KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Learning a Diseases Embedding using Generalized Word2Vec Approaches.

Milan van der Meer

Thesis submitted for the degree of
Master of Science in Engineering:
Computer Science, specialisation
Artificial Intelligence

Thesis supervisors:
Prof. dr. R. Wuyts
Alexander Vapirev

Assessors:
Prof. dr. ir. H. Blockeel
R. van Lon

Mentor:
Dr. E. D'Hondt

Academic year 2015 – 2016

Preface

I start with expressing my gratitude towards Roel for giving me the opportunity to work on a truly interesting subject. I also appreciate that you were not just an invisible promoter, but a promoter who showed a true interest in the subject and curiosity for the results of the research.
I also know that I am a lucky student who fell with his "butt in the butter" as Ellie took the time and effort to proofread my thesis a couple of times. Together with Roel, you really provided me with a lot of support and made it possible to finish my thesis in a way I'm proud of. Thank you.
On a more personal note, I want to thank the people who stood by my side. My parents who gave me the opportunities in life and also prepared me for those opportunities. My friends, the CW kneusjes, my kotgenoten, and of course Siemen, the guy who somehow still manages to stay around. I also want to thank my future wife, just to cover all bases.

Milan out.

Milan van der Meer

Contents

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

EHR	Electronic Health Record
ICD	International Classification of Diseases
WHO	World Health Organization
MedDRA	Medical Dictionary for Regulatory Activities
THIN	The Health Improvement Network
ML	Machine Learning
CBOW	Continuous Bag-of-Words
knn	k-Nearest Neighbors
kd	k-dimensional
MLP	Multilayer Perceptrons
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
DL4J	DeepLearning for Java
MSLR	Thomson Reuters MarketScan Lab Database
OMOP	Observational Medical Outcomes Partnership
OSIM2	Observational Medical Dataset Simulator Generation 2

Chapter 1

Introduction

In todays medical world, more and more data is stored using Electronic Health Records. Each time a patient goes to a hospital, doctor or receives lab results, those events are stored in the patient's Electronic Health Record. The medical world and governments are interested in EHRs as they can for example provide new insights into disease trajectories, drug treatments, medical costs, or the link between demographics and certain diseases.

Due to the increased usage of EHRs, a new research area emerged, namely the area of Electronic Health Record Analytics. EHR analytics is an active research field as a lot of different problems need to be solved, like different codings, privacy, interpretation, and large amounts of data. At the moment several research groups are working on utilizing EHRs to find medical patterns using several methods like querying, statistics, data mining, and artificial intelligence approaches.
The methods we mentioned vary from simple to very complex. But there is still room for improvement, especially in the field of advanced machine learning algorithms. This is the focus point of this thesis: applying advanced machine learning algorithms to find patterns in EHRs.

An EHR of a patient can be seen as a time series, namely a sequence of EHR events such as visits to the doctor. We make the analogy between sentences of words and sequences of EHR events. Based on this analogy, we propose novel techniques which are generalizations of the Word2Vec approach.
Before we apply these Word2Vec methods, we preprocess our EHRs to categorize our EHR events into more general events. For example, a bruise on your right leg and a bruise on your left leg should both be categorized under a bruises event.

We introduce generalized Word2Vec. This generalized Word2Vec makes it possible apply Word2Vec on medical data.
To make sure the generalized Word2Vec methods can be applied to large-scale medical data, we apply the generalization concept also on DeepWalk. DeepWalk makes it possible to generate a smaller dataset from the original dataset and then apply a

Word2Vec approach on this smaller dataset.
Besides the exploration on generalizing Word2Vec approaches, we also improve Word2Vec by tackling one of its shortcomings. This shortcoming of Word2Vec is that it is unable to handle unseen instances once it has built his model. We combine a k-nearest neighbors method with Word2Vec and make an estimation of the correlation to other diagnoses for the unseen instance.

We compare the model from our Word2Vec methods applied on the OSIM2 dataset to the results of the currently largest study on Danish EHRs. To make it possible to apply our methods on the OSIM2 dataset and generalize our events, we categorize our diagnoses and make a disease code mapping between two standards namely MedDRA and ICD-10.

Within the limitations of our validation method, we conclude that our model does match the Danish results well depending on how the matching is defined. Especially since we use several estimations such as the disease code mapping, different datasets, and categorization.

In chapter 2, we describe the general field of EHR and EHR analytics. We also introduce some state-of-the art methods. In chapter 3, we provide you with the needed background to understand our generalized Word2Vec approaches and afterwards describe our novel techniques. In chapter 4, we describe how we make a model of the OSIM2 dataset using our proposed techniques and how to validate our model. In chapter 6, we mention some possible improvements and extensions left to explore.

Chapter 2

Electronic Health Record Analytics

2.1 Introduction

In section 2.2 we explain what electronic health records are and how these are represented. In section 2.3, we explain how the electronic health records can be used to retrieve useful medical information. We also explain different approaches to retrieve this information from the health records. Those approaches range from querying databases to more advanced machine learning techniques.

2.2 Electronic Health Records

An electronic health record (EHR) is a collection of time-stamped data about a patient over a period point of time. It is stored digitally and thus can be established for a large number of patients over a long time period.
The data stored in an EHR provides an overview of the patients' health information. Health information like demographics, medical history, diagnoses, medications, and such, are stored [2].

Recently large countries like the US and the UK, are each investing more than 20 billion dollars into EHR systems [60]. Those systems are adopted by around 70% of the physicians in the US, but of those systems a lot of different implementation are used. Which means a large number of physicians are using different methods or systems. This causes different ways of information representation and makes it harder to compare EHRs across different systems. We focus on disease codes in the next section and introduce two standards: one used by mainly insurance companies and the other used by pharmaceutical companies.

2.2.1 Disease Codes

To make EHRs practical it is important to adhere to standards for data formatting. A well-documented and consistently used standard makes it easy to store and extract information from large-scale databases of EHRs. Without the possibility of extracting information, an EHR becomes a simple digital version of medical records on paper. Part of an EHR consists of the diagnosis of the patient. It provides information about his disease trajectory and allows analysis on his health situation. With a uniform system for classifying diseases nationwide, it is possible to provide a general picture on health situations of populations. Several have been defined and their usage is scattered. We discuss the 2 most relevant to our work.

ICD-10

The International Statistical Classification of Diseases and Related Health Problems (ICD) is a medical classification list made by the World Health Organization (WHO) [7]. Several variants are available like ICD-9, ICD-10, ICD-10 CM, and others. The ICD-10 contains more than $14,400$ codes about diseases, disorders, injuries, and other related health conditions. For example, the code for a sprained ankle is $S93.4$. It also provides hierarchical categories for those codes to allow a more general overview of diseases. ICD is mainly used by insurance companies.

MedDRA

The Medical Dictionary for Regulatory Activities (MedDRA) provides medical terminology in the form of disease codes [3]. A MedDRA code is an eight digit numeric code where the code itself has no meaning. The code functions as an unique identifier so new terms are just identified with the next available code. Note that this system does not provide a clear and easy to use system to retrieve a hierarchical structure of a code. MedDRA is mainly used by pharmaceutical companies.

2.3 EHR Analytics

EHRs provide a massive amount of data which in principle can be used to create useful insights. The data contains the medical history of a patient including medical measurements, diagnoses, prescribed drugs, and demographics (see figure 2.1, where procedures (CPTs), lab results (LABs), visits to primary care physician (PCP) and visits to specialists (SPEC) are part of the EHR). Based on those values, we could obtain the following insights:

- Effects of drugs
- Medical costs for certain diseases
- Duration and recovery percentage of certain diseases
- Correlation between demographics and certain diseases

- Link between current health state and health history
- Classification and prediction of future health states based on history

Those insights can be offered on an individual level, which means a right intervention to the right patient at the right time. EHR analytics can be used to have a personalized care and benefits the healthcare system by cutting costs and improve outcomes [45].

EHR analytics is an active research field as a lot of different problems need to be solved, like different codings, privacy, interpretation, and large amounts of data. In the following sections we talk about current EHR analytical methods.

2.3.1 Querying

Analytics in epidemiology on EHRs is typically done through querying a database [30]. A specialist has a hypothesis about correlations between conditions or patients. He can support this idea by selecting cases in EHRs and analyzing the results of his query.
This method is based on the knowledge and experience of a specialist. The information has to be actively sought after and unexpected or complex correlations are not always considered. Some complex relations cannot be found because of the limitations of the querying language, ex. a first-order logic query language. Currently The Health Improvement Network (THIN) research team at UCL conducts these kinds of research [5].

2.3.2 Data Analytics

More advanced methods are applied to EHRs than querying. In general, the goal is to find patterns in the EHR data which then can be used to predict outcomes of treatments or classify a patient's disease trajectory [35].

Several predictive/classification methods from machine learning can be used and show promising results [14]. Those results are achieved by using exploratory methods which are applied to the EHR data. We also note that methods in [14] as Multi-layer Perceptron networks are not ideal for prediction of time-series like medical data, see section 6.3. We conclude that there is still a lot of room for improvement.

More specialized approaches are also applied to EHR data [64] besides the above mentioned machine learning techniques. An EHR of a patient can be transformed into a matrix structure, see figure 2.1. On these matrix structures, large-scale data mining algorithms can be applied. Those make it possible to mine patterns in EHR data. The patterns found can be used as the basis for predictive models.
It is also possible to define patient similarities [61]. So when a patient is similar to a previous known case, his treatment can be based on those previous experiences. In a way, it could be seen as a similar approach as a recommender system [64].

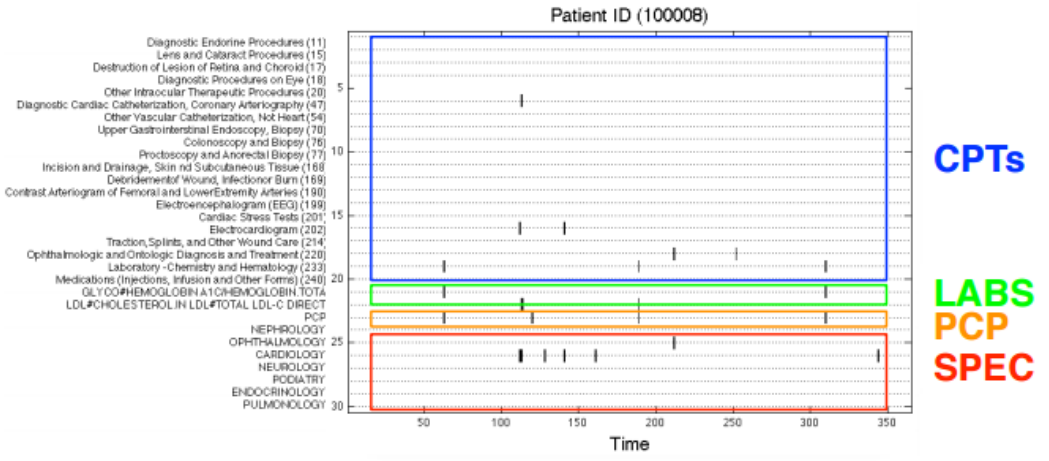


Figure 2.1: Example of an EHR transformed into a matrix structure [64]

2.3.3 Data Mining

In 2015, a more data mining approach is used to find patterns in EHR data on a dataset of the Danish population [34]. The results from [34] are clusters of disease trajectories which will be used to validate our approach described in chapter 3. Their results are used because it is currently the largest study performed on EHRs. You can find an example of a clustered trajectory in figure 2.2.

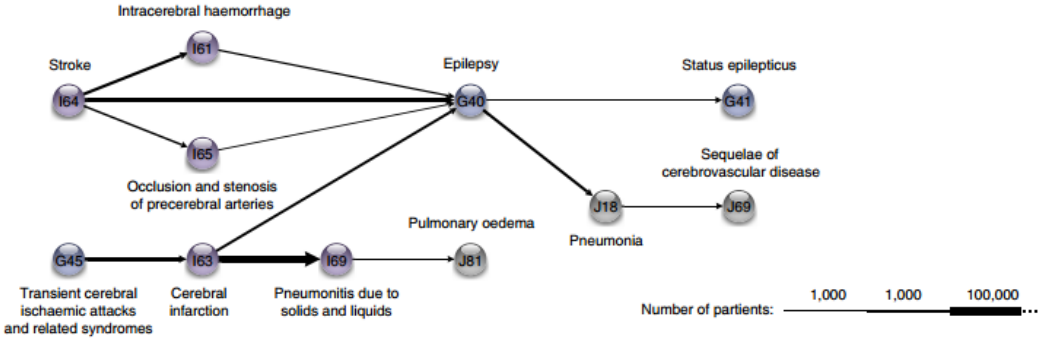


Figure 2.2: Cerebrovascular disease trajectory cluster for the Danish population [34]

The dataset which is used to apply the data analysis on, consist of EHRs collected over 15 years on over 6 million patients in Denmark. The size of this dataset makes it possible to retrieve statistically significant results.

