
F1-Score	0.67	0.536
Loss	1.195	1.71
Hours	13.26	17.25

Table 3.2: Results and running time for the two trained embeddings

sults as training on triple amount of data with decreasing learning rate from 0.025 to 0.0001.

For the implementation, we used TensorFlow and Keras frameworks. Tensorflow offers good efficiency and performance when it comes to preprocessing and managing large amounts of data. This is primarily achieved using pipelines built with `tf.data` module [2] which can load, preprocess, and feed data into your model without having to load all data into memory at once. Keras is a high level API which provides implementations of various architectures and utlities such as embeddings layers, dense layers, loss functions, metric calculation, optimizers. The components are also flexible and allow extension, for example using custom loss functions or custom model layerings. Even though we did not use GPU computing power, these packages can utilize GPU parallel training at a large scale.

For our training, we used a single thread, with a Intel i7 processor. The running times and scores are given in table 3.2. We can notice that the accuracy is not outstanding. However, we remind that the classification task was a "fake" one - our

[2] https://www.tensorflow.org/api_docs/python/tf/data/Dataset

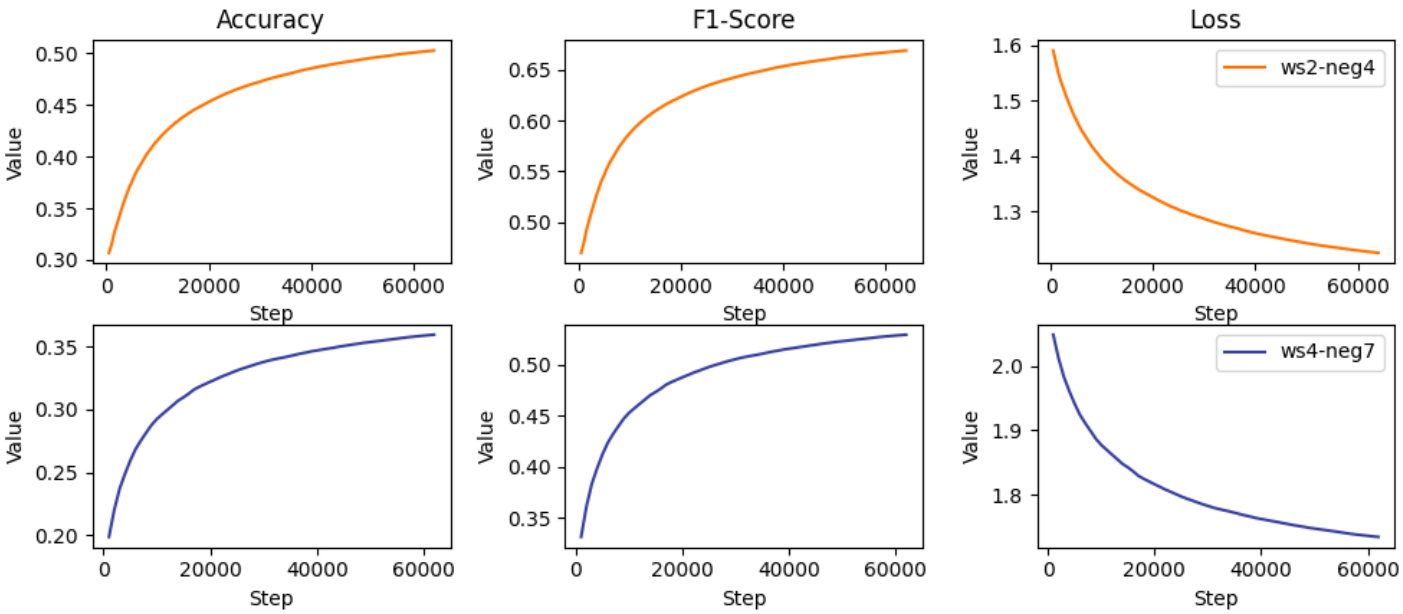


Figure 3.3: Metrics on the training set for the two model variations.

purpose was not creating a very good classifier, but to embed words using projection layers. Therefore, for evaluationg the quality of the embeddings, a different kind of tests is necessary. We perform a comprehensive analysis of the embeddings in the next section.

