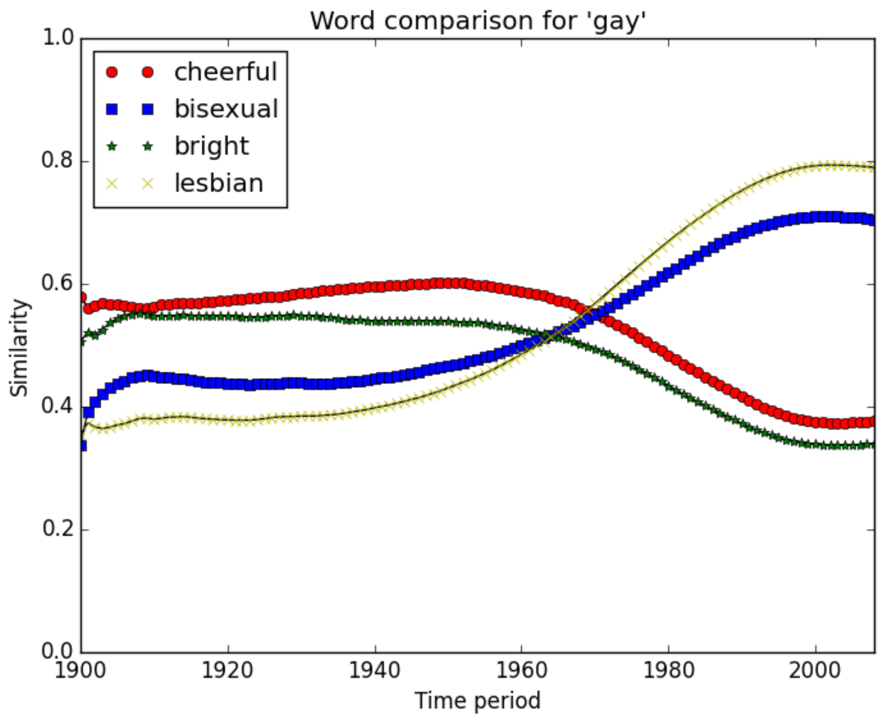
CHALMERS
UNIVERSITY OF TECHNOLOGY



Tracking temporal evolution in word meaning

with distributed word representations

Master's thesis in Computer Science - Algorithms, Languages and Logic

HENRIK ALBURG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Tracking temporal evolution in word meaning

with distributed word representations

HENRIK ALBURG

Department of Computer Science and Engineering
CSE LAB Research Group
Chalmers University of Technology
Gothenburg, Sweden 2015

Tracking temporal evolution in word meaning
with distributed word representations
HENRIK ALBURG

Supervisor: Fredrik Johansson, Department of Computer Science and Engineering
Examiner: Aarne Rante, Department of Computer Science and Engineering

Department of Computer Science and Engineering
CSE LAB Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Word visualization created with matplotlib

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2015

Tracking temporal evolution in word meaning
with distributed word representations
HENRIK ALBURG
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Some words change meaning over time and are thus used differently in text. The purpose of this thesis is to create a model able to find these changes in word meaning, by studying lots of data from different time periods. Building on recent advancements in machine learning and semantic modelling the model is successfully able to find and make sense of changes in word meaning over time. The model can automatically find the most changed words during a time span and these words tend to agree with our perception of the words that have changed the most. When measuring changes the model achieves a 0.6 correlation when compared to human raters.

Keywords: machine learning, word embeddings, skip-gram, group fused lasso, word2vec, time varying, nlp.

Acknowledgements

I would like to thank my supervisor at Chalmers, Fredrik Johansson, for his help and guidance during this thesis. I would also like to thank my advisor at Findwise, Jonatan Bengtsson, for all his help and input. Lastly I would like to thank Findwise Gothenburg, for letting me do my thesis at their office.

Henrik Alburg, Gothenburg, November 2015

Contents

List of Figures

1 Introduction

Language is everchanging. How we use words and what they mean to us sometimes change over time. New words are born, which changes not only how we use that word, but also other words with it. Older words might be replaced and almost stop being used. Some words go through gradual changes over longer periods of time, while some might change overnight. “Awful” was once a synonym to awesome and if something was “silly”, it was blessed, or worthy. As we know, this is not the case today, but when did these changes take place? How did the word meaning of “nuclear” get affected by the disaster in Chernobyl or the explosion of atomic bombs? Has the word “transmit” changed meaning as we have developed more advanced methods of communication? These are interesting questions for linguistic researchers to answer.

We all have an intuition about how words have changed by how we use them, but this intuition might be biased and it is probably not based on a very big, diverse, sample. However, with the vast amount of text digitalized, computers should be able to study how the usage of words change over time, in a more objective manner. It would probably be very interesting for linguistic researchers, to be able to make use of all the data and help answer some of their questions. It would also be interesting for people working with natural language processing (NLP), which has been described as one of the most important technologies of the information age. Also in order for computers to be able to understand and interpret text in a correct way, these changes must be taken into consideration.

Machine learning methods are becoming vital tools when it comes to making use and taking advantage of the vast amount of data available as written language. Instead of telling the computer how it should handle every corner case, as in traditional algorithms, or try to model the relationships between words, a programmer tells the computer how it should learn a representation from some data. The computer is told how to model words and what patterns in the data to look for in order to create represenations for every word. For complex structure such as text, where there is a lot of information, it is very hard, or close to impossible, to model it all by hand. Machine learning on the other hand shines when there are complex structure and a lot of data to parse.

One tool within machine learning suited for modelling written language is distributed word representations. These representations embed the words of a language in a geometric space allowing for natural distance computation and clustering, giving a computer the ability to compare words to each other. This subject has received great attention in the field of machine learning in recent years [1, 2].

By extending distributed word representation to model words for different time periods it should be possible to observe changes in words' meaning by calculating the difference in their representations. If done correctly, it should then be easy to find the words with the biggest changes and be able to observe when those changes took place. This is, however, not an easy task to do correctly. There are many things to take into consideration and not always straight forward to realize what works well in practice. How one such model, able to track the temporal evolution in word meaning, can be created, is what is done in this thesis.