They start with finding pairs of patients' diagnoses which have a strong temporal

correlation between them. After finding the correlated pairs, a test for directionality is applied. From this, only the pairs with a high enough indication for a direction are kept, ie which often appear one after the other within a certain time span.

The directed pairs are then connected into longer trajectories when they have overlapping diagnoses. The found trajectories are then clustered. From the clusters, diagnoses can be found which are key in the disease progression. Those key diagnoses could be used to predict future disease progression of patients.

2.4 Conclusion

EHRs contain important information of a patients medical history and current state. When a large amount of EHRs is available, relevant results can be found in the form of patterns. Those pattern can be used to predict and improve medical outcomes on a personal level.
The methods we describe vary from simple to very complex. But there is still room for improvement, especially in the field of advanced machine learning algorithms. The results of the Danish paper can be used to have a first validation of our approach, namely a generalized Word2vec approach.

In the next chapter we explain the required background knowledge to understand our approach on finding patterns in EHRs. After introducing you to those concepts, we also explain our approach, namely a generalized Word2vec approach.

Chapter 3

Generalized Word2Vec for Electronic Health Record Analytics

3.1 Introduction

In this chapter we introduce important concepts which are needed to understand our approach. These concepts can be used to solve problems described in chapter 2.

We start with explaining some background knowledge in section 3.2 such as time series and machine learning. We focus on basic concepts from machine learning and then focus more on neural networks. The main part to understand our approach is the introduction of Word2vec in section 3.3. This is then used to introduce Deepwalk as an extension on Word2Vec in section 3.4.

After introducing those concepts, we explain our generalized Word2Vec approaches in section 3.5.

3.2 Background Knowledge

3.2.1 Time Series Analysis

A time series consists of data points over a certain time period. We refer to this as a sequence of states. A state represents a data point and can differ from a single value to more complex representations like pictures.
The domain of time series analysis deals with extracting information or relations from a time series. It can have different goals like forecasting, classification, or exploratory analysis.

A medical history of a patient can be seen as a time series, namely a sequence of visits to the doctor. This means that methods which are applied to time series,

can also be applied to medical data to find patterns. We focus on machine learning approaches which are applicable to time series.

3.2.2 Machine Learning

Machine learning (ML) is a data-driven approach which aims to build a model which can be used to make predictions or decisions on new data [20]. Note that this model can be used to predict outcomes of time series. ML is carried out by algorithms which are able to learn models based on examples given by the designer. Based on the examples, machine learning provides 3 types of approaches, namely supervised learning, unsupervised learning, and reinforcement learning.
Supervised learning is concerned with the learning task where there are examples given with their corresponding label (for example, a picture with the label dog). Unsupervised learning is similar to supervised learning only no labels are given. We will not go into reinforcement learning.
We can also classify the ML approaches according to the desired output of our model. Those main tasks consist of classification, regression, and clustering. Classification has a discrete output, namely the predicted label. Regression has a continious output, namely a predicted value. Clustering is typically an unsupervised approach which tries to find patters in the data based on a similarity measurement.

In the field of classification neural networks are used to achieve state of the art results. For regression, linear regression can be used. One of the most popular methods for clustering is k-means.

3.2.3 K-nearest Neighbors

The k-Nearest Neighbors (knn) algorithm is a simple machine learning algorithm [17]. It will be used in our generalized Word2Vec approach.
When you are in a supervised context, you have several instances with labels. When you retrieve a new instance, you want to predict its label. Based on a predefined similarity measure (ex. Euclidean distance), you look for the k labeled instances nearest to the new instance in a vector space determined by the features/variables. From those, you pick the most common label in the pool of the k nearest instances. This label becomes your predicted label for the new instance (see figure 3.1).

3.2.4 K-dimensional Tree

A naive way to calculate the nearest neighbors for an element in a vector space, is by comparing all members with the new element and keep track of those distances. A more efficient way is to use a k-dimensional tree (kd tree) [59]. In this section we explain the workings of this approach in more detail [47].

A kd tree is a way of storing k-dimensional points. It is a binary tree where each node represents an element in the vector space. The node also contains information

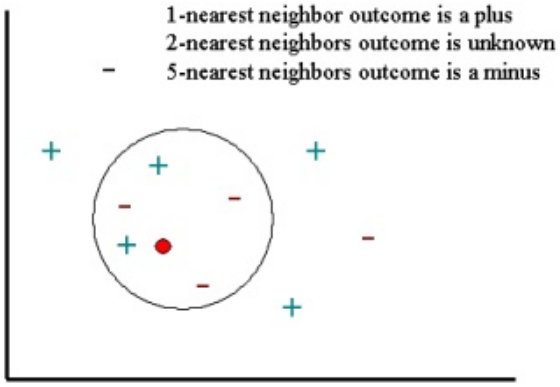


FIGURE 3.1: Representation on how the knn algorithm predicts a label for a new instance.

on how the tree is split up in terms of hyperplanes in the vector space. It keeps track of the plane it is split on and the left and right sub tree. In figure 3.2 you can see an example on how a simple kd tree is constructed and how it splits up the x,y plane. In the top figure, the splitting plane is not mentioned. The bottom figure shows that it is the $y = 5$ plane for the $[2, 5]$ and $x = 3$ for $[3, 8]$.

Now that we can construct the kd tree, we can use it to find the nearest neighbor for a certain input point. We start with the root node and go down the kd tree depth-first. At each node, the algorithm goes to the left or right depending on whether the input is less or greater than the current nodes value on the splitting plane. Once the algorithm reaches a leaf node, it marks this leaf node as the current nearest neighbor.
The algorithm unwinds the recursion of the tree, performing the following steps at each node:

- If the current node is closer than the current best, then it becomes the current best.
- It checks whether there is the possibility of points closer to the input point on the other side of the splitting plane. It makes a hypersphere around the current node with a radius equal to the current nearest distance.
  - If the hypersphere crosses the splitting plane, there could be a closer point on the other side. This means the algorithm will move down the other branch of the current node.
  - If the hypersphere does not cross the splitting plane, the whole other branch can be skipped.

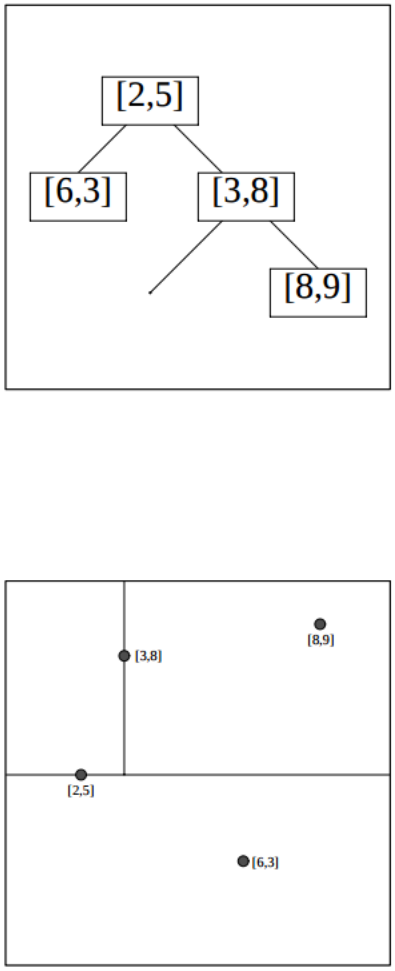


Figure 3.2: Representation on how a binary kd tree splits up the plane [47]

This algorithm is easily extended to find the k-nearest neighbors by keeping track of the k current bests. Kdtree has a complexity of $\mathcal{O}(\log n)$ compared to $\mathcal{O}(dn)$ with d the dimension of the search space for a naive knn implementation.

3.2.5 Neural Networks

A neural network is a machine learning approach based on biological neural networks. It can be used to find patterns in and make predictions about time series. Those time series can be speech, videos, medical data, ...
Besides being used in finding patterns and making predictions, it is also used in Word2Vec. This is explained in section 3.3.

Perceptron

The basic component of a neural network is the perceptron [55]. A perceptron takes multiple binary inputs and has a single binary output (see figure 3.3). Each input has a corresponding real numbered weight w_j. The output is decided by the following equation:

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{3.1}$$

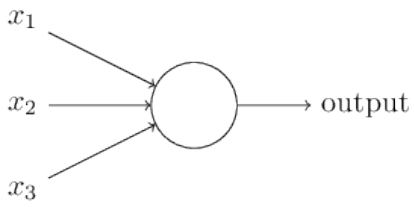


FIGURE 3.3: Simple representation of a perceptron [48].

The equation above, represents an activation function. The function retrieves certain inputs, and has an output between 0 and 1. There are several activation function possible in a perceptron as we will discuss later on.
We can build a network by connecting multiple perceptrons (see figure 3.4). By building these networks, more complex decisions can be made. The reason for this, is that once there are at least 3 layers of perceptrons (and non-linear activation functions), the network can find non-linear relations between the input and output [32].

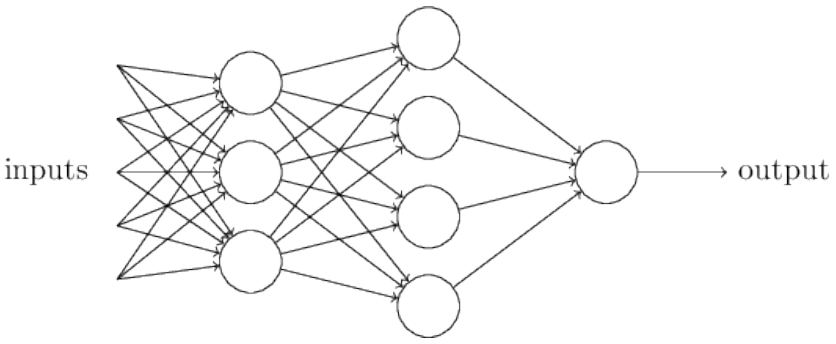


FIGURE 3.4: More complex network connecting multiple perceptrons [48].

Terminology

Now that we have seen how a general network is constructed, we introduce at some further vocabulary.
In figure 3.5, we see a four-layer network. As specified in the figure, we call the first layer the input layer, the last layer the output layer, and the layers in between hidden layers. Sometimes a multiple layer network is referred to as multilayer perceptrons or MLP.

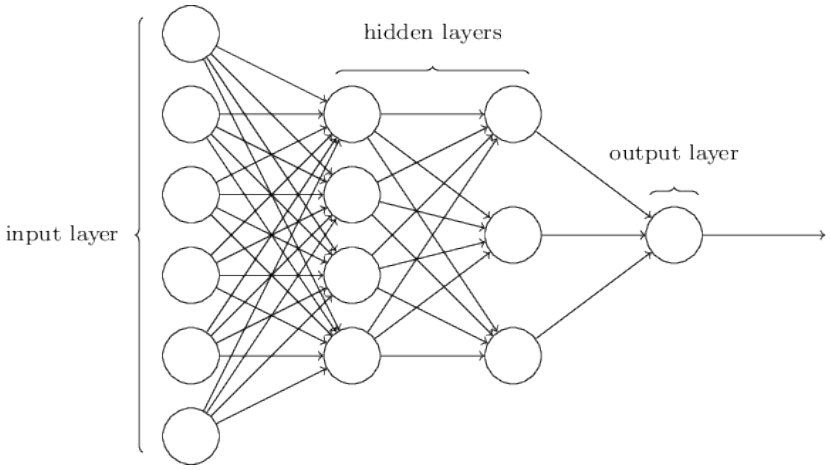


Figure 3.5: General vocabulary of a multilayer network [48].

We use w^l_{jk} to denote the weight corresponding to the connection between the k^{th} node in the $(l-1)^{th}$ layer and the j^{th} node in the l^{th} layer. We use b^l_j for the bias of the j^{th} node in the l^{th} layer and a^l_j for the activation of the j^{th} node in the l^{th} layer. See figure 3.6.