Chapter 4

Performance Evaluation

Our evaluations consist of using 4 different word embeddings in practice and analyzing the results for each one of them. We used the 2 word embeddings trained on medical literature and 2 widely used pretrained embeddings: a) GloVe on 100D trained on 6B of tokens - Wikipedia 2014 and Gigaword 5, which is an archive of news text data and b) Word2vec, trained on about 100B tokens from Google News on 300D. For quantitative evaluation, we tested semantic similarity on correlation sets and applied them on medical IE tasks.

4.1 Metrics

For a quantitative evaluation, we used the standard metrics found in the literature:

- **Pearson Correlation Coefficient** measures the linear relationship between two variables X and Y:

$$r_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{4.1}$$

  where n is the sample size, σ is the standard deviation and $\bar{X}, \bar{Y}$ are the means of X and Y. The correlation coefficient assumes that the relationship between the two variables under consideration is linear, so this measure is relevant only in that case. [13] The value of r ranges from -1 to 1, with 0 indicating no linear relationship, 1 indicating a perfect relationship, meaning that both variables increase or decrease together and -1 indicating a perfect negative linear relationship - when one variable is increasing, the other one is decreasing and vice-versa.

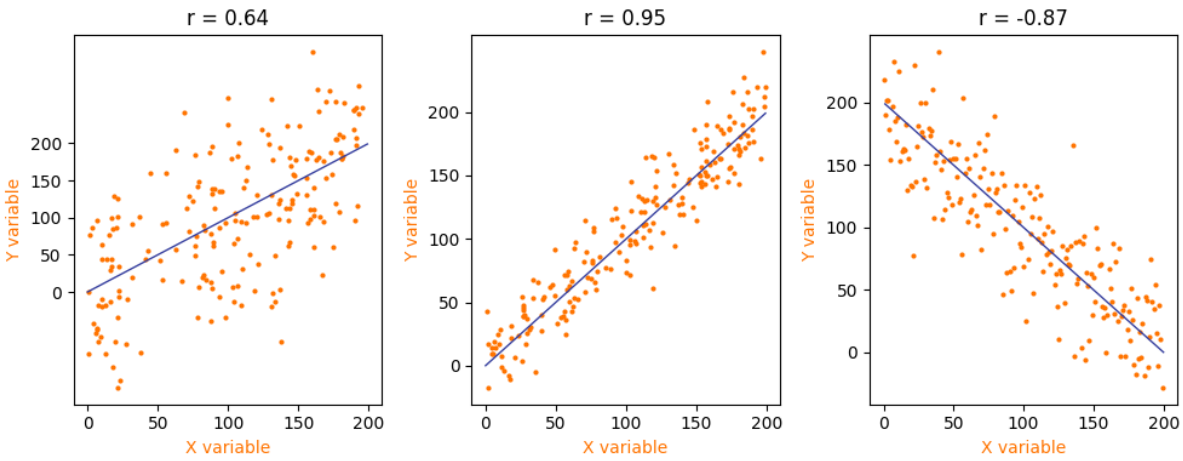


Figure 4.1: Visualizations for three different correlation coefficients

- **Classification Accuracy** - represents the proportion of samples correctly classified by a classifier model M and a set of samples S, each having a class $C(s), s \in S$ and a predicted class by the model $M(s)$. It is defined as

$$Acc_S(M) = \frac{\{s \in S | M(s) = C(s)\}}{|S|} \tag{4.2}$$

  For a binary classification (with only one class), we can define the accuracy as

$$Acc = \frac{TP + TN}{P + N}$$

  where TP are the samples correctly classified as being of the class, TN the samples correctly classified as not being so, P and N are the actual representatives of each class.

- **Recall** used in the context of a binary classification (or with respect to a specific class in a multi-class classification) represents the model degree of completeness, i.e, the proportion of positive samples (matching the class) that are correctly identified. It is calculated using the formula

$$recall = \frac{|TP|}{|TP| + |FN|} = \frac{|TP|}{|P|} \tag{4.3}$$

  This metric is relevant in instances where identifying all actual positives is critical, such as disease detection.

- **Precision** is also used in the context of a specific class and represents the proportion of correct classifications among all positive predictions. It tests the correctness of the model in recognizing a class. It is calculated with the formula

$$precision = \frac{|TP|}{|TP| + |FP|} \tag{4.4}$$

This metric is relevant in instances where the cost of false positive is high, such as misclassifying a healthy patient.