1.1 Goal

The goal for this thesis is to create a dynamic model able to track the temporal evolution in word meaning for a set of words. Specifically the model should:

- be able to measure the amount of change in a given word
- be able to automatically find the words with the most and the least change in word meaning
- be able to model changes in word meaning as a smooth function over time
- be able to limit the amount of changes in the representation

One key result to evaluate is if the model is able to find all words with next to no change over a period of time and if this agrees with our intuition about the semantic changes for the words.

1.2 Challenges

In order to create a model that is able to track the temporal evolution of language, many obstacles must be overcome. To be able to automatically find the words, within a set, that has changed the most over some time period one must be able to measure the change of a word. With distributed word representations it is possible to measure the difference between two words, however now we need to do it through time. Therefore, the vectors must be modeled over time and the time periods must be dense enough so that the changes can be tracked and it is clear when the change took place.

Our intuition about language is that most words do not change their semantic meaning, even over long periods of time and this is something that the model should capture. However, a change in a word might look as changes in the word's new surrounding. This creates noise and uncertainty about what words are actually changing their meaning, which is far from ideal and should therefore be taken into consideration.

To model word changes as smooth functions over time is hard when the data comes from discrete events. For dense enough data, this might not matter, but when the data comes from discrete time periods with long periods of time between them it might be harder to realize the underlying function.

1.3 Scope

Words and language are very complex in general and there is currently nothing near a perfect model. Therefore, simplifications has to be made. Firstly, the model will not try to disambiguate between different word meanings for a word. That is, every word will be modeled as only having one sense, whereas we could model words with multiple senses as done by Huang et al. [3].

Also, although the model is said to model smooth changes in word meaning over time, there will be discrete time periods specified in regular intervals. It can be thought of as snapshots from different time points in a model with smooth changes. This could perhaps be extended to a model where time periods are not given at all, as future work.

Lastly, changes of word frequencies will not be taken into consideration. It could further be investigated how a change in frequency of a given word might relate to the change of its semantic meaning, or even how it might relate to the finding of a change in semantic meaning, but this will not be covered in this thesis.

1.4 Thesis Outline

Section 2 introduces the theoretical framework on which this thesis is built and is useful to understand the rest of the thesis. However, if the reader is familiar with the Skip-gram model and regularization, it can be skipped completely. Section 3 introduces the new model created in this thesis by describing the motivation, creation and implementation of the model, building on much of the theory from section 2. Section 4 briefly introduces the relevant research in the area, and a baseline model which will be used for comparisons in the experiments. Section 5 presents the data set used for evaluation and the different experiments used to measure how well the model represents changes in the semantic meaning of words, compared to a baseline model described in section 4. Section 6 evaluates and discusses the results and suggests possible extensions, or changes, to the model. Finally, the conclusion in section 7 summarizes the thesis, gives some ideas for future work and some final thoughts.

2
Theory

This chapter explains the underlying theory that is relevant to understand the rest of this thesis. Firstly, it presents distributed word representations and some key results using it. Then the Skip-gram model is introduced, which is commonly used to learn the vector representations. Finally, an introduction to regularization is given, which is needed to understand the next chapter.

2.1 Distributed word representation

Distributed word representations model words' semantic meaning as dense continuous vectors. Here, a representation means that something is represented as a mathematical object [1]. In our case a word w is represented as a vector $\mathbf{u}_w$ from the vector space $\mathbb{R}^d$. Being distributed implies that one dimension of the vector partly models many aspects of a word [4]. This means that one dimension is able to model several aspects of a word and many dimensions are used together to model a certain aspect of a word.

Every word in a fixed vocabulary W is mapped to a vector, or a point in multidimensional space, and the more similar two words' semantic meaning are, the closer their vectors should be in vector space. We can therefore measure the semantic similarity of two words by calculating the vector distance, or the angle, between the two words' vectors. It is also possible to do clustering of words, by doing clustering on their vectors.

The semantic meaning of a word is based on its context. This idea was introduced in the distributional hypothesis by Harris in 1954 [5], which states that words that occur in the same contexts tend to have similar meaning. Thus, two words that are interchangeable in a sentence are in some way semantically similar and words that often occur with the same neighbour words have similar meaning.

Good word vectors have shown to exhibit linear structure where simple vector arithmetic yields comprehensible results. For instance, results by Mikolov et al. [2] have shown that the difference between vec('man'), where vec('man') is the vector in our model for the word man, and vec('king') is the same as the difference between vec('woman') and vec('queen'), which indicates that the vectors discover the underlying structure of words. This makes it possible for the model to answer analogy questions like "king is to man as ... is to woman" with the correct answer: queen.

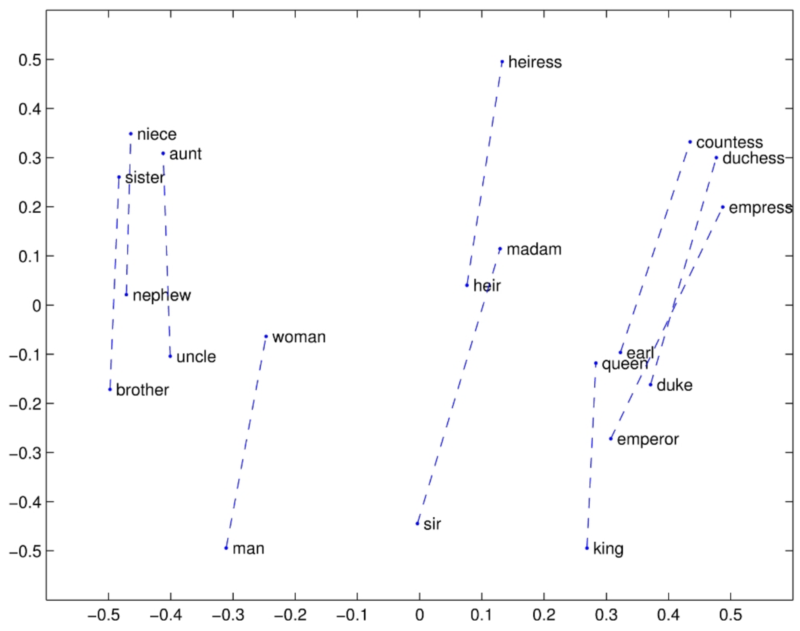


Figure 2.1: Words with the same relationship, such as brother to sister and king to queen, seems to have approximately the same difference between their word vectors. High dimensional vectors plotted in 2D with PCA. Source: http://nlp.stanford.edu/projects/glove/

As can be seen in figure 2.1, the linear structure can be shown among many word-pairs with the same relationship. Also other relationships such as that between a word in singular and the same word in plural, or a country and its capital, often have the same vector difference between them [6]. This is a good indicator that the distributed word representations models the semantic meanings well and it is probably the main reason that the resulting vectors have been able to give great results in various NLP applications [7, 8, 9, 10, 11, 12, 13].