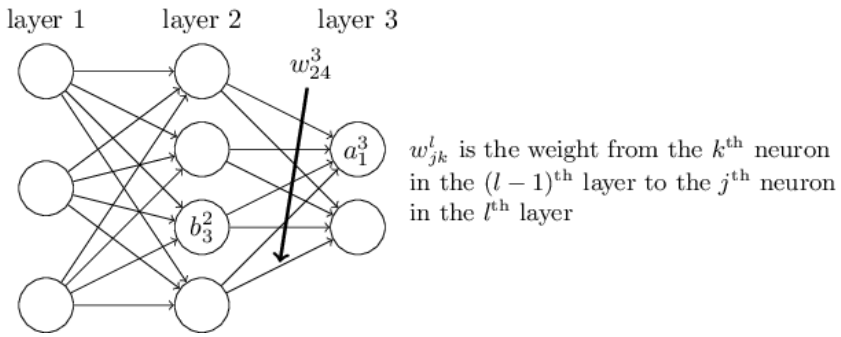


Figure 3.6: Visual representation of the terminology for a neural network [48].

We remove the indexes for the node numbers which results in the following vector notations:

$$a^l = \sigma(w^l a^{l-1} + b^l) = \sigma(z^l) \tag{3.2}$$

Training a network

To train a neural network, we input an example with a known output (or also called label). The network calculates a certain output based on the current weights, which is also called a forward pass. When this output is incorrect, it should be possible to adjust the weights to the effect that the network has as output the correct label, which is also called the backward pass. Note that the change in weights, should only affect the output in a limited way (see figure 3.7). The reason for this is that otherwise all the previous inputs could be labeled incorrectly. So, the concept of training a neural network means: adjusting the weights in a way that the behavior of the network is stable for the previous examples but such that the current example is labeled correctly.

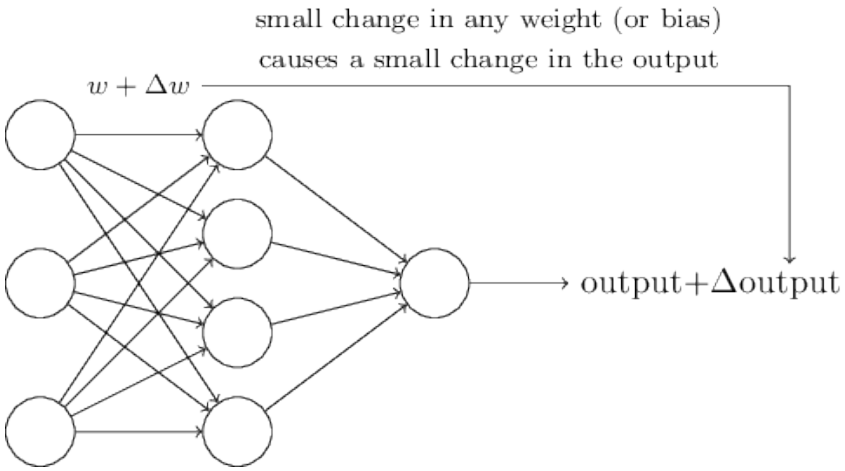


FIGURE 3.7: Change in weights, with respect to the impact on the output [48].

To achieve this effect, sigmoid neurons were adopted instead of perceptrons. A sigmoid neuron has the same basic behavior as a perceptron, but has as activation function the sigmoid function instead of the step function, see figure 3.8. A sigmoid function is easy to work with as it is bounded, easy differentiable, and monotonic [22].
We also introduce a bias b which we ignored in the explanation of the perceptrons. Weights can now range between 0 and 1 while the output is calculated with $\sigma(w*x+b)$ where σ is the sigmoid function. This results in the following formula:

$$output = \sigma(w * x + b) = \frac{1}{1 + exp(-\sum_j w_j x_j - b)} \tag{3.3}$$

We approximate $\Delta output$ in function of Δw_j and Δb:

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \tag{3.4}$$

Because of the linearity and the smoothness of the sigmoid function, it is now easier to choose changes for the weights and biases to achieve a correct output. By adjusting the weights, we train our network to achieve a higher accuracy on the examples seen.

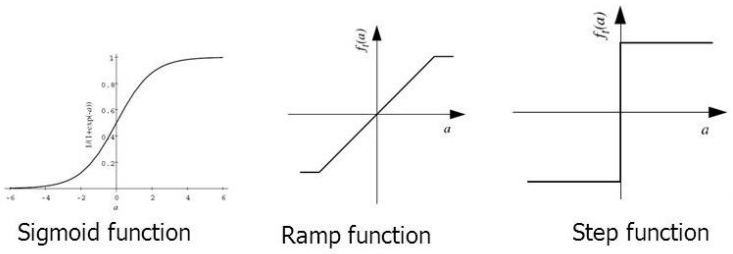


Figure 3.8: Different activation functions.

3.2.6 Backpropagation

To train a neural network, we define a function which has to be minimized. This function is called the cost function. Backpropagation aims to efficiently calculate the gradient of the chosen cost function with respect to the individual weights and biases [57]. Backpropagation can be used in combination with an optimization technique called gradient descent. Based on the gradient, gradient descent updates the weights of the neural network and minimizes the cost function [48].

Cost Function

As mentioned before, backpropagation aims to calculate the gradient of the cost function C with respect to each weight and bias. For a neural network, we can choose any cost function which fulfills the following mentioned criteria.
The first one is that it needs to be possible to write it as a summation over cost functions for individual training examples. Secondly, it needs to be derivable. And lastly, the cost function is a function of the activations of the last layer.
An example cost function which we will use is:

$$C = \frac{1}{2} \sum_j (y_j - a_j^L)^2 \text{ with L the last layer} \tag{3.5}$$

Fundamental Equations

Backpropagation has 4 equations. The equations allow us to calculate the error and the gradient of the cost function. These are used to adjust the weights and biases based on the gradient descent algorithm.

First we calculate the error for each node in the last layer. The error is based on how fast the cost function is changing as a function of the j^{th} output activation and on how fast the activation function σ is changing at z_j^L:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \text{ with } z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L \text{ and } \sigma' the derivative of \sigma \tag{3.6}$$

This can be written as a neat vector equation:

$$\delta^L = (a^L - y) \circ \sigma'(z_j^L) \tag{3.7}$$

The next equation explains why the algorithm is called backpropagation. The equation calculates each layer's error vector based on the next layer and so propagates the error back through the layers:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \circ \sigma'(z_j^L) \tag{3.8}$$

With those 2 equations we calculate the error in each layer of the neural network. Those errors can be used to calculate the derivatives of the cost function with respect to the weights and the biases:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l. \tag{3.9}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{3.10}$$

When the derivatives are calculated, we can apply the gradient descent algorithm and update the weights and biases accordingly. This process represents the learning of a neural network.

3.3 Word2Vec

3.3.1 Motivation

In this section, we explain Word2Vec [46] from a linguistic point of view. Word2Vec lies at the basis of our generalized extension which can be applied to medical data. Word2Vec is introduced in 2015 by the Google Brain Team and is currently an active research field.

In natural language processing tasks, a good representation of words helps algorithms perform better. Word2Vec learns a representation which maps words to vectors in a low-dimensional space compared to the vocabulary size. In this representation, context-similar words are mapped close to each other in the new vector space. The new representation is called a 'word embedding'. We could say in an informal way: a linguistic background is learned and can be used by other algorithms to increase their performance.

A sentence can be seen as a time series of words. Therefore it is possible to generalize Word2Vec to medical data (which is also a time series). As we apply a generalized Word2Vec to medical data, we call this a disease embedding.

3.3.2 Neural Networks

When the Word2Vec algorithm is trained using the Skip-Gram model (see section 3.3.3), one relies on a lookup table. This table contains the mapping of words to their vector representation. This lookup table can be found by training a 2-layer neural network with as cost function the function described in 3.14. The training can be done with backpropagation for example.
The trained 2-layer neural network can be placed in front of another neural network [50]. The former converts the words to their vector representation and feeds it into the latter neural network. It is empirically shown that the results of the neural network can be improve by putting the lookup table in front of it [46]. As mentioned before, in a way, you offer background knowledge to the neural network.

3.3.3 Skip-Gram Model

There are two main models used for Word2Vec, namely Continuous Bag-of-Words (CBOW) and the Skip-Gram model.
The former predicts a word based on a given context (ex. predict Paris when capital France is given). The latter does the inverse of this approach [54]. Empirical results have shown that the Skip-Gram model tends to perform better on larger datasets [63] and establishes a better representation for infrequent words [1]. In medical data infrequent diagnoses may be important along a patient's trajectory. For those reasons, we choose use the Skip-Gram model in the rest of our work.

Learning a Word2Vec representation of a corpus $Text$ with the Skip-Gram model proceeds as follows.
Based on given words w and their contexts c, we set the parameter θ of the probability $p(c|w;\theta)$ to maximize equation 3.11. The probability $p(c|w;\theta)$ is indeed the probability of a context occurring with word w as mentioned before. We refer to section 3.3.3 for more information about θ.

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(c|w;\theta) \tag{3.11}$$

with D the set of all word-context pairs we extract from the corpus.

Establishing word-context pairs

Given a sequence of words, we define their context based on n-grams [29]. In figure 3.9, n-grams are shown for the sentence "This is a sentence".

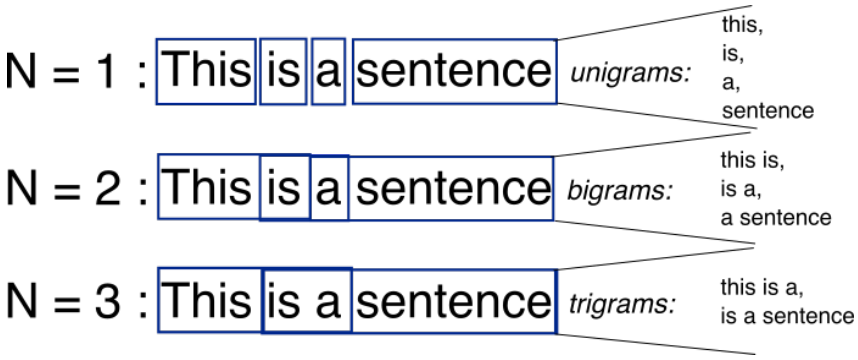


FIGURE 3.9: Explanation of n-gram [4]

For the n-gram, we define the context of a word w_t as $w_{t\pm n}$. A larger n typically results in more specific contexts for training examples and thus can lead to a higher accuracy, if there is enough data to back this up. Indeed, if there are not enough cases handling those specific contexts, the correlation between the word and his context is less statistically significant.

Parameterization by θ

We start by rewriting the conditional probability using a soft-max function [21]:

$$p(c|w;\theta) = \frac{e^{v_c * v_w}}{\sum_{c' \in C} e^{v_{c'} * v_w}} \tag{3.12}$$

where v_c and v_w are vector representations for context c and word w, and C is the set of all available contexts. Because we want to maximize $p(c|w;\theta)$, it means that θ represents the parameters v_{c_i} and v_{w_i}. Computing the optimal parameters is very expensive because you need to calculate this over all contexts $c' \in C$. We also switch from product to sum by taking the logs:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log p(c|w;\theta) = \sum_{(w,c)\in D} (\log e^{v_c * v_w} - \log \sum_{c' \in C} e^{v_{c'} * v_w}) \tag{3.13}$$

Negative Sampling

To compute the hyperparameters $\theta = \{v_c, v_w\}$ using the Skip-Gram model more efficiently, we introduce negative sampling [26].
Instead of calculating $\sum_{c' \in C} e^{v_{c'} * v_w}$ over all contexts, we make a set D' which consists of randomly sampled word-context pairs. With this new set, we remove the costly term $\sum_{c' \in C} e^{v_{c'} * v_w}$ and replace it with $\sum_{(w,c)\in D'} e^{v_{c'} * v_w}$:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log p(c|w;\theta) = \sum_{(w,c)\in D} (\log e^{v_c * v_w} - \log \sum_{(w,c)\in D'} e^{v_{c'} * v_w}) \tag{3.14}$$

Informally, while not ensuring that if words appear in the same context, their vectors are closer together than all the other word vectors, we only make sure they are more similar than a several vectors chosen randomly. Negative sampling makes the Skip-gram model usable in terms of computation.

3.4 DeepWalk

DeepWalk is an approach for applying Word2Vec on graph-structured data [53]. DeepWalk generates random sequences based on the graph structure. Word2Vec is then applied on those sequences to learn a good vector representation for the vertices. We can say that it is an extension to the Word2Vec approach by allowing it to be applied to more varied types of data.