- **F1-score** is a measure of the trade-off between precision and recall. Using only recall can raise some issues, because if the model classified all samples as positive, the recall would be 1. Similarly, using only precision, if the model retrieved only one item in total, which happens to be a true positive, the precision would be $1/(1+0) = 1$. One way to assess the performance giving equal weight to precision and recall is using the f1 score:

$$F1 - Score = \frac{2 \times precision \times recall}{precision + recall} \tag{4.5}$$

4.2 Correlation Coefficient Tests

Using the correlation coefficient defined above, 4.1, we tested compared our word embeddings with GloVe and `word2vec` against 3 biomedical datasets used to measure the semantic similarity between medical terms. For a pair of words or phrases, we compared the correlation between their cosine similarity (eq. 2.1) and the similarity identified by the testset. For a phrase, such as *Lactose Intolerance*, we considered the average vector representation over the words in the phrase. The testsets are as follows:

- **Pederesen's Dataset** [14], a set of 30 concept pairs annotated by trained professionals on a scale of 4 to 1: practically synonymous (4.0), related (3.0), marginally related (2.0) and unrelated (1.0).

- **Hliaoutakis's Dataset** [15], a set of 34 selected and evaluated by doctors on a continuous scale with 0 meaning the terms are completely unrelated and 4 meaning the terms are interchangeable. The similarity in the dataset contains the normalized similarity rates ranging from 0 to 1.

- **Pakhomov's Dataset** [16] a set of 101 pairs manually rated on a scale of 1–10 (1-closely related, 10-unrelated) by 13 medical coding experts. Their ratings were then averaged.

  For all comparisons, the same words have been used. This means that if certain words could not be found within the vocabulary of one compared set of embeddings, the pair has been excluded altogether, for all embeddings. Around 15 word pairs were excluded in total.

Dataset	WS2-NEG4	WS4-NEG7	GloVe-6B-100D	word2vec-100B-300D
Pederesen's (30)	0.366	<u>0.517</u>	0.392	0.31
Pakhomov's (77)	<u>0.203</u>	0.185	0.101	0.064
Hliaoutakis' (34)	0.261	0.273	<u>0.347</u>	0.297
Corpus Size	72.6M	72.6M	6B	100B

Table 4.1: Correlation Coefficient of word similarities computed with the word embeddings, the dataset scores and the training corpus size in number of tokens. Best score is underlined.

Even though the same words have been used for all datasets, some of the missing words were only missing from `GloVe` or `Word2Vec`, being present in our vocabulary. The pairs of the datasets are balanced in terms of term length, with the average number of tokens per term being 1.56 for Pedersen's, 1.51 for Pakhomov's and 1.62 for Hliaoutakis' dataset.

For **Pedersen's Dataset**, our `WS4-NEG7` word embeddings are the closest to the human judgement, being followed by `GloVe-6B-100D`. Overall, all embeddings perform best on this dataset. This may be due to the fact that fewer physicians were involved in the scoring (3). Moreover, all of them were from the same medical field, which makes the scoring more consistent.

On **Hliaoutaki's Dataset**, being of a comparable size, we notice that general purpose embeddings outperform the ones trained by us. Of course, this is due to the larger volume training data. It could also indicate that the terms here are more general slightly and more frequent in non-medical literature, so that general embeddings can capture their similarity as well.

Finally, on **Pakhomov's Dataset**, our `WS2-NEG4` outperform the other embeddings. We notice an overall decrease of performance for this dataset. This may be due to the fact that more physicians and medical coders were involved in scoring the terms, leading to some degree of ambiguity. It can also be the result of the bigger size of the testset compared other two, providing a more impartial perspective.

Overall, our two sets of embeddings have shown to capture medical semantic similarity with respect to human judgement better than the general purpose embeddings.

4.3 Information Extraction Tasks

In order to compare the performance of the word embeddings, we used them as a representation layer for a classifier in the context of medical documents classification. Therefore, the representation of each document is a vector of features obtained in the following way: for a document $d = (w_1, w_2, ..., w_n)$ represented as a sequence of words and a vocabulary V, the vector representation is defined by

$$v_d = \frac{1}{K} \sum_{w \in V} v_w \tag{4.6}$$

where K is the number of words found in the document and also present in the vocabulary V.

The representations have been fed to a conventional machine learning model, which is Support Vector Machine (SVM). We consider it to be suitable since it scales well to high-dimensional datasets, deals with non-linear relationships and has been seen to provide high classification accuracy in relevant or similar tasks [5].