2.2 Skip-gram model

The Skip-gram model, popularized through the software Word2Vec, was one of the model architectures introduced by Mikolov et al. [2] in 2013 and is a model able to calculate distributed word representations with state of the art performance at a low computational cost. Therefore it has been widely used, both within research and in practice. It should be noted that what will be referred to, in this thesis, as the Skip-gram model actually is what is called the *Skip-gram model with negative sampling* as introduced by Mikolov et al. in [6]. This is as to not confuse the reader with the different architectures of the Skip-gram model not relevant to this thesis. The explanation in this section is similar to that of Levy et al. in [14].

The Skip-gram model is unsupervised in the sense that it takes a corpus as input and outputs word vectors for all words in a vocabulary W. It makes use of the

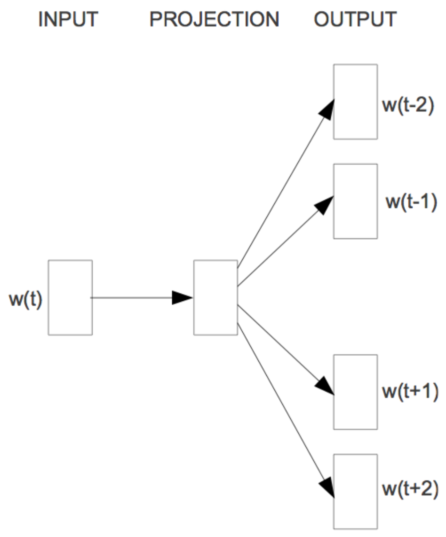


Figure 2.2: The Skip-gram model uses the word vector of a word to predict its surroundings.

distributional hypothesis, explained in section 2.1, by trying to, given a word, predict the context-words within a window, as shown in figure 2.2. Thus it creates word-context pairs for every context-word within this window, which are then used to train the model. The window is slided through the text, creating pairs from the entire input which results in our corpus D.

For every word w in a word-context pair (w, c) from the corpus D the model also randomizes a fixed number of contexts, decided as a parameter, to create new word-context pairs (w, c') that have no relation to the text. These pairs are what is called *negative samples*. The idea is that the model should be able to differentiate the pair that came from the input text from the other, random, pairs. This means that the model tries to classify if a pair has come from the corpus or not. By changing the word vectors we are trying to find the vectors that maximizes the likelihood for our classifications, that is, we want to find the set of vectors that are best at distinguishing a pair from D from the random ones.

Every word w in the vocabulary W is given two vectors, one as a word-vector, $\mathbf{u}_w$, and one as a context-vector, $\mathbf{v}_w$. These vectors make up the parameters of the model, represented as matrices U and V, where a vector is a row in the matrix. The model wants to find the parameters, the set of vectors, that maximizes the likelihood of all classifications. Namely that all the pairs, (w, c), from the corpus, D, are predicted as coming from the corpus and that all the negative samples, (w, c'), are predicted as not coming from the corpus. For a given word-context pair this means that the model tries to predict if the pair actually came from the corpus or not and this is done by using the logistic function between the word and context vector.

$$p(D = 1|w, c) = g(\mathbf{u}_w, \mathbf{v}_c) = \frac{1}{(1 + e^{-\mathbf{u}_w^\top \mathbf{v}_c})} \tag{2.1}$$

here $p(D = 1|w, c)$ is the probability that the pair (w, c) came from the corpus D and this is represented as the dot product between the word vector for word w and the context vector for word c. This is then used in a logistic function where a high dot product will result in a probability closer to 1.

The Skip-gram model captures the intuition that words used in similar contexts have similar meaning. If two words w_1 and w_2 are semantically similar then our model says that their vectors will be similar. This also leads to that the probability of using one of the words with a certain context word c should be almost the same as using the other, since the logistic function depends on the vectors themselves. Thus we will have similar predictions for the pair (w_1, c) and (w_2, c) because the words are semantically similar and since they will often occur with the same context-words, according to the distributional hypothesis, this will lead to good predictions [15].

To describe our intuition mathematically, a likelihood function can be used. By saying that the word pairs are independent the likelihood to observe all word pairs can be describes as a product of observing the individual word pairs, which is described in equation (2.1). The likelihood for all word pairs, both from the corpus D and the negative samples, thus becomes

$$L(U, V) = \prod_{(w,c)\in D} p(D = 1|w, c) \prod_{(w,c)\notin D} p(D = 0|w, c) \tag{2.2}$$

In order to obtain the best possible word vectors, the model should maximize this likelihood. By trying to predict that all pairs (w, c) from the corpus gets a probability close to 1 in the logistic function and that the negative samples gets a probability close to 0 the model is able to differentiate between pair of words from the input text and random ones. Thus, our objective is to find the parameters, or vectors, that maximizes the given likelihood.

In order to maximize the likelihood in equation (2.2) it has to be rewritten to a more common form. By using the log likelihood of a function it is possible to rewrite the products as sums of logarithmic expressions, while keeping the same maximum for the function. It is also possible to rewrite a maximization as a minimization of its negation giving us a standard expression that is easier to solve. From the likelihood we can thus derive our objective function

$$\underset{U,V}{\text{minimize}} \quad -\ell(U, V) = \sum_{(w,c)\in D} \log g(\mathbf{u}_w{}^\top \mathbf{v}_c) + \sum_{(w,c)\notin D} \log g(-\mathbf{u}_w{}^\top \mathbf{v}_c) \tag{2.3}$$

By optimizing this objective function we are able to find the best set of parameters, the best vectors, that can differentiate between word-pairs from the corpus and random word-pairs. Rewriting a likelihood like this is often done to probabilistic models in order to apply gradient methods to solve them. In our case the method is Stochastic Gradient Descent, explained in the next section.