In figure 3.10, we see an overview of the DeepWalk algorithm. It consists of two parts.
First, in line 6, a random walk generator generates for each vertex v_i of the graph G, a random walk of length t. It will do this γ times but the order $\mathcal{O}$ in which the vertices are traversed, is randomly chosen for each pass. With those walks, a sequence of vertices is generated.
Secondly, in line 7, this sequence is used for Word2Vec with the Skip-Gram model For this SKip-Gram model we use w as the size for the n-grams (also called window size) and d as the dimension of the new vector space. This process is explained in section 3.3.

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$
1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\mathcal{O} = \text{Shuffle}(V)$
5: **for each** $v_i \in \mathcal{O}$ **do**
6: $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
7: SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
8: **end for**
9: **end for**

Figure 3.10: Overview of the DeepWalk algorithm [53]

3.5 Generalized Word2Vec Approaches

In this section we explain our approach on how to find patterns in EHRs using Word2Vec techniques. For this, we formulate a generalized Word2Vec approach tailored to the context of learning disease embeddings.

3.5.1 Data representation

Before we can start with explaining how Word2Vec can be applied on EHR data, we introduce our data representation.
The medical history of a patient is a time series of EHR events. On certain timestamps, an EHR event is available containing the medical info (diagnose, lab result, ...) about that patient. We denote this EHR event the vector m_t^p, with p a patient number and t a timestamp. This means that each patient has a sequence of vectors $s^p = m_t^p, m_{t+1}^p, m_{t+3}^p, \ldots$ Each vector m_t^p contains certain values depending on the event. It can contain values of the event's time, demographics, blood pressure, diagnoses, and others (see figure 2.1.

3.5.2 Generalized Word2Vec

As explained in section 3.3, Word2Vec is typically applied to a large text corpus containing a large amount of sentences. After applying this method, an embedding is found that represents words in a new vector space. In this vector space the relationship between words is shown by the distance of the words from each other.

A medical dataset containing EHRs from different patients can be seen as a large text corpus. It contains patients, each with an EHR (or a sequence of EHR events) s^p, which is equivalent to one sentence. The set $\{s^p\}$ is considered as a corpus of sentences made up of words m_t^p. Each vector m_t^p is removed from the features which uniquely link it to the patient like the timestamp or patient identifier, we call this trimmed vector m. This is to make the link to words. Different sentences can contain the same words in that vectors m can be part of different sequences.

With the above, it is possible to apply Word2Vec on corpus $\{s^p\}$. We call this a generalized Word2Vec approach. From this, we can learn an embedding for the vectors m. The embedding shows the relationships between the vectors m with others in the sequences $\{s^p\}$ based on their contexts.
Others have explored other analogies with for example protein sequences (BioVec) [9]. Note that with BioVec, their vectors m are still the same as words as they are both one-dimensional.

3.5.3 K-Nearest Neighbors Word2Vec

A problem with embeddings established for a certain dataset is that because Word2Vec builds a lookup table, there is the possibility that a certain instance is not present in the lookup table and therefore no representation can be found for this instance.

Since we are working in the context of a generalized Word2Vec approach, where instances are represented by complex objects like vectors m, the possibility of this happening increases. For example, an EHR event can be a multi-dimensional vector where each feature has a broad range of values. Compared to a word, which is a one-dimensional vector with as range all words in a dictionary, an EHR event has more possible combinations.

Therefore we introduce a k-Nearest Word2Vec approach whereby we extend our generalized Word2Vec approach with a knn feature (see section 3.2.3). To our knowledge, this has not been done before.

After the training of a Word2Vec model, we have learned a lookup table which maps a known vector v_{old} to his new vector representation v_{new}. When a not-yet-seen vector $v_{unknown}$ needs to be mapped to his v_{new}, a normal Word2Vec is not capable of this.
In our approach, we find the k-nearest neighbors of the $v_{unknown}$ from the representations v_{old} in the lookup table. We then take the new representations of the knn, and combine them using a weighted average to estimate the new representation v_{new} for $v_{unknown}$. This way our approach is capable of finding a representation for complete new instances.

In short: we look for the knn of the $v_{unknown}$ from all the known vectors in the lookup table in their original representation v_{old}. Based on the found knn, we take a weighted average of the representations v_{new}. This weighted average is the v_{new} for the $v_{unknown}$.

3.5.4 Generalized Deepwalk

Deepwalk starts from graph-structured data and is therefore not directly applicable to EHR data. One advantages of Deepwalk that it provides a method which can control a fixed amount of sequences. When the EHR dataset becomes very large, it would take a considerable amount of time to train a Word2Vec model.

Therefore it would be interesting to transform the EHR data into a weighted graph structure. The weights are based on the frequency of diagnoses following each other in all the sequences.
After the graph transformation, the amount of weighted random walks can be limited to create a smaller set of sequences based on the original dataset. Those sequences can be used to train a Word2Vec model faster as there is fewer data. The same logic from the previous two sections is applicable to generalize DeepWalk from words to more abstract objects.

Graph Transformation

In algorithm 3.5.4, we explain the graph transformation.
To execute the graph transformation, we start by looping over each sequence from $\{s^p\}$. For each sequence we keep track of the previous seen element and the current element. From those 2 elements we make 2 vertices connected by an edge in line 6. We add this edge to the current graph in line 7:

- If the edge between those two vertices already exists in the graph, we increase the weight of the existing edge by one.
- If the edge does not exist in the current graph, but one of the vertices does exist, we add a new edge between the existing vertex and the newly added vertex. The weight of this new edge equals to one.
- If the edge does not exist and the two vertices also do not exist in the graph, we add them all to the graph.

Algorithm 1 Graph Transformation

```
1: output: Graph
2: for all Sequence in Sequences do
3:     for all Element in Sequence do
4:         PreviousElement
5:         CurrentElement
6:         Edge = MakeEdge(PreviousElement, CurrentElement)
7:         Graph.addEdge(Edge)
8:     end for
9: end for
```

3.6 Conclusion

In this chapter we discusses general machine learning concepts and focused on Word2Vec. We conclude that Word2Vec is used to find good representation of words based on their context. It also causes that similar words will be close to each other in this new representation.

With the concepts explained, we introduced our generalized Word2Vec approaches with as goal to find patterns in EHRs. We extended Word2Vec so that it can handle more abstract objects than words, namely vectors representing an EHR event. We also extended Word2Vec to find a representation for a new instance based on knn. We also generalized DeepWalk to reduce the training time of Word2Vec on large medical datasets.

Chapter 4

Word2Vec Model Building

4.1 Introduction

In this chapter
DiseaseMapping (generalization) OSIM Clusters TensorFlow DL4J

4.2 OSIM2 Dataset

To validate our approaches mentioned in section 3.5, we rely on the Observational Medical Dataset Simulator Generation 2 (OSIM2) dataset [49]. This dataset was generated by the Observational Medical Outcomes Partnership (OMOP) as a golden standard to be able to reproduce studies about the effects of drug treatments. It contains around 10 million of hypothetical patients based on Thomson Reuters MarketScan Lab Database (MSLR). MSLR contains administrative claims between 2003 and 2009 from a privately-insured population. There are 16 different OSIM2 datasets where each of them is injected with different kinds of signals. We rely on the "OSIM2_10M_MSLR_MEDDRA_14".

The OSIM2 dataset contains multiple database tables which are dumped as comma-separated values (csv) files. The files contain information about the patients' demographics, diagnoses, drug treatments, timestamps, hospital visits, ...
To make it easier to work with this dataset, we joined the multiple files into one file with on each row an event of a patient containing all relevant information. The relevant information which is kept is: birth year, gender, condition type, condition (MedDRA coding), time difference since previous diagnosis, and season (summer, fall, winter, spring). We ignored drug treatments and hospital visits as those are too complex to categorize. Thus, one EHR event (or vector m) is 6 dimensional, see table 4.4.1 for an example.

We will compare our model with the clusters found in Anders Boeck Jensen et. [34] (see section 2.3.3). Although these clusters are not a golden standard, it is the only study to which we can compare our model. It provides an initial idea on how

well our model estimates disease relations. In section 6.3, we introduce a validation approach which can be explored in the future.
Before we can compare our model to the clusters found in the Danish paper, we have to apply several estimations and methods which are described in section 4.4.

4.3 Software Ecosystem

To build our model, we look for a software library which provides us with the following tools:

- Word2Vec implementation
- Highly customizable as we want to adjust internal workings of Word2Vec
- Built with large datasets in mind (no memory leaks, multi-threading)

4.3.1 TensorFlow

TensorFlow is an open-source machine learning software library released at the end of 2015 [8]. It is developed by the Google Brain Team. It puts its focus on neural networks and their applications in language processing and image recognition.
It provides a Python interface for efficient C++ code. We found that Tensorflow is not well documented and does not provide the freedom needed to easily rewrite some core features of the Word2Vec implementation, for example manipulating the internal trained lookup table to add new instances based on knn.

4.3.2 DeepLearning4Java

DeepLearning4Java (DL4J) is an open-source machine learning software library released by Skymind [62]. It puts its focus on deeplearning and makes it possible to easily extend DL4J's implementation with the user's implementation.
It runs on their scientific computing engine ND4J which provides fast matrix operations. DL4J is completely written in Java and provides a lot of freedom to manipulate lookup tables and extend Word2Vec methods to work on abstract objects such as vectors.

4.4 Generalized Word2Vec Model Building

As described in section 3.5, we use generalized Word2Vec approaches to find patterns in EHR data. We use the OSIM2 dataset and represent each EHR event as a 6 dimensional vector. This vector is comparable to a word in a normal Word2Vec approach and functions as the abstract object in our generalized Word2Vec approaches.

4.4.1 Categorization of Features

Because we are working with high dimensional data and a each dimension has a wide range of possible values, most instances of OSIM2 are unique. To find patterns which are more generally applicable, we generalize our data by projecting values for some attributes into categories. See table 4.4.1.

Attribute	Requirement	Category
Time between EHR events	≤ 0	0
	≤ 10	1
	≤ 20	2
	$20 <$	3
Birthyear	≤ 1900	0
	≤ 1910	1
	≤ 1920	2
	$\vdots$	$\vdots$
	≤ 2020	12
Season		3 (winter)
		4 (spring)
		1 (summer)
		2 (fall)

TABLE 4.1: Different categories for OSIM2 attributes.

It is easy to generalize concepts such as time intervals and demographics, for example we can say that people between the age 0 and 10 belong to category A. This becomes complex for disease diagnoses as it requires a lot of knowledge to categorize those. We come back to disease coding in section 4.4.2.

To see the effect of our categorizations, see table 4.4.1. You can see the 3 most common vectors from our dataset before and after the categorizations. The vectors have the following structure: *[birthyear, gender, conditionType, diagnose, timeDifference, season].*

4.4.2 Disease Code Categorization and Mapping

In the section we discuss the method to categorize the disease codes in the OSIM2 dataset. For example, a bruise on your right leg and a bruise on your left leg should both be categorized under a bruises category.

The OSIM2 dataset uses MedDRA disease codes for the diagnoses, see section 2.2.1. We mentioned that MedDRA does not have an easy to use hierarchy. With a hierarchy, it would be trivial to categorize a disease code to its highest hierarchy.

Before Categorization		
Vector	*Occurrences*	*Percentage*
[1956.0, 8532.0, 65.0, 5.00000701E8, 0.0, 3.0]	17574	0.0095
[1954.0, 8532.0, 65.0, 5.00000701E8, 0.0, 3.0]	17536	0.0094
[1955.0, 8532.0, 65.0, 5.00000701E8, 0.0, 3.0]	17476	0.0094

After Categorization		
Vector	*Occurrences*	*Percentage*
[6.0, 8532.0, 65.0, 784955.0, 1.0, 3.0]	282086	0.15
[7.0, 8532.0, 65.0, 784955.0, 1.0, 3.0]	235459	0.13
[5.0, 8532.0, 65.0, 784955.0, 1.0, 3.0]	230216	0.12

Table 4.2: Three most common vectors in our dataset before and after categorization.

To solve this, we map MedDRA disease codes to ICD-10 disease codes. The reasoning behind this is that ICD-10 provides a clear and easy to use hierarchy. Next to this, ICD-10 disease codes also make it possible to compare our results with [34].
We would have liked to use the method described in [19] to map the codes but therefore the disease mapping made by UMLS between ICD-10 and MedDRA is needed. This however is not publicly available.