From a mathematical perspective, SVMs rely to hyperplanes in an N-dimensional space that act as a decision boundary in the context of a binary classification of data points (in our case, document vectors). For a hyperplane $w \cdot x + b = 0$ and a data point, the decision rule is

$$w \cdot x + b \geq 0, \text{for positive cases,}$$

$$w \cdot x + b < 0, \text{for negative cases}$$

Or, assuming $y_i \in -1, 1$ is the label of the data point, we can generalize

$$y_i(w \cdot x + b) \geq 0, \text{ for all data points} \tag{4.7}$$

The underlying idea behind support vector machines is to maximize the distance between the data points and the hyperplane. Let H be the hyperplane $w \cdot x + b = 0$. The distance of a point to H (which should be maximized) is:

$$d(H, x) = \frac{|w \cdot x + b|}{||w||_2} = \frac{y_i(w \cdot x + b)}{||w||_2}$$

with y_i being the label of the tuple x. Considering the constraint 4.7, maximizing the "margin" is equivalent to minimizing

$$\frac{1}{2}||w||^2$$

which is convenient since it becomes a convex optimization problem. Since we have a minimization problem subject to a constraint, we use the Lagrangian Multipliers, we get that the hyperplane coefficients are computed with

$$w = \sum_i \alpha_i y_i x_i$$

where the terms α_i are the Lagrangian multipliers obtained when maximizing the Lagrangian function. Consequently, the decision rule becomes

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) + b \tag{4.8}$$

This decision rule shows clearly that the decision depends on the dot products of known samples and unknown samples. A support vector machine as described up

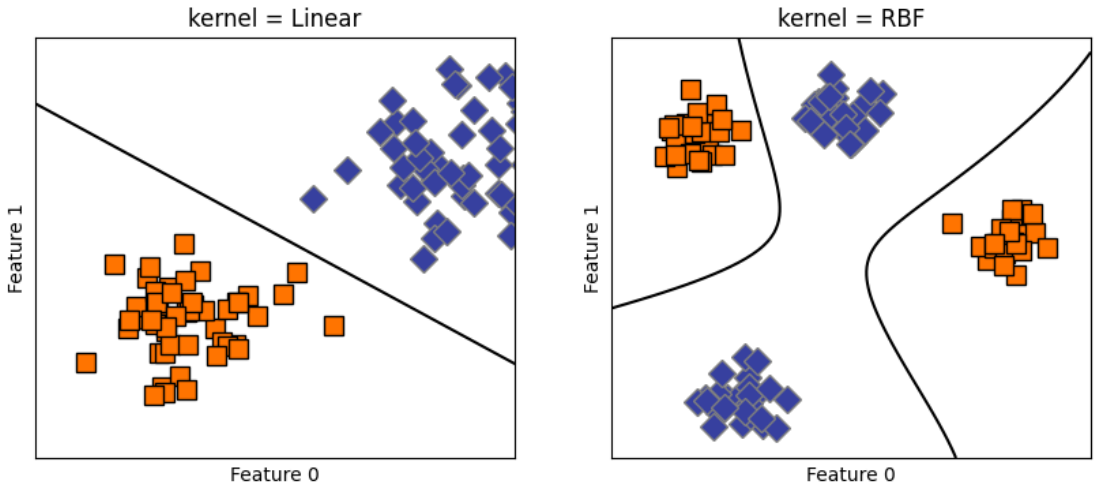


Figure 4.2: Two use cases of the standard kernel and the RBF kernel

until now can only classify linearly-separable data points. This is because if the relationship between features and classes is not linear, the dot product $x_i \cdot x$ will not capture it. We can still classify non-linearly separable data points using the kernel trick - instead of explicitly mapping the inputs to higher dimensions with various functions $\phi(x)$, we use a kernel function to directly compute the dot product of the inputs directly, without mapping the space to a higher dimension. One such function is the **Radial Basis Function**

$$K(x_1, x) = e^{-\gamma \|x_1 - x\|^2}$$

where γ is a free parameter and represents the inverse of the variance of the Gaussian distribution used to weigh the similarity between data points.