The Skip-gram model also has some engineered improvements such as sub-sampling of frequent words. Since almost every word will co-occur with the word "the" it does not matter as much that word-pairs where one of the words is "the" is trained. Therefore, some pairs will be discarded and frequent words have a higher probability of being discarded.

2.3 Stochastic gradient descent

Stochastic gradient descent (SGD) is a commonly used and well tested approximate method for finding the set of parameters that minimizes some objective function. It is based on gradient descent but more suited for problems on a larger scale [16], as it avoids computing the full gradient at every iteration.

Gradient descent tries to find the minimum of a function by staring in some point, calculate the gradient for the function in that point and then move to a new point in the gradient's direction. Then calculate the gradient from this new point and move along the gradient's direction again and do this until convergence, which means that a minimum has been found. Lets say that we would like to minimize an objective function f on the following form

$$\underset{x}{\text{minimize}} \quad f(x) = \sum_{i=1}^{n} f_i(x)$$

Here we want to find the parameter x^* that minimizes the function $f(x)$ and every $f_i(x)$ is associated with some example or observation. Functions like these often appear in probabilistic models, because the log-likelihood of independent observations becomes a sum over the observations. Gradient descent would need to calculate the gradient for every observation, $f_i(x)$, before doing an update on the parameter x to get it closer to x^*. However, when n is large, this might be very expensive. If we have 3 million observations then in order to get the full gradient we would need to calculate the gradient for the 3 million terms and sum them together. Then the full gradient can be used to do a small update in the parameter, which does not seem efficient.

What SGD does is that it calculates the gradient for a single example instead of the entire expression. As in the above case where we have multiple examples, SGD can calculate the gradient for one $f_i(x)$ and do an update according to its gradient. It might not be the optimal direction to move in but the expected direction after a lot of examples will be approximately optimal. This is not as reliable as gradient descent in general, but with enough training examples it has been shown to give good results and a fast convergence speed [16]. Instead of doing 3 million calculations to get the full gradients for one update step it is possible to do 3 million updates with one calculation of the gradient of an example each.

Often when there are independent observations where the likelihood becomes a product of the probabilities for the independent observations, as in equation (2.2), it is possible to rewrite the likelihood function as a sum of the logarithmic probabilities, without changing its maximum. By doing this, and try to minimize the likelihood of its negative, we get an equation such as the one in equation (2.3). SGD is applicable to equations on that form, making it easy to find the best set of parameters for the problem.

2.4 Regularization

Regularization is widely used within machine learning as a way of adding additional information to a model. This could be in order to promote a simpler model and prevent overfitting or to include prior information, such as some intuition about the true model. By adding a new term besides the likelihood, the objective function is slightly changed to accommodate both the likelihood and this new term. Thus, we have to find a balance between optimizing for the likelihood and the regularization term in order to find the global optimum. It will therefore create solutions which focuses on both the likelihood and the additional information. This is illustrated by an equation on the following form

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad -\ell(x|\boldsymbol{\beta}) + R(\boldsymbol{\beta})$$

where $\ell(x|\boldsymbol{\beta})$ is the likelihood function for the data x given the parameters $\boldsymbol{\beta}$ and $R(\boldsymbol{\beta})$ is the regularization term. As before the objective is to minimize $-\ell(x|\boldsymbol{\beta})$ but certain aspects of $\boldsymbol{\beta}$ will make the objective bigger, decided by $R(\boldsymbol{\beta})$. It can put a penalty on the number of features used or on high parameter values, in order to promote a simpler model.

There are different types of regularization terms, different ways of penalizing the parameters, used differently depending on the wished outcome. To get sparse solutions, where few parameters actually are used, the most common type of regularization is the *Lasso* [17]. This can be seen as variable selection, as many parameters will be set to 0, and therefore not be used, which creates a simpler model and avoids overfitting. The Lasso penalizes the sum of absolute values of the parameters. Note that lambda is a non-negative tuning parameter able to decide the strength of the penalty.

$$R(\boldsymbol{\beta}) = \lambda||\boldsymbol{\beta}||_1 \tag{2.4}$$

It is well used and has been shown to give sparse solutions where only the most relevant parameters will be used. This can make the model more intuitive because those parameters used can be seen as those who actually matter for the result. However, it is not enough in more advanced cases when there is underlying structure to the parameters.

One more advanced method is called the *Fused Lasso* [18] and it is able to make use of more advanced structure by penalizing the differences between neighbouring parameters. This can be used when the parameters are in some way ordered, such as in time series. It takes the form

$$R(\boldsymbol{\beta}) = \lambda \sum_{i=2}^{N} \left|\boldsymbol{\beta}_i - \boldsymbol{\beta}_{i-1}\right| \tag{2.5}$$

where two consecutive parameters should have a similar value in order to not get penalized, but this structure is not always complex enough to model the underlying structure, mainly because a parameter might be a part of some group and not the entirely independent. Another common structure is when the parameters are being grouped together in which case the *Group Lasso* [19] is commonly used. It is often useful when a group of parameters should be included together or none of them at all. It takes the form

$$R(\boldsymbol{\beta}) = \lambda \sum_{g=1}^{G} \|\boldsymbol{\beta}_g\| \tag{2.6}$$

where $\boldsymbol{\beta}_g$ is the set of parameters belonging to the g:th group. The intuition here is that parameters belong together, maybe the model uses arrays of parameters or there are actual underlying groups and the objective is to find the groups that affect the result. The Group Lasso can be combined with the Fused Lasso for a more advanced structure, simply called *Group Fused Lasso* [20].

$$R(\boldsymbol{\beta}) = \lambda \sum_{i=2}^{N} \|\boldsymbol{\beta}_{i,\bullet} - \boldsymbol{\beta}_{i-1,\bullet}\|_2 \tag{2.7}$$

Though the group fused lasso can limit the change in structures such as groups of variables over time, it creates a more complex optimization problem. Depending on the rest of the function, and the size of the problem, it might not be possible to efficiently calculate a gradient for the expression making its use case situational.