The mapping is based on the description of each disease code. Both MedDRA and ICD-10 have a short medical description of each code. Each description is first filtered from stop words. Afterwards, the MedDRA code is matched to the ICD-10 code based on the matching words in both descriptions. The matching is defined as the ICD-10 code with the highest percentage of words matching.

In figure 4.1, we show the statistics of this mapping process. Note that we only take disease codes into account if they are part of the OSIM2 dataset. In the figure, each bar represents the percentage of the words matching between descriptions. The height of the bars represent the percentage of all disease codes in the OSIM2 dataset which have this amount of word matches.

We find a 63% average of the words in the description that match between MedDRA and ICD-10 descriptions. We also see that around 85% of all the disease code mappings have a match of at least 33%. We assume this is a reasonable mapping as medical terms are quite specific and if 33% matches, this corresponds to a match on to a higher hierarchical level.
For the codes which have a zero percent match, the Damerau-Levenshtein algorithm [10] is applied. The algorithm calculates the edit distance between two strings using character insertion, character deletion, character replacement, and adjacent character swaps. The description with the lowest edit distance is then chosen as a match.
Another option would have been to remove those codes from our dataset. As an

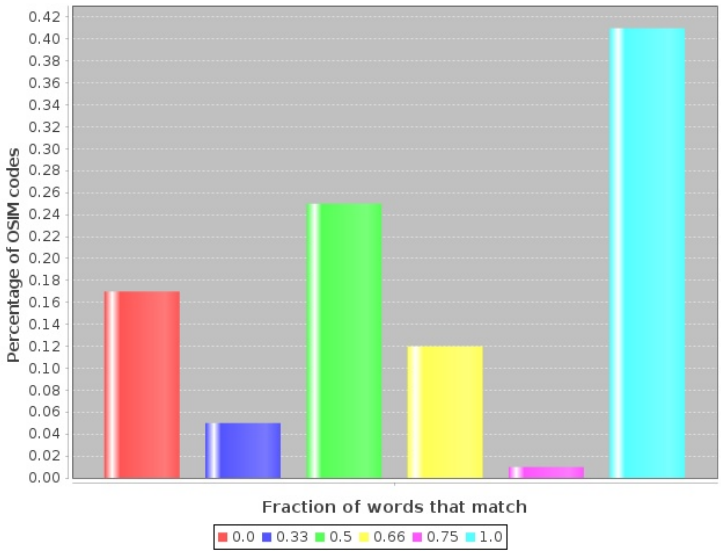


FIGURE 4.1: Percentage of OSIM codes compared to their fraction of their matching words.

EHR contains more information than only the disease code, a lot of information would have been lost. This however is not tested and is a possibility for future work.

With the mapping, we can now categorize the disease codes. We use the hierarchy of ICD-10 and reduce the ICD-10 code to its first 3 symbols. For example, we reduce the code *E00.1* to its higher category *E00*.

4.4.3 Comparison with Danish Clusters

To test our approaches we compare our results with the clusters found from Anders Boeck Jensen et. [34], we call these the Danish clusters. For this, we first need to generate our own clusters from OSIM2. This method is described in this section. Note that the comparison with the Danish paper is not ideal as mentioned in section 4.2. We refer to chapter 6 for information about another possible experiment in the future.

We now explain how we construct clusters from OSIM2 based on our Word2Vec model, we call these Word2Vec clusters. Afterwards, we compare the Word2Vec clusters with the Danish cluster and define a matching percentage between the two.

First we apply our generalized Word2Vec approaches on the OSIM2 dataset. For a description of the parameters used and their functions, see section 4.4.4. As end result, we obtain a lookup table containing the mapping between the old vector space and the new one. From this lookup table, we take EHR events and find their k-nearest neighbors from the other events in the new vector space (see section 4.4.4 for more information about clusterK). The Word2Vec clusters are the set of the EHR

event and its knn.
Depending on the approach, we take all or a subset of the EHRs.

We now describe 2 experiments which find a matching percentage between the two kinds of clusters.
The first experiment goes as follows:

- We start with an element e_{wv} from the lookup table and form a Word2Vec cluster C_{wv} around it as described above.
- We then check to which Danish cluster C_d element e_{wv} belongs.
- For each element in C_{wv}, we check if it belongs to C_d.
  - If it does, we add 1 to the number $t_{matches}$.
  - If it does not, we add 1 to the number $t_{non\_matches}$.
- After doing this for all elements in C_{wv}, we calculate the matching percentage as $p_{match} = \frac{t\_matches}{t_{matches}+t_{non\_matches}} = \frac{t\_matches}{|C_{wv}|}$.

The second experiment goes as follows:

- We start with an element e_{wv} from the lookup table and form a Word2Vec cluster C_{wv} around it as described above.
- We then check to which Danish cluster C_d element e_{wv} belongs.
- We set t_d equals to the number of elements in C_d.
- For each element in C_{wv}, we check if it belongs to C_d.
  - If it does, we add 1 to the number $t_{matches}$.
- After doing this for all elements in C_{wv}, we calculate the match percentage as $p_{match} = \frac{t\_matches}{t_d} = \frac{t\_matches}{|C_d|}$.

For each method, we calculate the average p_{match} over all elements in the lookup table and keep track of the maximum and minimum value for p_{match}.

4.4.4 Parameters

The effectiveness of a neural network such as the one used to learn disease embeddings, depends on parameter tuning. Parameter tuning is a difficult problem and complex methods have been proposed to solve this [11]. Therefore we explored 504 different settings. Note that this is not an extensive parameter tuning setup, but an overview on the effect and logic behind some parameters. An extensive parameter tuning setup is possible as future work.

In table 4.4.4, you can see an overview of the different parameters for each approach. Every time, all possible combinations of the mentioned parameters have been tested.

Parameter	Generalized Word2Vec	Knn Word2Vec	DeepWalk
Vectorlength	[50, 100]	[50, 100]	[50, 100]
Batch Size	500	500	500
Epoch	1	1	1
Window Size	[5, 10, 15]	[5, 10, 15]	[5, 10, 15]
Learning Rate	[0.025, 0.1]	[0.025, 0.1]	[0.025, 0.1]
Minimum Word Frequency	[5, 10]	[5, 10]	[5, 10]
ClusterK	[100, 1000, 5000]	[100, 1000, 5000]	[100, 1000, 5000]
K	/	[10, 50, 100]	/
Walklength	/	/	[5, 10, 15]

TABLE 4.3: Parameters which are tested for the different approaches

Vectorlength fixes the dimensions of the new vector space to which the EHR events are projected to. A larger size is more expressive but a smaller size decreases the complexity of the vocabulary after training. Each dimension can be seen as a representation of an unknown linear relation to some context-disease pairs [41].
Batch size is the number of training examples used in one gradient descent step.
Epoch is the number of times you go over all the training examples and use them to train your neural network by updating the weights.
Window size is the size of the n-gram we use to create contexts. A window size is an estimation of the relevant context for a word. This means that the window size depends on the used corpus [46] [40].
Learning rate decides how the weights are adjusted during backpropagation. A large value makes the neural network learn faster but may also cause the network not to learn at all. A small value on the other hand can cause a slow convergence or overfitting [39].
Note that in our implementation, we use an adaptive gradient descent algorithm [23]. Informally, it increases the learning rate for sparse features and therefore is able to take predictive but rarely seen features into account.
Minimum Word Frequency removes instances from the training set below a certain minimum value. A low frequency causes the inclusion of instances which only have a few contexts for which relations can be learned from. A higher frequency could improve the accuracy for a Word2Vec model but also throw away information. Note that the influence of this parameter is not well researched.
ClusterK is the parameter which creates Word2Vec clusters as described in the previous section. A higher clusterK causes the Word2Vec cluster to be larger and a lower value the other way around.
K is a specific parameter for the knn Word2Vec approach. It decides on how many nearest neighbors the new vector representation is based on. For more explanation

see section 3.5.3.
Walklength is a specific parameter for DeepWalk. It decides how long the generated sequence will be. For more explanation see section 3.4.

4.5 Results

4.5.1 Generalized Word2Vec

Here we discuss the results from our two experiments on the generalized Word2Vec approach. We trained a Word2Vec model on the complete OSIM2 dataset. From this model, we compare the Word2Vec clusters with the Danish cluster as described in section 4.4.3. For each parameter, we fix the other parameters and check their influence on the matching percentage. After checking the individual parameters, we take the parameters setting with the best matching percentage and interpret those results.

Vectorlength

Both in experiment 1 and experiment 2, we found that the vectorlength had no substantial influence on the matching percentage. See figure 4.2 for the matching percentage of experiment 1 and figure 4.3 for experiment 2. The two bars in the figures each show a parameter value and the black stripes both represent the minimum and the maximum value of the matching percentage. Similar results are achieved with all the other parameter settings.

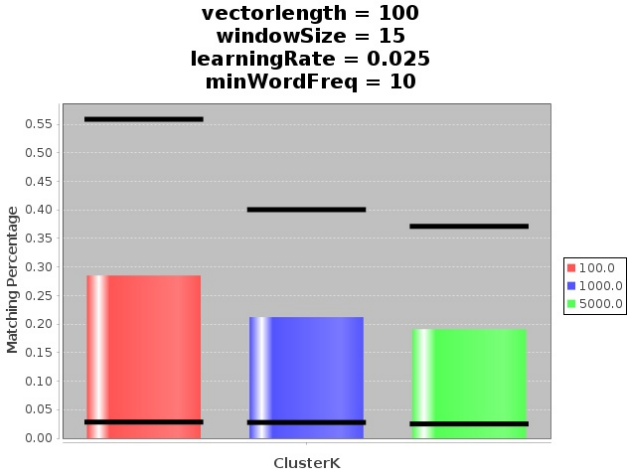


Figure 4.2: Mapping percentage with the vectorlength parameter of generalized Word2Vec approach in experiment 1.

We conclude that a vectorlength of 50 is already expressive enough to differentiate between different diseases. Only sometimes the vectorlength of 100 gives a small gain

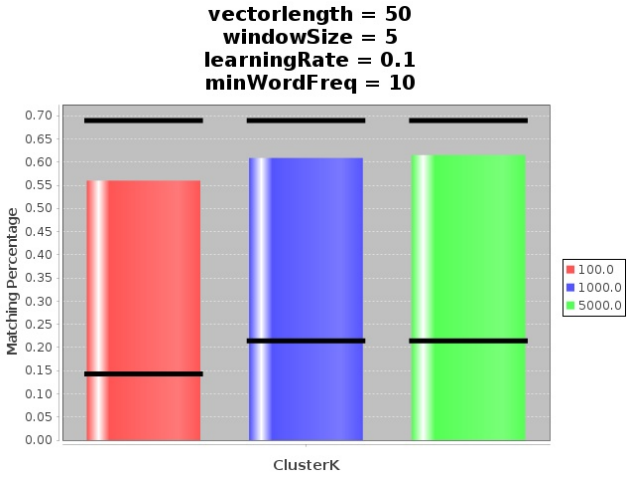


FIGURE 4.3: Mapping percentage with the vectorlength parameter of generalized Word2Vec approach in experiment 2.

in matching percentage which can be explained by the increase of expressiveness of the new vector space.

Window Size

Both in experiment 1 and experiment 2, we found that the window size had no substantial influence on the matching percentage. See figure 4.4 for the matching percentage of experiment 1 and figure 4.5 for experiment 2. Similar results are achieved with all the other parameter settings.

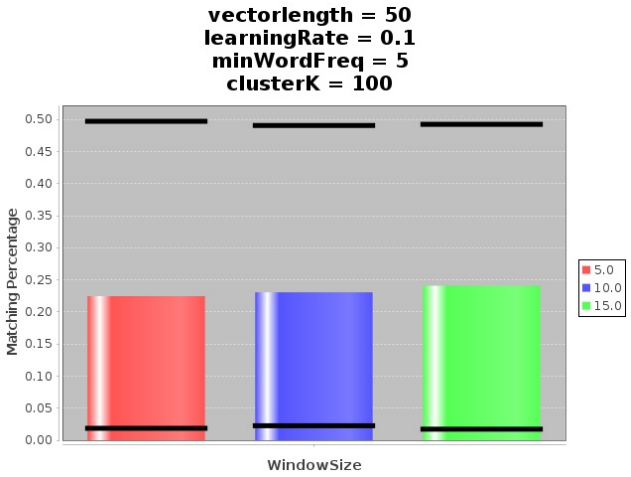


FIGURE 4.4: Mapping percentage with the window size parameter of generalized Word2Vec approach in experiment 1.