4.3.1 Task 1: Human or Veterinary Topic Classification

In this task, the aim is to classify medical texts (PUBMED articles) based on their human or veterinary medical nature. The dataset is publicly available online on Kaggle (Marcel Hiltner Master's student at FAU (Erlangen, DE).[1]. The training set consists of 38360 items and the test set is 4795 items (12.5%). One item looks like this:


```
{
    "pmid": "10982143",
    "text_types": ["Case Reports", "Journal Article"],
    "title": "toxoplasmosis as a suspected cause of abortion in a greenland
        muskox (ovibos moshatus wardi).",
    "abstract": "toxoplasma gondii tachyzoites were (seen in the placenta
       of a late-term aborted greenland muskox (ovibos moschatus wardi)
       fetus in a captive herd at the san francisco zoo.(...)",
    "title_abstract": "toxoplasmosis as a suspected cause of abortion in a
       greenland muskox (ovibos moshatus wardi). toxoplasma gondii
       tachyzoites were seen in the placenta of a (...)",
    "meshtermlist": ["Abortion, Veterinary", "Agglutination Tests", "
       Animals", "Anti-Infective Agents, Local", "Antibodies, Protozoan", "
       Birth Weight", "Female", "Histocytochemistry", "Oxytetracycline", "
       Placenta", "Povidone-Iodine", "Pregnancy", "Ruminants", "
       Tetracyclines", "Toxoplasma", "Toxoplasmosis, Animal"],
    "labels": 1
}
```


For training, the `title_abstract` key has been used, which represents the concatenation of the title and the abstract.

Metric	WS2-NEG4	WS4-NEG7	GloVe-6B-100D	word2vec-100B-300D
Accuracy	0.9165	0.9284	0.9382	0.9812
F1-Score	0.9161	0.9284	0.9383	0.9812
Precision	0.9208	0 0.9283	0.9379	0.9812
Recall	0.9115	0.9286	0.9386	0.9812
Corpus Size	72.6M	72.6M	6B	100B

Table 4.2: Accuracy, Precision, Recall and F1-Score on the human/veterinary medicine classification task with a **Linear** Kernel SVM

For the linear kernel, the general word embeddings perform better than the do-

[1] www.kaggle.com/datasets/marcelhiltner/pubmed-human-veterinary-medicine-classification

main specific ones. One explanation might be that since they are trained on broad data, the spatial positions of the embeddings make the document representations more linearly separable. For all cases, the precision and recall are relatively balanced. Since the test and training sets is also class-balanced, this indicates that the model deals similarly with false negatives and false positives.

Metric	`WS2-NEG4`	`WS4-NEG7`	`GloVe-6B-100D`	`word2vec-100B-300D`
Accuracy	0.9501	0.968	0.9472	0.986
F1-Score	0.9500	0.968	0.9471	0.986
Precision	0.9514	0.968	0.9481	0.987
Recall	0.9487	0.967	0.9462	0.986
Corpus Size	72.6M	72.6M	6B	100B

Table 4.3: Accuracy, Precision, Recall and F1-Score on the human/veterinary medicine classification task with a **RBF** Kernel SVM

On the **RBF** Kernel SVM, we see a noticeable improvement in performance for our medical domain word embeddings. This time, both of them outperform `GloVe-6B-100D`, even though they are trained on significantly less data. One possible reason is that the domain specific word embeddings create a feature space with more complex and non-linear relationships, being based on real medical semantic relationship which many times is complex. For example, a symptom can be associated with several diagnoses, treatment also varies from patient to patient. When word vectors try to approximate the relationship between them, it is likely that non-linear patterns are formed. Non-linearity might also have been introduced by the softmax activation performed prior to the cross entropy loss function during training. Consequently, mapping the similarity of datapoints into higher space allows more relevant features of each document to take part into classification.

As we have previously mentioned in chapter 2, the two variants trained on less data with a constant learning rate perform better than the model trained on more data and decreasing learning rate:

Metric	Linear Kernel	RBF Kernel
Accuracy	0.889	0.93
F1-Score	0.887	0.93
Precision	0.896	0.939
Recall	0.879	0.92
Corpus Size	210M	210M

Table 4.4: Performance of a model trained on 3 times more data with a decreasing learning rate 0.025 - 0.0001

4.3.2 Task 2: Smoking-Status Detection

In the second task we used the word embeddings in a task proposed by [2], which aims to detect the smoking status of patients based on their medical discharge records. The dataset is fairly small, with only 502 samples, of which 20% have been used for testing. As opposed to the previous task, this is a multi class classification problem, with the classes:

1. `CURRENT SMOKER`
2. `SMOKER`
3. `NON-SMOKER`
4. `PAST SMOKER`
5. `UNKNOWN`

with the difference between `SMOKER` and `CURRENT SMOKER` is that a smoker is a ”patient who is either a Current or a Past Smoker but whose medical record does not provide enough information to classify the patient as either”. The dataset is somewhat ”noisy”, as it contains a lot of medical abbreviations and personal data such as names. In an attempt to reduce the degree of noisiness, we preprocessed it minimally by removing the titles of fields. That is, we removed phrases such as ”admission date”, ”discharge date”, ”prior conditions”, ”medication history” and others. We have also performed standardization of the final document embeddings (centering around 0 and transforming the data to have the standard deviation around 1). Another purpose of

the standardization was capturing the distribution of the feature space better, since the dataset is small.

Since the dataset is not class balanced, with the `UNKNOWN` label occurring twice as often as the other, for both training and evaluation we used the 'balanced' approach - giving weights inversely proportional to class frequency.

Metric	WS2-NEG4	WS2.5-NEG7	GloVe-6B-100D	word2vec-100B-300D
Accuracy	0.485	0.504	0.445	0.584
F1-Score	0.482	0.51	0.43	0.536
Precision	0.542	0.566	0.449	0.559
Recall	0.485	0.504	0.445	0.584
Corpus Size	72.6M	72.6M	6B	100B

Table 4.5: Accuracy, Precision, Recall and F1-Score on the smoking status detection task with a **Linear** Kernel SVM

Metric	WS2-NEG4	WS2.5-NEG7	GloVe-6B-100D	word2vec-100B-300D
Accuracy	0.475	0.495	0.415	0.514
F1-Score	0.499	0.523	0.442	0.544
Precision	0.661	0.653	0.582	0.652
Recall	0.475	0.495	0.415	0.514
Corpus Size	72.6M	72.6M	6B	100B

Table 4.6: Accuracy, Precision, Recall and F1-Score on the smoking status detection task with a **RBF** Kernel SVM.

The limited size of the training set significantly impacts the performance of all models, with `word2vec-100B-300D` performing best with 0.58 on Linear kernel and with 0.51 on **RBF** kernel. One additional reason of word2vec performing the best, besides being trained on vast amounts of data, is that, as pointed out by Wang et al. [5], "the terminology used in the smoking status extraction task also appears frequently in the news, such as medications and advice for smokers". On this task, our medical word embeddings outperform the `GloVe-6B-100D` embeddings even with the linear the kernel, this suggests an higher degree of linear separability within the data.

Another effect is the overfitting induced by the RBF kernel, which decreases the performance for all models. One possible reason is that it unnecessarily adds complexity to a model trained on limited data.

On this task, we can expect a better performance for word embeddings trained on EHR instead of medical literature, since the training data consists of EHR notes.

4.3.3 Embeddings Visualization

We picked 20 terms from 3 different categories: drugs, symptoms and disorders. In order to obtained a visualization of the embeddings, we performed Principal Component Analysis and selected the 2 most significant components.

We observe that 3 clusters are naturally being formed with respect to each term class. For our embeddings, the cluster compactness is to a certain extent the same. We also note that certain disorders that in some cases can also be considered as symptoms are striving to the red cluster accordingly, for example: ”anxiety”, ”anorexia”, ”eczema”, ”insomnia”.

Another aspect is that in case of `glove-6B-100D` embeddings, the drug cluster is very compact as opposed to the other embeddings. Since the drug names are general and not sampled from a specific medical subfield, this compactness is not necessarily desirable.

Looking at our medical trained word embeddings, we can discover patterns, for example the words ”autism”, ”parkinson”, ”schizophrenia” and”parkinson” concern the brain and the nervous system. Other examples are: the proximity of ”psoriasis” and ”arthritis” (arthritis being one of the complications of psoriasis), ”depression” and ”anxiety”.