2.4.1 Proximal Methods

Proximal methods are techniques used in optimization when some part of the problem might be non-smooth, large scale or non-differentiable. The algorithms are very general, and can solve a lot of different optimization problems, and has been shown to be well suited for problems of high dimension [21]. By making use of the *proximal operator*, the problem can often be turned into a projection with a closed form solution, making it very efficient to solve.

The proximal operator of a convex function f is defined as

$$\boldsymbol{prox}_f(x) = \underset{u}{\operatorname{argmin}}(f(u) + (1/2)\|u - x\|_2^2), \tag{2.8}$$

Proximal gradient methods are applicable to a lot of regularization problems and problems on similar forms where there are two terms

$$\min_x \; F(x) + R(x), \tag{2.9}$$

where F is convex and differentiable but R might not be differentiable. By realizing that some optimal solution x^* minimizes equation (2.9) if and only if it is a solution to

$$x^* = prox_{\lambda R}(x^* - \lambda \nabla F(x^*)), \tag{2.10}$$

Thus it is possible to solve this equation instead, which is differentiable because of the strict convexity of the l_2 norm in the proximal operator.

A good example of a problem able to be solved in this way this is the Lasso regularization problem. l is convex and differentiable but R is only convex. Thus we can compute the proximity operator for the regularization term (or find someone who has) to simply use the solution

$$(prox_{\lambda R})_i = \begin{cases} x_i - \lambda, & x_i > \lambda \\ 0, & |x_i| \leq \lambda \\ x_i + \lambda, & x_i < -\lambda \end{cases} \tag{2.11}$$

Thus the entire problem can be solved by doing the following update

$$\beta^{k+1} = prox_{\lambda R}(\beta^k - \lambda \nabla F(\beta^k)), \tag{2.12}$$

in the same fashion as is done by SGD, explained in section 2.3.

The group lasso can be solved similarly where the proximity operator becomes

$$prox_{\lambda R}(\beta_g) = \begin{cases} \beta_g - \lambda \frac{\beta_g}{||\beta_g||_2}, & ||\beta_g||_2 \geq \lambda \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

making it very efficient to solve.

3
Our Approach

This section describes the creation of a new model, able to fulfill the goals of the thesis. First it motivates certain aspects of the model and suggest how these can be accomplished. Then the model is described in more detail before the algorithm and implementation is presented.

3.1 Time varying word embeddings

Distributed word representations, often referred to as word embeddings, have shown to be general and powerful representations of words, where every word w is embedded as a vector $\mathbf{u}_w$. This can simply be extended to the time varying setting by embedding every word w as a vector for every time period t, creating vectors $\mathbf{u}_{w,t}$ for every word w in every time period t. By doing this it is possible to model not only word meaning, but word meaning at different times. In section 2.2 we had a corpus D, consisting of all our word-pairs (w, c). Now we will have word pairs for every discrete time period, partitioning our corpus into parts for every time period $D = \{D_t\}_{t=1}^{T}$.

For the model to successfully capture changes in word meaning over time it is important that a change in a word's embedding is equivalent to a change in the semantic meaning of that word. There should only be a change in the word embedding from one time period to the next if there is a change in word meaning during that time. Thus the change in word meaning for the word w between time periods t and t' can be measured by measuring the similarity of the vectors $\mathbf{u}_{w,t}$ and $\mathbf{u}_{w,t'}$. It is important to be able to measure the change of a word vector in this way in order to measure the word's semantic change over time.

As is known, most words do not significantly change word meaning over time, even across long time periods. This prior information can be taken into account, to create a better model. By limiting the changes in the word embeddings for consecutive time periods, that is, to limit $\|\mathbf{u}_{\bullet,t} - \mathbf{u}_{\bullet,t+1}\|$, it is possible to take advantage of this knowledge. One possible way of doing this is to introduce regularization into the model, penalizing the mentioned change.

3.2 Extending the Skip-gram model

The Skip-gram model has shown great results in creating good word embeddings, efficiently. Therefore, it makes sense to extend that model with a dimension for time. The new model needs to take time into consideration, and since we believe that all our word-pairs are independent we can extend the likelihood as a sum over time periods. It also needs to make a distinction in the data where data from time period t is labeled as D_t, instead of a unified corpus D, as explained in the previous section. Also the new notation for word embeddings in a given time period, where the vector for the word w in time period t is embedded as $\mathbf{u}_{w,t}$ should be used. Thus our new log-likelihood becomes

$$\ell(U,V) = \sum_{t=1}^{T} \left[\sum_{(w,c)\in D_t} \log g(\mathbf{u}_{w,t}{}^\top \mathbf{v}_{c,t}) + \sum_{(w,c)\notin D_t} \log g(-\mathbf{u}_{w,t}{}^\top \mathbf{v}_{c,t}) \right] \tag{3.1}$$

This new likelihood is very similar to that of the Skip-gram model, only with some change in notation and a sum over all time periods. The function $g(z)$ is the logistic function, as in equation (2.1).

To incorporate the knowledge that word tend to not change from one time period to the next, regularization is added to the model. This is done by adding the term

$$R(U,V) = \lambda \sum_{t=2}^{T} \sum_{w\in V} \left[\|\mathbf{u}_{w,t} - \mathbf{u}_{w,t-1}\|_2 + \|\mathbf{v}_{w,t} - \mathbf{v}_{w,t-1}\|_2 \right]. \tag{3.2}$$

The Group Fused Lasso is chosen as the penalty because the model should penalize a change in an entire word vector, that is, in all dimensions together, for consecutive time periods. Using the Lasso penalty for regularization would lead to word vectors with few non-zero dimensions, though the idea of word embeddings is that the vectors should be dense and not sparse. Also fewer dimension could have been chosen from the beginning. To introduce the Fused Lasso penalty would lead to some dimensions not changing from one time period to the next, or even consecutive dimensions being similar, and not the whole word, which is closer but still not ideal. The Group Fused Lasso penalty would force sparsity in entire word changes [20] which is desired, however, it is more computationally expensive to solve.