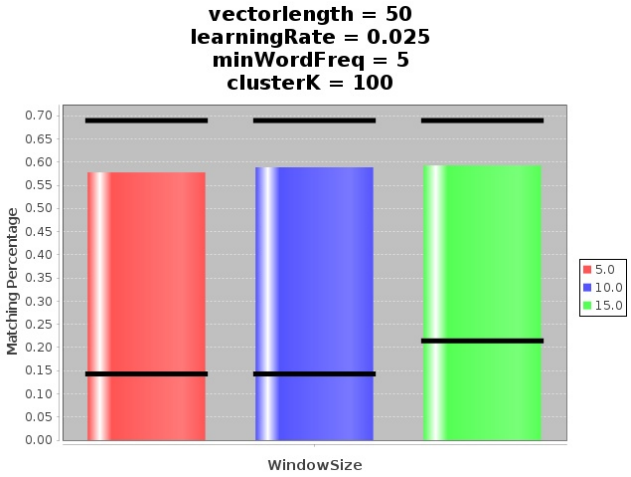


Figure 4.5: Mapping percentage with the window size parameter of generalized Word2Vec approach in experiment 2.

We expected the window size to have more impact as a higher size makes more specific contexts. And more specific contexts, tend to make better contexts. A reason for this could be that we don't have enough data to justify more specific contexts. Also, because our sequence lengths have an average length of around 19, a window size of 5 is not that much different than a size of 10 as in many cases it will cut of the size at the start or end of the sequence. The Danish paper uses sequences of length 4, so this could also imply that a window size of 5 is already enough to cover the relationships the Danish paper found and therefore a higher size will not have an impact on the matching percentage.

Learning Rate

Both in experiment 1 and experiment 2, we found that the learning rate had no substantial influence on the matching percentage. See figure 4.6 for the matching percentage of experiment 1 and figure 4.7 for experiment 2. Similar results are achieved with all the other parameter settings.

Learning rate is a parameter which is hard to tune. This explains the amount of literature around learning rate and the implementation of automatic tuning algorithms for learning rate. Therefore our results do not provide any possible conclusion as we only tested 2 different learning rates.
Although it is also possible that we chose our learning rate high enough in both cases and because of the learning decay, they both achieved the same results.

Minimum Word Frequency

Both in experiment 1 and experiment 2, we found that the minimum word frequency had no substantial influence on the matching percentage. See figure 4.8 for the

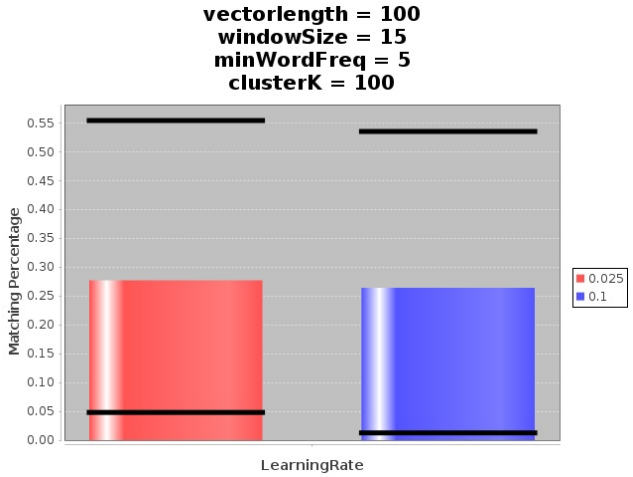


FIGURE 4.6: Mapping percentage with the learning rate parameter of generalized Word2Vec approach in experiment 1.

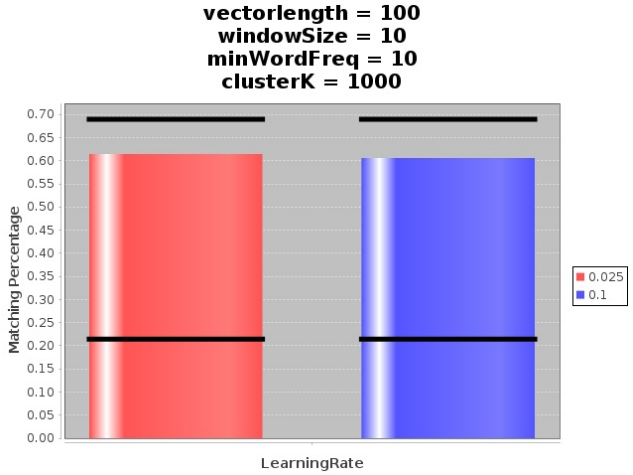


FIGURE 4.7: Mapping percentage with the learning rate parameter of generalized Word2Vec approach in experiment 2.

matching percentage of experiment 1 and figure 4.9 for experiment 2. Similar results are achieved with all the other parameter settings.

We expected the minimum word frequency to have a better accuracy with a higher value as we would be sure they have enough instances to be learned from. Because we applied generalization, 99.82% of the instances are above the minimum word frequency of 10 to 69.90% before generalization. We therefore expect the minimum word frequency to have lost its influence. A future direction to explore is to ignore the generalization and solely work with minimum word frequency.

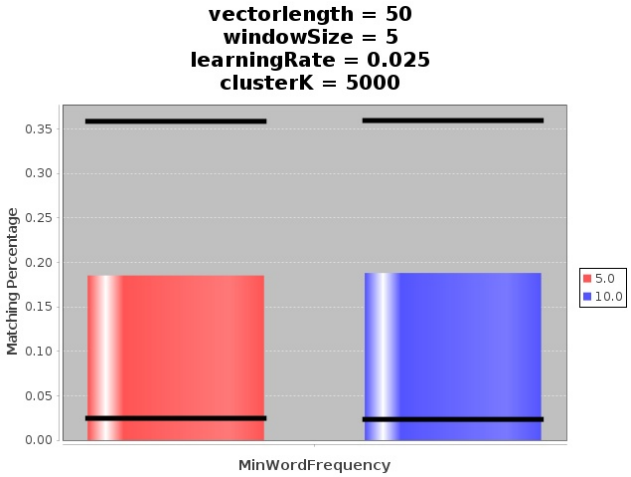


Figure 4.8: Mapping percentage with the minimum word frequency parameter of generalized Word2Vec approach in experiment 1.

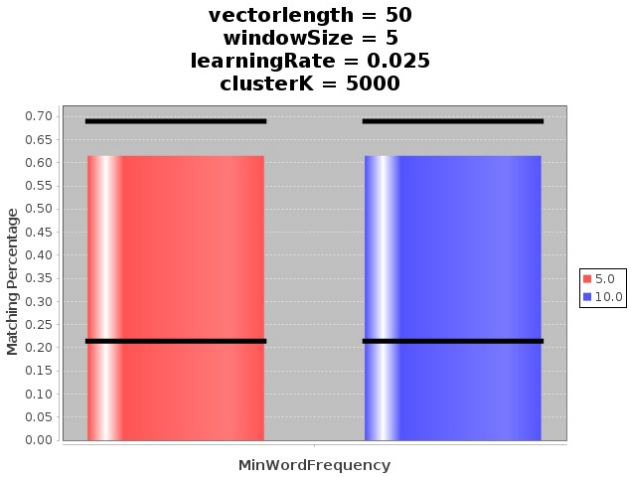


Figure 4.9: Mapping percentage with the minimum word frequency parameter of generalized Word2Vec approach in experiment 2.

ClusterK

See figure 4.10 for the influence in the matching percentage of experiment 1 by the parameter clusterK. Similar results are achieved with all the other parameter settings.

We see a trend that with a lower clusterK there is a higher matching percentage. This means that in the smaller clusters, there is a higher density of instances which correspond to the Danish clusters. This shows that our method is not completely random especially in the case of some disease codes as a maximum matching percentage of 55% is found.

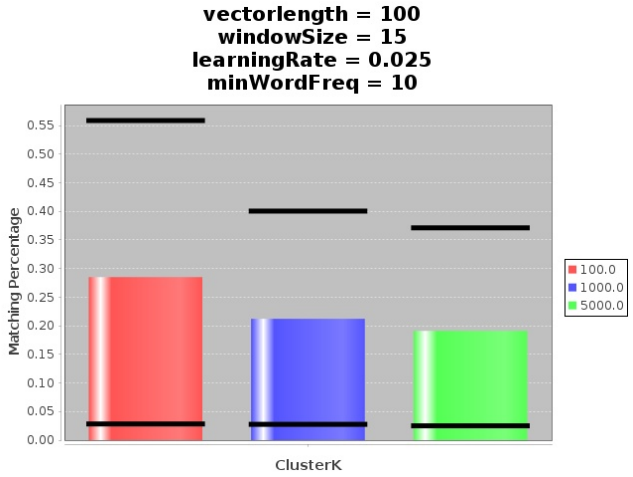


FIGURE 4.10: Mapping percentage with the clusterK parameter of generalized Word2Vec approach in experiment 1.

See figure 4.11 for the influence in the matching percentage of experiment 2 by the parameter clusterK. Similar results are achieved with all the other parameter settings.

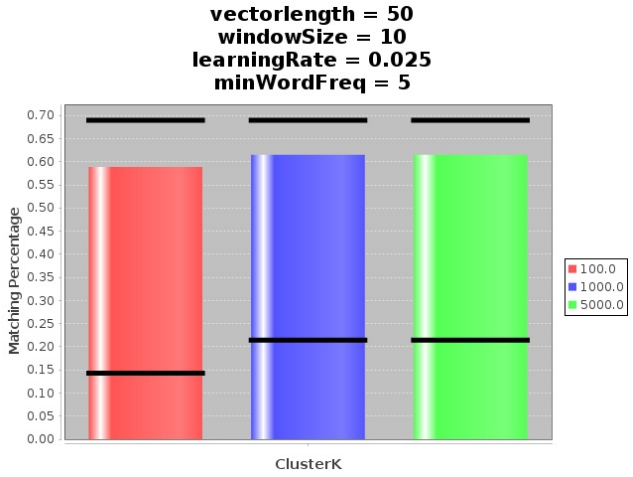


FIGURE 4.11: Mapping percentage with the clusterK parameter of generalized Word2Vec approach in experiment 2.

We see that most matches with the Danish clusters are found in the smaller Word2Vec clusters. This corresponds with the results from experiment 1.

4.5.2 K-Nearest Neighbors Word2Vec

Vectorlength

Similar as generalized word2vec

Window Size

Similar as generalized word2vec

Learning Rate

Similar as generalized word2vec

Minimum Word Frequency

Similar as generalized word2vec

ClusterK

K

TODO

4.5.3 DeepWalk

TODO

Vectorlength

Similar as generalized word2vec

Window Size

Similar as generalized word2vec

Learning Rate

Similar as generalized word2vec

Minimum Word Frequency

Similar as generalized word2vec

ClusterK

TODO

Walklength

Brunak 4

4.5.4 Model Comparison

TODO

4.6 Conclusion

Chapter 5

Conclusion

This thesis started with explaining the new research area of Electronic Health Record Analytics. We explored the possible impact of this area as it allows to find medical patterns on a large scale. Those patterns range from drug discovery, disease progression for individuals, and reducing medical costs. At the moment several research groups are working on utilizing EHRs to find medical patterns using several methods like querying, statistics, data mining, and artificial intelligence approaches.
Our research sought to explore the usage of new machine learning approaches to find correlations between different diagnoses. The correlations found can be used in combination with prediction or classification methods.

We introduced a new way on how to apply Word2Vec methods by explaining the link between sentences of words and sequence of medical records. We call this approach a generalized Word2Vec approach and it can be applied on medical data to the correlations between different diagnoses.
To make sure the generalized Word2Vec methods can be applied to large-scale medical data, we applied the generalization concept also on DeepWalk. DeepWalk makes it possible to generate a smaller dataset from the original dataset and then apply a Word2Vec approach on this smaller dataset.
Besides the exploration on generalizing Word2Vec approaches, we also improved Word2Vec by tackling on of its shortcomings. This shortcoming of Word2Vec is that it is unable to handle unseen instances once it has built his lookup table. We combine a k-nearest neighbors method with Word2Vec and make an estimation of the correlation to other diagnoses for the unseen instance.

EXPERIMENTS + Limitations of the experiment
CONCLUSION

For more information about another possible experiment which also combines the Word2Vec approaches with prediction or classification methods, we refer to the next chapter. In this chapter we also talk about some possible improvements and future directions to explore.