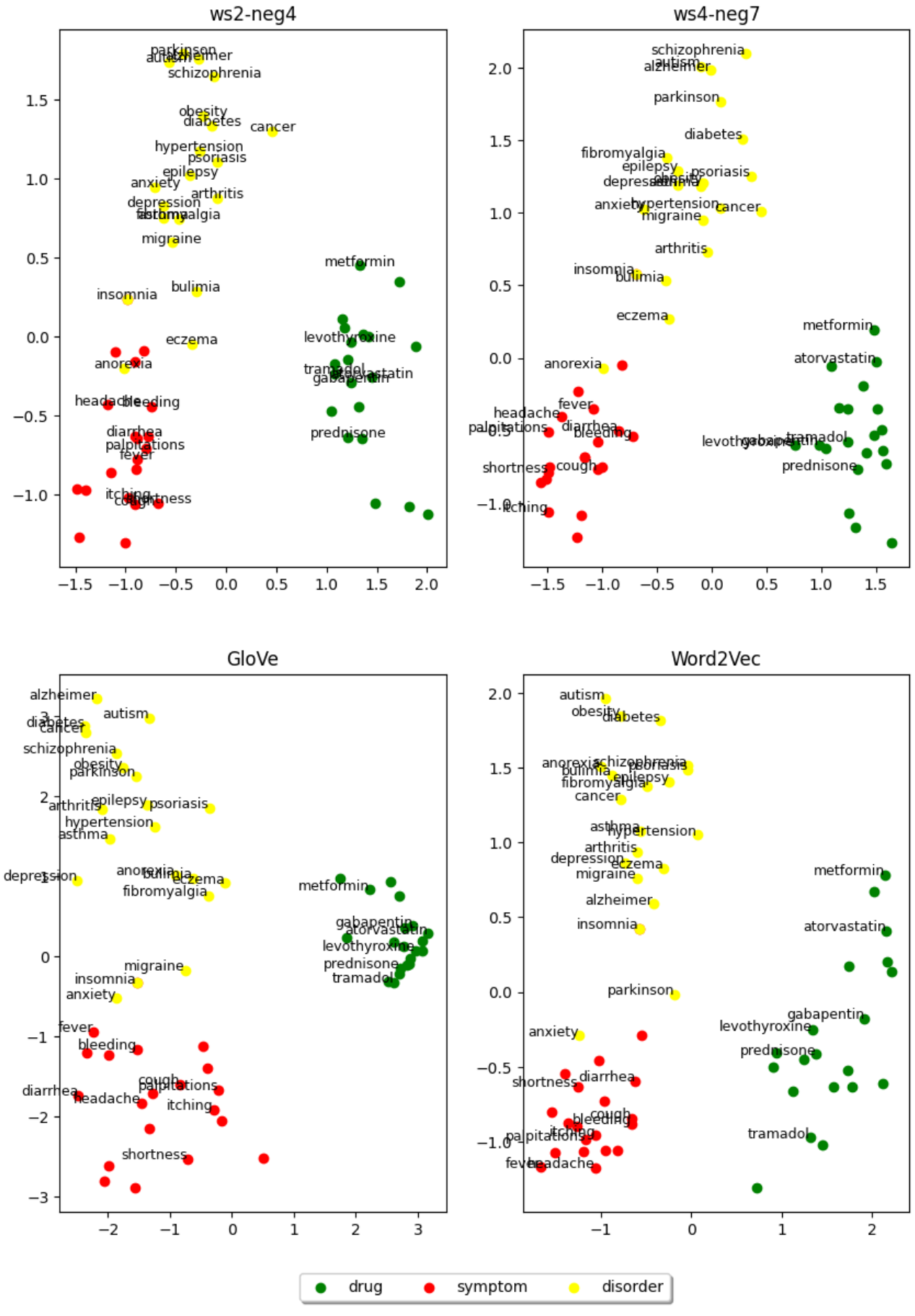


Figure 4.3: 2-Dimensional visualization using PCA of 20 drugs, disorders and symptoms.

4.4 Summary

Wrapping up, we conclude that our embeddings are showing a good performance on the experiments we have conducted.

On the correlation tests, we have managed to outperform the standard word embeddings on 2 out of the three relevant data sets tested, proving favorable semantic properties of the embeddings.

On the medical document classification tasks, our embeddings still performed well, scoring better than general word embeddings trained on a lot more data, especially for little training data. Through the use of the kernel trick, we have also seen the presence of non-linear patterns in the embeddings.

Finally, visualizing the word embeddings on 2D with PCA allowed us to observe the intuitive word clustering with respect to their medical meaning. For our embeddings, this word clustering seems to be more accurate and faithful.

Conclusions

This work is a comprehensive report of how we have trained and empirically evaluated domain specific word embeddings (biomedical domain). Using Word2Vec Skip-Gram with negative sampling proposed by Mikolov et al [9, 10] we have trained word embeddings on medical scientific literature and compared them to GloVe and Word2Vec general purpose embeddings.