The parameter λ can be tuned in order to enforce more sparsity, less changes in word meaning. If λ is 0 then the model will be the same as the one without regularization and a high enough λ would result in a model without changes. The λ can be tuned to values in between in order to obtain the wanted number of changes. If one knows that there is little change during the time span than a larger λ can be used and if it is known that there will be lots of changes then a smaller λ can be used.

The objective function can now be described as

$$\underset{U,V}{\text{minimize}} \quad -\ell(U,V) + R(U,V) \tag{3.3}$$

where $\ell(U,V)$ is equation (3.1) and $R(U,V)$ is equation (3.2).

3.3 Algorithm

The algorithm created for training the time varying word embeddings is based on stochastic proximal gradient descent (SGD). It also incorporates the extensions discussed in the previous sections, meaning that the objective that is minimized is the equation in (3.3).

First, mainly because of the regularization, the objective function will be rewritten in terms of changes between time periods instead of vectors for every time period. Let $\Delta\mathbf{u}_{w,t} = \mathbf{u}_{w,t+1} - \mathbf{u}_{w,t}$, denote the change in the embedding of word w at time t. The vector $\mathbf{u}_{w,t}$ can now be represented as $\hat{\mathbf{u}}_w + \sum_{j=1}^{t-1} \Delta\mathbf{u}_{w,j}$, the sum of the start vector, $\hat{\mathbf{u}}_w$, and the changes in all time periods before t. This will not change the outcome of the objective function, however, it will make the regularization easier to handle.

The model will make use of pre-trained word embeddings for the first time period and this set of vectors will not be updated during training of the time varying word embeddings. The model will be very sensitive to big changes in the first time period and by approximating the word embeddings for that time period we get a better start than letting the model figure out the word embeddings during run time. This should make it faster and easier to achieve good word embeddings.

By defining $f(w,c,t) = \log g(\mathbf{u}_{w,t}{}^{\top}\mathbf{v}_{c,t})$ for any $w, candt > 1$ we get

$$f(w,c,t) = \log g\left((\hat{\mathbf{u}}_w + \sum_{j=1}^{t-1} \Delta\mathbf{u}_{w,j})^{\top}(\hat{\mathbf{v}}_c + \sum_{j=1}^{t-1} \Delta\mathbf{v}_{c,j})\right) \tag{3.4}$$

and for the entire likelihood function this simply becomes

$$\ell(U,V) = \sum_{t=1}^{T} \left[\sum_{(w,c)\in D_t} f(w,c,t) + \sum_{(w,c)\notin D_t} (1 - f(w,c,t)) \right] \tag{3.5}$$

Further, the regularization term R becomes

$$R(U,V) = \lambda \sum_{t=1}^{T-1} \sum_{w\in V} \left[\|\Delta\mathbf{u}_{w,t}\|_2 + \|\Delta\mathbf{v}_{w,t}\|_2 \right] . \tag{3.6}$$

which is much simpler to handle than the previous expression. By rewriting the model as changes between time periods it is possible to use the Group Lasso [22], which will limit the amount of changes between time periods in a correct way. An optimization problem with a Group Lasso penalty can also be solved using proximal gradient descent with a standard group lasso proximal operator. This means that it is possible to use a closed form solution, which is very efficient to use. As explained in section 2.4.1, the regularization becomes a projection on the form [21]

$$\mathcal{P}_{\lambda k\alpha}(\Delta\mathbf{u}_w^t) = \begin{cases} \Delta\mathbf{u}_w^t - \lambda k\alpha \frac{\Delta\mathbf{u}_w^t}{\|\Delta\mathbf{u}_w^t\|_2}, & \|\Delta\mathbf{u}_w^t\|_2 \geq \lambda k\alpha \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

that needs to be applied right before a word embedding is updated. It shrinks the changes and if the change is small enough it sets it to zero, which has been shown to produces sparse results.

The plan is to solve the optimization problem with the use of SGD which means that the gradients with respect to the change vectors for an example is needed. An example in this case is a triplet of (w, c, t), namely a word-pair from a given time period. The likelihood is decomposed to a sum of these examples

$$\ell(U, V) = \sum_{w,c,t} \ell_{w,c,t} \tag{3.8}$$

where for every example

$$\ell_{w,c,t} = \begin{cases} f(w, c, t), & (w, c) \in D^t \\ \log(1 - e^{f(w,c,t)}), & \text{otherwise} \end{cases} \tag{3.9}$$

It is now possible to write the gradients of every example with respect to the change vectors

$$\nabla_{\Delta \mathbf{u}_{w,t}} \ell_{w,c,t'} = \begin{cases} g\left(-\mathbf{u}_{w,t'}{}^{\top}\mathbf{v}_{c,t'}\right)\mathbf{v}_{c,t'}, & t < t', (w, c) \in D^{t'} \\ -g\left(\mathbf{u}_{w,t'}{}^{\top}\mathbf{v}_{c,t'}\right)\mathbf{v}_{c,t'}, & t < t', (w, c) \notin D^{t'} \\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{3.10}$$

$$\nabla_{\Delta \mathbf{v}_{c,t}} \ell_{w,c,t'} = \begin{cases} g\left(-\mathbf{u}_{w,t'}{}^{\top}\mathbf{v}_{c,t'}\right)\mathbf{u}_{w,t'}, & t < t', (w, c) \in D^{t'} \\ -g\left(\mathbf{u}_{w,t'}{}^{\top}\mathbf{v}_{c,t'}\right)\mathbf{u}_{w,t'}, & t < t', (w, c) \notin D^{t'} \\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{3.11}$$

Note that in the above expressions, $\mathbf{u}_{w,t}$ and $\mathbf{v}_{c,t}$ implicitly contains the sums over all time points up till t.

3.3.1 Temporal gradient normalization

The gradients in equations (3.10) are dependent on all $\Delta\mathbf{u}_{w,t'}$ for all t' up to t. Therefore, all those $\Delta\mathbf{u}_{w,t'}$ should be updated according to the gradient with respect to $\Delta\mathbf{u}_{w,t'}$. However, this creates accumulated updates of unequal size because later time periods create more updates and, when summed together, this results in a bigger change. To regulate this we propose a heuristic for controlling the scale of the parameter updates at different time points.