Chapter 6

Future Work

In this chapter we discuss some possible improvements or further research which are possible to add to our approaches.
In section 6.1 we discuss a possible improvement on how we handled categorization of the medical data which we discusses in section 4.4.1. In section 6.2, we introduce a optimization for our Word2Vec approaches by utilizing data distribution. In section 6.3, we explain a specific neural network which can use the found patterns by the Word2Vec approaches to predict or classify patients' disease trajectories.

ADD META ABOUT EXPERIMENTS

6.1 Categorization

In our Word2Vec approaches, we applied categorization on the medical states to reduce the amount of infrequent states. This was needed to retrieve more general n-grams. For this categorization, we divided attributes like age into specific intervals.

Instead of dividing attributes into specific intervals, we could apply normalization to those attributes. Based on the distribution of the data, we can make more sensible categories and assign them to the attributes accordingly.

6.2 Distributed Word2Vec

Word2vec can be made distributed as the underlying idea is quite simplistic, it counts occurrences of n-grams. Counting occurrences based on labels, is a well known problem and is often solved by MapReduce algorithms [18].

6.3 Patient Classification

As mentioned in section 3.3, a trained 2-layer neural network can be placed before another neural network and the former functions as a lookup table. In this section,

we discuss the latter neural network which allows us to further investigate the effectiveness of our Word2Vec approach to better classify or predict patients. More concrete: we could check if a better accuracy is acquired with the lookup table in front of the prediction/classification neural network or without.

6.3.1 Problem Definition

The medical history of a patient is seen as a time series with as datapoints EHR events. Based on the time series, we want to classify it into different disease trajectories. A patient who is classified into a specific disease trajectory, can be treated more specifically as we would have a better view what the next stages of the trajectory could be.

The medical data of multiple patients is a 3 dimensional tensor (multi-dimensional matrix), see figure 6.1. This data structure is the input structure for a neural network.

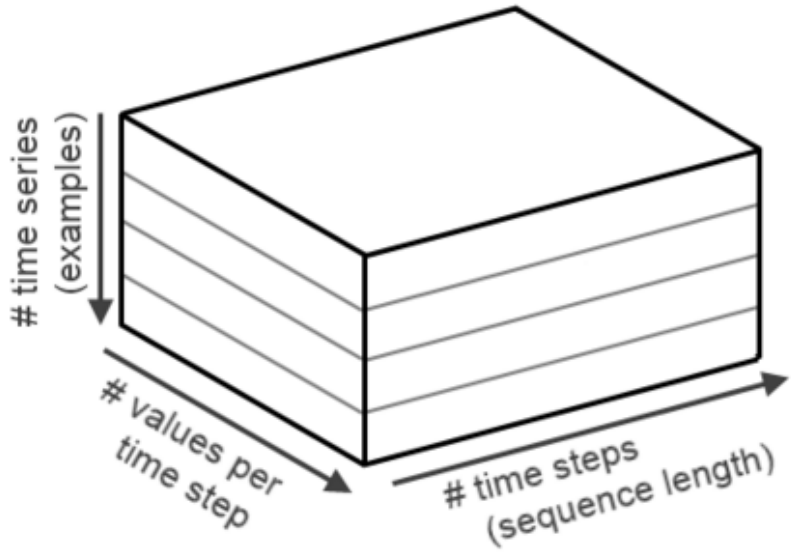


Figure 6.1: Overview of the data structure for medical data with a time aspect [6]

Medical data has some problems which we will discuss.
It often consists of long time periods. This means there could be a long range of dependencies between events. In the context of training neural networks, this can cause a problem known as the vanishing gradient problem [52].
Patients do not have regular intervals in their medical data. The irregular intervals need to be transformed into regular intervals otherwise the time aspect will not be consistent throughout the data.
The standardization of the attributes needs to be taken into account. Preferably some sort of normalization should be applied as well.

Medical data has a high dimensionality. A lot of parameters need to be taken into account to retrieve accurate results. This causes a well known problem: Curse of Dimensionality [37]. It causes the data to be sparse and have more infrequent cases. Therefore, more data is needed to cover all cases. Especially in medical data where outliers are important.

6.3.2 Approach

Here we describe our approaches for the problems mentioned in the previous section. We solve the vanishing gradient problem with a special form of recurrent neural network, see section 6.3.3.
By applying our Word2Vec approach, the input is projected on another vector space and results into a lookup table. The standardization and normalization is already done by our Word2Vec approaches.
As we mentioned, the Curse of Dimensionality causes the need for more data. The neural network in section 6.3.3 often handles high dimensional data [15] [43] [56] [44]. In a sense, because it keeps track of the time aspect of the data, it uses the data more thoroughly and thus handles the high dimensionality better.

A lot of these problems are covered in an extensive work of Graves [27] which covers the use of advanced neural networks to label sequences.

Padding and Masking

The transformation of irregular intervals into a regular ones, is done with padding and masking.

If we do not use any masking and padding, our data can only be of equal length sequences and at each time step a classification label. Our data on the other hand, consists of several different length sequences and only has one label for each sequence, namely the classification label of the whole sequence.

The method of padding is simple. It adds empty events (ex. zeros) to the shorter sequences until all sequences are of equal length for both input and output.
Padding changes the data quite drastically and this would cause problems during the training because the network does not know which elements are padding and which are not padding. For this problem we use the method of masking. With masking, we use two additional arrays which contain the information about whether an input or output is padding or not. See figure 6.2, the second figure shows how masking is done for a many to one case (which is the case that corresponds to labeling a complete sequence).

6.3.3 Neural Network

In this section we explain in more detail why Long-Short Term Memory (LSTM) [31] networks handle the vanishing problem and long-term dependencies.

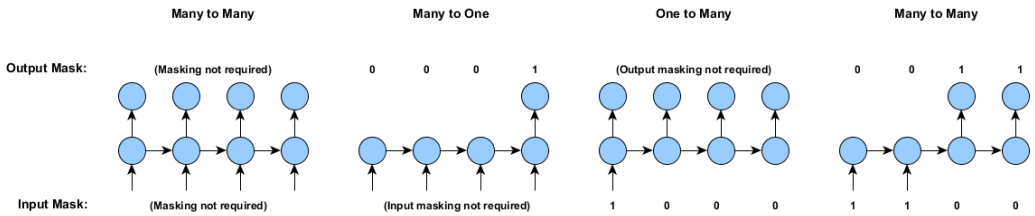


Figure 6.2: Multiple masking methods [6]

First we shortly repeat the structure of a neural network in figure 6.3. We see the input x, different layers with their neurons, σ as the activation function, and y as output.

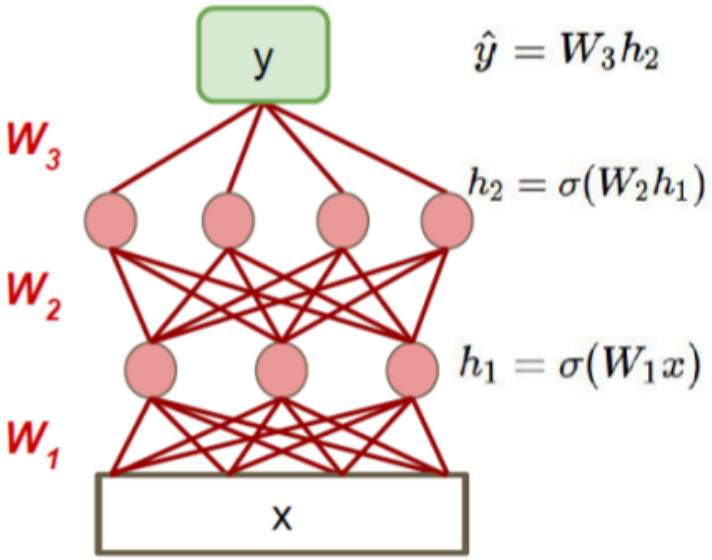


Figure 6.3: General structure of a neural network [58]

Recurrent Neural Network

A standard neural network does not have any persistence. They will classify their input but when they get a stream of inputs (ex. speech, video, ...), they will classify each word independently of each other and without any regards of the previous inputs. A recurrent neural network (RNN) addresses this problem by introducing networks with loops [42]. This way, the output of a previous input has effect on the next input. In figure 6.4, we transform those loops into multiple copies of the same

network which makes it easier to reason about.

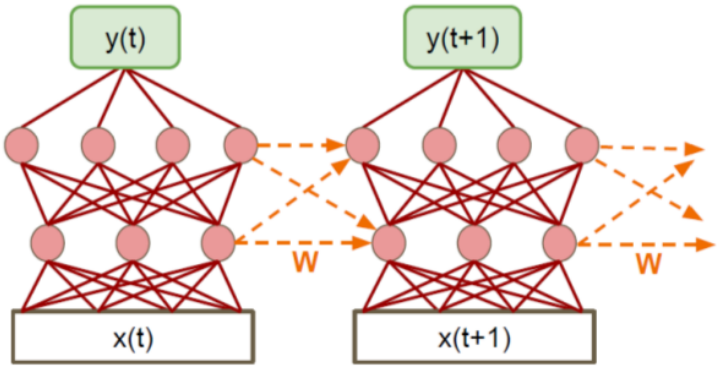


FIGURE 6.4: Unrolled recurrent neural network [58]

The problem with RNNs is mainly that they have trouble learning long-term dependencies which is often essential in time series [13].

Long Short Term Memory

A LSMT network is a specific RNN which is capable of learning long-term dependencies [24]. We will explain the difference with a standard RNN and why a LSTM can learn these long-term dependencies.

A recurrent network is, as we said, a chain of connected neural networks. Those networks can have a simple structure like a single *tanh* layer, see figure 6.5. Representing a complete RNN as one layer which has neurons with activation function *tanh*, makes it easy to reason about.

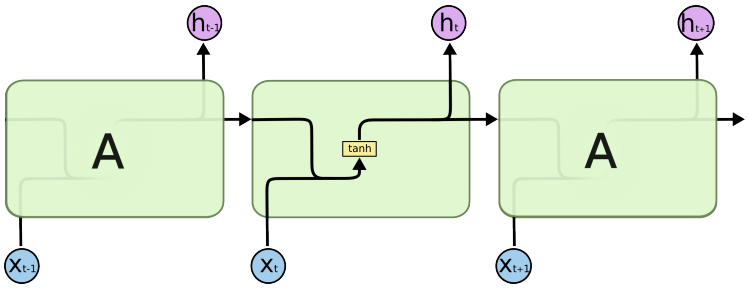


FIGURE 6.5: Unrolled recurrent neural network with a single tanh layer [51]

It is important to see the difference with a LSTM. The repeating network does not have a single neural network layer, but has 4 layers which each fulfills a specific

goal, see figure 6.6.

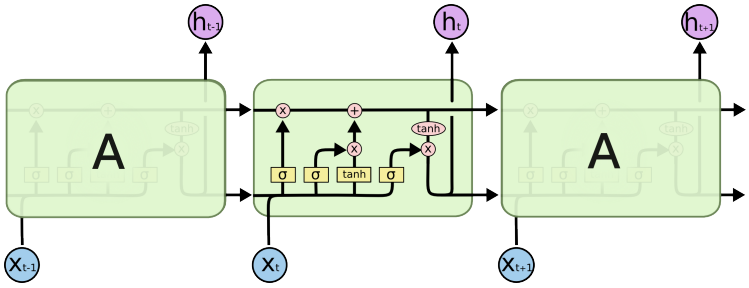


Figure 6.6: Unrolled LSTM network where each network has 4 layers [51]

The main idea behind LSTM is that each repeated network has its own cell state. It functions as a memory which can be updated with each new input. On figure 6.7, you can see the cell state C through time. It can be compared with a conveyor belt which interacts with the input at certain gates. This way the state is updated throughout several inputs and this way a LSTM can keep track of long-term dependencies.

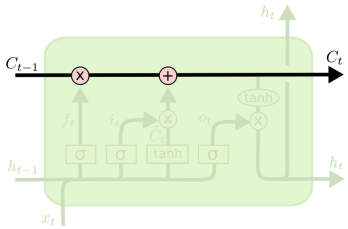
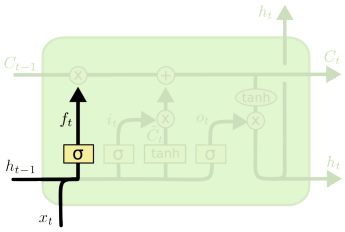


Figure 6.7: Representation of the cell state for a LSTM network [51]

In the following figures, we show the different gates and their functions in changing the cell state depending on the input and the output of the previous network. Next to each figure, the formulas are shown on how the cell state is updated. There should be no surprises as they are not much different than the standard formulas of neural networks.