Firstly, through quantitative tasks such as testing the correlation to similarity scores, we found that the resulting embeddings manage to reflect medical semantic traits for words, making them closer to human judgement for that matter.

Secondly, on the quantitative tests, namely medical information extraction, our embeddings have managed to outperform the pretrained GloVe embeddings. We have noticed an increase of performance when mapping the relationship of vector representations to a higher dimension (using the kernel trick). Another aspect is potential advantage of domain specific word embeddings on classification tasks with small amount of training data.

Thirdly, considering the quantitative tasks and also the visualization, we note a difference in the way the medical semantic are reflected by the embeddings, with ours managing to offer more precise word associations in the context a PCA visualization of word clusterings.

Lastly, considering the significantly smaller amount of training data used in the process (72.6M vs 6B vs 100B), we consider the training of domain specific word embeddings to be feasible and that, if sufficiently fine-tuned, it *might* provide a boost in performance for specific tasks. Even so, as seen in the empirical tests, there is no "best" word embedding for all possible tasks, and it is possible that general word embeddings are equally suitable for domain specific tasks.

As future directions, the running time of the model can certainly be further reduced (which is now 13-17 hours) by making use of parallel training. Also, experiment-

ing with hyper-parameters such as window size, negative sampling and maybe most importantly embedding dimension could present some advantages. Another aspect is the nature of the training data. Being trained on scientific literature, our embeddings can be expected to perform better on tasks that rely on scientific text. Conversely, we can presume that embeddings trained on real EHR notes would perform better on tasks that deal with real clinical text. However, retrieving such data can come with some issues, such as personal information issues, unstandardized text (institution-specific notation, abreviations), etc.

Bibliography

[1] Yanshan Wang, Liwei Wang, Majid Rastegar-Mojarad, Sungrim Moon, Feichen Shen, Naveed Afzal, Sijia Liu, Yuqun Zeng, Saeed Mehrabi, Sunghwan Sohn, and Hongfang Liu. Clinical information extraction applications: A literature review. *Journal of Biomedical Informatics*, 77:34–49, 2018.

[2] Özlem Uzuner, Ira Goldstein, Yuan Luo, and Isaac Kohane. Identifying patient smoking status from medical discharge records. *Journal of the American Medical Informatics Association*, 15(1):14–24, 2008.

[3] Kirk Roberts, Matthew S Simpson, Ellen M Voorhees, and William R Hersh. Overview of the trec 2015 clinical decision support track. 2015.

[4] Zhu HR Sable C Shanker V Ely J Yu H. Lee M, Cimino J. Beyond information retrieval–medical question answering. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, page 469–473, 2006.

[5] Yanshan Wang, Sijia Liu, Naveed Afzal, Majid Rastegar-Mojarad, Liwei Wang, Feichen Shen, Paul Kingsbury, and Hongfang Liu. A comparison of word embeddings for the biomedical natural language processing. *Journal of Biomedical Informatics*, 87:12–20, 2018.

[6] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 2017.

[7] A. J Sutton. *Concepts in Word Embeddings: Theory and Applications, Student thesis: Doctoral Thesis Doctor of Philosophy (PhD)*. 2021.

[8] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, 2014.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[11] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[12] Aleksei Dudchenko and Georgy Kopanitsa. Comparison of word embeddings for extraction from medical records. *International journal of environmental research and public health*, 16(22):4360, 2019.

[13] Bruce Ratner. The correlation coefficient: Its values range between+ 1/- 1, or do they? *Journal of targeting, measurement and analysis for marketing*, 17(2):139–142, 2009.

[14] Ted Pedersen, Serguei VS Pakhomov, Siddharth Patwardhan, and Christopher G Chute. Measures of semantic similarity and relatedness in the biomedical domain. *Journal of biomedical informatics*, 40(3):288–299, 2007.

[15] Angelos Hliaoutakis. Semantic similarity measures in mesh ontology and their application to information retrieval on medline. *Master's thesis*, 2005.

[16] Serguei VS Pakhomov, Ted Pedersen, Bridget McInnes, Genevieve B Melton, Alexander Ruggieri, and Christopher G Chute. Towards a framework for developing semantic relatedness reference standards. *Journal of biomedical informatics*, 44(2):251–265, 2011.