By introducing a function, $k(t, t')$, we are able to limit the size of the update in time period t' based on an example in time period t. This allows us to make sure that all updates have the same size, by using a normalized function $norm\_k(t, t')$ defined as

$$norm\_k(t, t') = \frac{k(t, t')}{k(t, \bullet)} \tag{3.12}$$

which will always sum to 1 independent of t.

This function also gives added flexibility to our model, which allows for more experiments. The first function used normalized the updates so that for time period t there would be t updates each of size $1/t$, always summing the size to 1. However, after further reflections, it does not intuitively make sense to update the change vectors in all time periods before t equally. At the current state we could update time period 2 because of an observation in time period 82. If our time periods represents years then we would update a vector based on an observation 80 years later.

Because of these reasons, we decided to experiment with different functions, by updating a fixed set of time periods and sometimes not updating all time periods the same amount. By observing the results from different functions, especially between the function updating all time periods up to t the same and a function only updating the time period of observation, it became clear that this function has a smoothing effect on the result. Therefore, this function and its effect will often be referred to as the smoothing function. Only updating the time period of the observation will be referred to as using no smoothing while full smoothing updates all time periods up to t equally.

As with the regularization, the smoothing is able to be turned off by utilizing the correct function. The function that does this is the function that only updates the time period of the observation

$$k(t,t') = \begin{cases} 1, & t = t' \\ 0, & \text{otherwise} \end{cases} \tag{3.13}$$

Other examples of smoothing functions are the full smoothing function, which updates all time periods up to t equally

$$k(t,t') = \begin{cases} 1, & t \geq t' \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

Another more complex smoothing might be an linearly increasing smoothing over the last 10 years, more as a triangular window, described with the function

$$k(t,t') = \begin{cases} 0, & t - t' \geq 10 \\ 10 - (t - t'), & t \geq t' \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

This function will only update the last ten years and the closer t' is to t the bigger the update. All functions, as described here, will be normalized later on in order to make the entire update size comparable to the same size as when no smoothing is used.

Another positive effect of this heuristic is that some functions speed up training of the change vectors. If N is the number of word-pairs per time period and T is the number of time periods, then the constant smoothing function gives the algorithm a time complexity of $O(NT^2)$. By updating a fixed set of time periods, say K time periods, we limit the time complexity to $O(NTK)$ where K is constant and smaller

Algorithm 1 ALGORITHM

```
1: Input: Dataset $D = \{D_1, \dots, D_T\}$
2: Train $\{\mathbf{u}_{w,1}\}, \{\mathbf{v}_{w,1}\}$ on $D_1$
3: for window $s = 1, \dots, M$ do
4:     for samples $k = 1, \dots, K$ do
5:         if $k = 1$ then
6:             $(w, C, t) \leftarrow$ sample word $w$ with context $C$ from $D^t$
7:         else
8:             $(w, C, t) \leftarrow$ negative sample
9:         end if
10:        for $c \in C$ do
11:            Compute $\nabla_{\Delta \mathbf{v}_{c,t'}} \ell_{w,c,t}$ according to (3.11)
12:            Compute $\nabla_{\Delta \mathbf{u}_{w,t'}} \ell_{w,c,t}$ according to (3.10)
13:            Add $\nabla_{\Delta \mathbf{u}_{w,t'}} \ell_{w,c,t}$ to $\nabla_{\Delta \mathbf{u}_{w,t'}} \ell_{w,t}$
14:            for $t' = 1, \dots, t-1$ do
15:                Compute $k = norm\_k(t, t')$
16:                $\Delta \mathbf{v}_{c,t'} \leftarrow \mathcal{P}_{\lambda k \alpha}\left(\Delta \mathbf{v}_{c,t'} + k\alpha \nabla_{\Delta \mathbf{v}_{c,t'}} \ell_{w,c,t}\right)$
17:            end for
18:        end for
19:        for $t' = 1, \dots, t-1$ do
20:            Compute $k = norm\_k(t, t')$
21:            $\Delta \mathbf{u}_{w,t'} \leftarrow \mathcal{P}_{\lambda k \alpha}\left(\Delta \mathbf{u}_{w,t'} + k\alpha \nabla_{\Delta \mathbf{u}_{w,t'}} \ell_{w,t}\right)$
22:        end for
23:    end for
24: end for
```

than T. The kernel function in equation (3.13) only updates one time period per example, leading to a time complexity of $O(NT)$ updates, which is significantly faster than the constant smoothing.

3.3.2 Pseudo Code

This is all the information needed in order to create the actual implementation of SGD. Now it is possible to create the training algorithm by using these pieces together with the framework of SGD. The pseudo-code of the algorithm is presented in algorithm 1.

In our case there will be a corpus for every time period, which leads to a non-uniform sampling strategy. Instead of sampling a triplet (w, c, t) or a window uniformly from all time periods we will first sample the time period t uniformly from T, which gives us a corpus D^t which is then sampled uniformly for a window. This means that some time periods may be sampled more than others, however it will be fairly equal in the long run. It is assumed that this does not affect the outcome of the algorithm.

The rest of the algorithm follows SGD where the gradient for every pair is calculated with respect to the change-vector. Then, several time periods might be updated depending on the kernel smoothing function, the loops on rows 14 and 19, which adapts the size of the update. Lastly, the regularization results in the projection, line 16 and 21, that projects the original update step, as shown on line 11 and 12, which will make it efficient and easy to implement on top of an SGD implementation. This does not affect the rest of the algorithm in other ways than creating sparse results.

3.4 Implementation

With the original paper on the Skip-gram model, Mikolov et al. [2] released their model in a tool called Word2Vec[1]. Word2Vec is simply run on a big text-file and is able to save word vectors to disk for later use. The code has since been ported to other languages with one of the most used ones being the implementation in publicly available package gensim[2], in Python.

The gensim version of Word2Vec makes use of Python for its high level code while the low level implementations make use of optimized C/Fortran code for efficient vector operations, through Cython, making its performance comparable to optimized C code. This means that the top level framework that needs to be adapted is written in Python while the learning algorithm needs to be rewritten in Cython, or C like code.