We start with the forget gate layer of a LSTM. Based on x_t and h_{t-1}, it outputs a number between 0 and 1 for each number in the cell state C_{t-1}. This is shown in figure 6.8. When the output is a 0, the number in the cell state is completely erased. With the output 1, the number does not change.

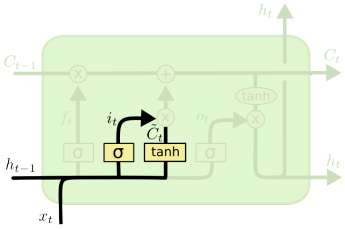
Next we look to the input gate layer. This gate decides which values will be updated in the cell state and outputs those in i_t. It is then combined with the vector $\tilde{C}_t$, which contains the new candidate values based on the input x_t and h_{t-1}. This is



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

FIGURE 6.8: Forget layer of a LSTM network [51]

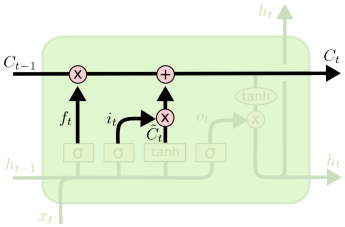
shown in figure 6.9.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

FIGURE 6.9: Input layer of a LSTM network [51]

We can now combine the previous results and adjust the cell state. We multiply the old state with f_t so we forget the needed elements. Then we add $i_t * \tilde{C}_t$ which are the new candidate values multiplied by the amount on how much we want to update each state value. See figure 6.10.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

FIGURE 6.10: Update process of the cell state of a LSTM network [51]

Finally, we need to output h_t to the next network. This is based on the input and the cell state C_t. See figure 6.11.

Variants Long Short Term Memory

In "LSTM: A Search Space Odyssey" [28], research is done between different variants of LSTM networks. It was concluded that the forget gate and the activation function

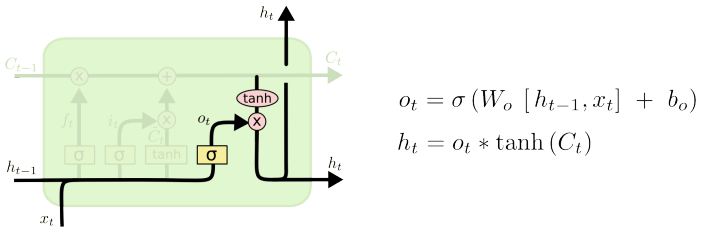


Figure 6.11: Decide the output of a LSTM network [51]

are the most important parts. Other variants do not have a large influence and mainly add extra complexity.

6.4 Conclusion

Before this chapter, we already mentioned some possible roads to explore like a better disease code mapping strategy or extensive parameter tuning.

In this chapter we talked shortly about two possible improvements, a better categorization approach and a better use of data distribution to speed up the process. We mainly focused on LSTM neural networks and explained why they should be suitable to predict or classify disease trajectories. They are able to handle the Curse of Dimensionality, long time dependencies, and take the time aspect of medical data into account. Our Word2Vec approaches can be used in combination with this kind of neural network. The prediction/classification accuracy of the LSMT neural network makes it possible to validate the working of our approaches by testing if the accuracy increases with the use of our Word2Vec approaches.

Bibliography

[1] Google Code Archive - Long-term storage for Google Code Project Hosting. https://code.google.com/archive/p/word2vec/. (Accessed on 05/16/2016).

[2] HealthIT.gov | the official site for Health IT information. https://www.healthit.gov/. (Accessed on 05/15/2016).

[3] MedDRA. http://www.meddra.org/. (Accessed on 05/03/2016).

[4] The Speech Recognition Wiki. http://recognize-speech.com/. (Accessed on 05/16/2016).

[5] THIN research team UCL. https://www.ucl.ac.uk/pcph/research-groups-themes/thin-pub. (Accessed on 05/27/2016).

[6] Using Recurrent Neural Networks in DL4J - Deeplearning4j: Open-source, distributed deep learning for the JVM. http://deeplearning4j.org/usingrnns. (Accessed on 05/21/2016).

[7] WHO | International Classification of Diseases (ICD). http://www.who.int/classifications/icd/en/. (Accessed on 04/27/2016).

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] E. Asgari and M. R. K. Mofrad. Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics. *PLoS ONE*, 10(11):1–15, 11 2015.

[10] G. V. Bard. Spelling-error Tolerant, Order-independent Pass-phrases via the Damerau-levenshtein String-edit Distance Metric. In *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers - Volume 68*, ACSW '07, pages 117–124, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[11] M. Bashiri and A. F. Geranmayeh. Tuning the parameters of an artificial neural network using central composite design and genetic algorithm. *Scientia Iranica*, 18(6):1600 – 1608, 2011.

[12] J. Beel, B. Gipp, S. Langer, and C. Breitinger. Research Paper Recommender Systems: A Literature Survey. *International Journal on Digital Libraries*, pages 1–34, 2015.

[13] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-term Dependencies with Gradient Descent is Difficult. *Trans. Neur. Netw.*, 5(2):157–166, Mar. 1994.

[14] C. Bennett and T. Doub. Data Mining and Electronic Health Records: Selecting Optimal Clinical Treatments in Practice. *CoRR*, abs/1112.1668, 2011.

[15] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *ArXiv e-prints*, June 2012.

[16] A. G. Chris Nicholson. Using Recurrent Neural Networks in DL4J - Deeplearning4j: Open-source, distributed deep learning for the JVM. http://deeplearning4j.org/usingrnns. (Accessed on 05/03/2016).

[17] T. M. Cover. Nearest neighbor pattern classification. 1982.

[18] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[19] G. Declerck, J. Souvignet, J. M. Rodrigues, and M. Jaulent. Automatic annotation of ICD-to-MedDRA mappings with SKOS predicates. In *MIE*, volume 205 of *Studies in Health Technology and Informatics*, pages 1013–1017. IOS Press, 2014.

[20] P. Domingos. A Few Useful Things to Know About Machine Learning. *Commun. ACM*, 55(10):78–87, Oct. 2012.

[21] K. Duan, S. S. Keerthi, W. Chu, S. K. Shevade, and A. N. Poo. Multi-category Classification by Soft-max Combination of Binary Classifiers. In *Proceedings of the 4th International Conference on Multiple Classifier Systems*, MCS'03, pages 125–134, Berlin, Heidelberg, 2003. Springer-Verlag.

[22] W. Duch and N. Jankowski. Survey of Neural Transfer Functions. *Neural Computing Surveys*, 2:163–213, 1999.

[23] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[24] F. Gers. Long Short-Term Memory in Recurrent Neural Networks, 2001.

[25] C. L. Giles, S. Lawrence, and A. C. Tsoi. Noisy Time Series Prediction Using Recurrent Neural Networks and Grammatical Inference. *Mach. Learn.*, 44(1-2):161–183, July 2001.

[26] Y. Goldberg and O. Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014.

[27] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.

[28] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber. LSTM: A Search Space Odyssey. *CoRR*, abs/1503.04069, 2015.

[29] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks. A Closer Look at Skip-gram Modelling.

[30] A. Hameurlain, J. Kung, R. Wagner, H. Decker, L. Lhotska, and S. Link, editors. *Transactions on Large-Scale Data- and Knowledge-Centered Systems IV - Special Issue on Database Systems for Biomedical Applications*, volume 8980 of *Lecture Notes in Computer Science*. Springer, 2011.

[31] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[32] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[33] K. J. M. Janssen, A. R. T. Donders, F. E. J. Harrell, Y. Vergouwe, Q. Chen, D. E. Grobbee, and K. G. M. Moons. Missing covariate data in medical research: To impute is better than to ignore. *Journal of Clinical Epidemiology*, 63(7):721–727, 2016.

[34] A. B. Jensen, P. L. Moseley, T. I. Oprea, S. G. Ellesoe, R. Eriksson, H. Schmock, P. B. Jensen, L. J. Jensen, and S. Brunak. Temporal disease trajectories condensed from population-wide registry data covering 6.2 million patients. *Nat Commun*, 5, Jun 2014. Article.

[35] C. K. R. Jimeng Sun. Big Data Analytics for Healthcare. 2013.

[36] A. Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. (Accessed on 05/03/2016).

[37] E. Keogh and A. Mueen. *Encyclopedia of Machine Learning*, chapter Curse of Dimensionality, pages 257–258. Springer US, Boston, MA, 2010.

[38] S. Kimura, T. Sato, S. Ikeda, M. Noda, and T. Nakayama. Development of a Database of Health Insurance Claims: Standardization of Disease Classifications and Anonymous Record Linkage. *J Epidemiol*, 20(5):413–419, Sep 2010. 20699602[pmid].

[39] W. M. Koolen, T. van Erven, and P. Grünwald. Learning the Learning Rate for Prediction with Expert Advice. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2294–2302. Curran Associates, Inc., 2014.

[40] O. Levy and Y. Goldberg. Dependency-Based Word Embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 302–308, 2014.

[41] O. Levy and Y. Goldberg. Neural Word Embedding as Implicit Matrix Factorization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2177–2185. Curran Associates, Inc., 2014.

[42] Z. C. Lipton. A Critical Review of Recurrent Neural Networks for Sequence Learning. *CoRR*, abs/1506.00019, 2015.

[43] Z. C. Lipton, D. C. Kale, and R. C. Wetzel. Phenotyping of Clinical Time Series with LSTM Recurrent Neural Networks. *CoRR*, abs/1510.07641, 2015.

[44] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187 – 197, 2015.

[45] C. Miani et al. Health and Healthcare: Assessing the Real World Data Policy Landscape in Europe. 2014.

[46] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, abs/1310.4546, 2013.

[47] A. Moores. Efficient Memory-based Learning for Robot Control. 1991.

[48] M. Nielsen. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com/. (Accessed on 05/03/2016).

[49] Observational Medical Outcomes Partnership. OSIM2 - Observational Medical Dataset Simulator Generation 2 | Observational Medical Outcomes Partnership. http://omop.org/OSIM2. (Accessed on 05/28/2016).

[50] C. Olah. Deep Learning, NLP, and Representations - colah's blog. http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/. (Accessed on 05/16/2016).

[51] C. Olah. Understanding LSTM Networks – colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed on 05/03/2016).

[52] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training Recurrent Neural Networks. *CoRR*, abs/1211.5063, 2012.

[53] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online Learning of Social Representations. *CoRR*, abs/1403.6652, 2014.

[54] X. Rong. word2vec Parameter Learning Explained. *CoRR*, abs/1411.2738, 2014.

[55] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386, 1958.

[56] H. Sak, A. W. Senior, and F. Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *CoRR*, abs/1402.1128, 2014.

[57] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[58] J. Simm. IMEC Technical talk.

[59] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(1):579–589, 1991.

[60] C. P. Stone. A Glimpse at EHR Implementation Around the World: The Lessons the US Can Learn. 2014.

[61] J. Sun, F. Wang, J. Hu, and S. Edabollahi. Supervised Patient Similarity Measure of Heterogeneous Patient Records. *SIGKDD Explor. Newsl.*, 14(1):16–24, Dec. 2012.

[62] D. D. Team. Deeplearning4j: Open-source distributed deep learning for the JVM.

[63] G. B. Team. Vector Representations of Words. https://www.tensorflow.org/versions/0.6.0/tutorials/word2vec/index.html. (Accessed on 05/16/2016).

[64] F. Wang, N. Lee, J. Hu, J. Sun, and S. Ebadollahi. Towards Heterogeneous Temporal Clinical Event Pattern Discovery: A Convolutional Approach. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 453–461, New York, NY, USA, 2012. ACM.

Master thesis filing card

Student: Milan van der Meer

Title: Learning a Diseases Embedding using Generalized Word2Vec Approaches.

UDC: 621.3

Abstract:

Here comes a very short abstract, containing no more than 500 words. LaTeX commands can be used here. Blank lines (or the command `\par`) are not allowed!

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Thesis submitted for the degree of Master of Science in Engineering: Computer Science, specialisation Artificial Intelligence

Thesis supervisors: Prof. dr. R. Wuyts
Alexander Vapirev

Assessors: Prof. dr. ir. H. Blockeel
R. van Lon

Mentor: Dr. E. D'Hondt