The top level framework is changed to accommodate for previously explained functions. Word2Vec parses through the text in order to create its vocabulary but the new version gets a dictionary with pairs of words and number of times they occured in the time period, resulting in the same vocabulary for every time period. Also the model is extended to use matrices of three dimension, where the third dimension is time and the vectors for the first time period needs to be pre-trained and loaded before the actual training. Finally, evaluation methods were added. Methods that can create the resulting vectors for a time period, measure the similarity of two words over all the time periods or find the least/most changed words.

The learning algorithm, the implementation of SGD, is updated to fit algorithm 1 as explained in the previous section. It is adapted to return some more information about its processing, mainly concerning the convergence of the model. Also, λ, the regularization parameter, and the smoothing function are set as parameters.

[1]code.google.com/p/Word2Vec/
[2]github.com/piskvorky/gensim/

4
Related Research

Lately, researchers have experimented with training distributed word representations for different data sets and then compare the resulting models for differences in the training data. By training different models for data sets of different time periods it has been shown that although the resulting vectors are different the relationship between words' vectors is approximately the same. Although the models from the start are static in the sense that they are trained separately and have nothing to do with each other, the vectors will often have similar relationships between them. The similarity between the vectors for the words *cat* and *dog* should be close to the same for two models since it represents the semantic similarity between the words, however, the vector for *dog* might be quite different in the models.

Kulkarni et al. [23] trained entirely separate models on data from different time periods. After training the models for every time period they tried to create a unified coordinate systems for all the models, by aligning the vector spaces of two consecutive time periods. They assumed that the two vector spaces were equal under a linear transformation and that most words do not change meaning from one time period to another, which indicates that the relationship between words should in some way be preserved. Thus they rotate one of the models so as to minimize the change between the two time periods, leaving most words nearly unchanged. The words that did not fit this alignment are then said to have changed during the interval. With this method they are able to observe words with a statistically significant change in word meaning. Their evaluation includes the most changed words and examples of how the words differs in usage from the early to the late time periods they observe.

Kim et al. [24] trained models for every year 1900-2008 on the Google Books Ngram corpus [25]. They chronologically train separate models for every year, though they use the resulting vectors for the previous year as the initialization for the next year, instead of randomly initialising the vectors as in Word2Vec. By doing this they seem to get a unified coordinate system for all time periods without having to align the vector spaces afterwards. Their technique successfully identifies words whose usage have changed and by comparing the similarity of these words to other words across time they are able to track, or make sense of, the changes. By looking at how a word, over time, becomes more or less similar to other words they are able to give some intuition of how the word has changed. They also conclude that their model includes a lot of noise, or a random drift, where every word change every time period while the sum of changes can be close to 0 over long periods of time, thus calling it noise.

Another application of training separate models and linking them has been done in machine translation and it has shown to be surprisingly effective [11]. By training separate models for the different languages and then use a translation matrix in order to convert a vector from one model's vector space to the other it is possible to find the translation of a word. The translation matrix suggests that there exists a linear transformation of the vector spaces, turning and shrinking or expanding the vector space, that makes the models almost equivalent, where equivalent models would have the same vectors for the same words. However, the best results are achieved when the language translated from have vectors of higher dimensions, on the scale 2-4 times higher, than the language translated to. This means that in order to get the best results in both directions, four models would be needed.

5 Experiments

This chapter presents a set of experiments designed to analyze the performance of the model. It presents a set of result from the experiments that will later be used to evaluate whether the model fulfills the goals of the project. The experiments are also be conducted on another model, as introduced by Kim et al. [24] from the related research, in order to say something about how the model compares to its competition.

5.1 Dataset

The data set used for training all models is the Google Books ngram corpus [25]. It contains n-grams from over 8 million books, which is almost 6% of all books ever published in english. For the tests 5-grams are used, meaning sequences of 5 words from a sentence. The models will make use of 20 million 5-grams per year, for the years 1901-2008.

By training the model on 5-grams we limit our window size to 4. It is impossible to create pairs of words with more than 3 words between them since we only get 5 consecutive words in a row. However, this has been shown not to affect the end result of Skip-gram models considerably [26] and should therefore not affect the results of our experiments enough to be taken into account.

To get a grip of how much 20 million 5-grams actually is, some comparisons can be made. If a book has 400 pages and 250 words per page, then 20 million 5-grams is as much text as 1000 books. If we do not consider that the 5-grams might overlap, which they probably do, then it can be thought of as if the model is trained on approximately 1000 books per year. This is about 500MB of raw text data per year.

To make the comparisons more fair, all models will make use of the same set of vectors for the year 1900. These vectors will be pre-trained in the same way the sequentially trained model is trained, presented in the next section, for the years 1850-1900. Thus all trained models will start with the same set of vectors for the year 1900 which will give as fair of a comparison as possible.

All models will also make use of the same vocabulary, that is the words to be trained. The same words are trained for every year, in order to be certain that every word has a word embedding in every time period. This vocabulary consists of all the words that occurs more than 18 times in some year 1900-2008, based on the 20 million 5-grams for those years. Thus some words might not be used during the first decade and some might die off at some point, however tracking words that occur in all the given years would remove too many words that have been born during the 20th century.

5.2 Models

The model described in the work by Kim et al. [24] and explained in section 4 is used in all comparisons. This model has been shown to automatically find words with lots of change in word meaning, and also intuitively explain their changes by comparing words over time, which will make for good comparisons. It should be noted although an implementation based on their paper will be used, the same model can be achieved with our model if training sequentially, using the smoothing function in (3.13) and not using any regularization. This model will be referred to as the sequentially trained model, STM for short.

In the experiments several different versions of the model are used, in order to measure and reason about the effect from the smoothing and the regularization. We always tried to choose the best parameters for the different models, however due to limited time this might not always be the case.

The first and simplest model is the one that makes no use of the regularization and smoothing, meaning that the regularization parameter, $lambda$, is set to 0 and that the smoothing function $k(t, t')$ is set as in equation (3.13). This model is labeled BASE in the tests.

There are two models that makes use of the smoothing but not the regularization. Their regularization parameter is set to 0, but they use different smoothing functions. The first model uses a full smoothing, meaning that all time periods from 1 to t are updated the same. The function is the one in equation (3.14) and this model is labeled FS in the tests (for Full Smoothing).

The other model makes use of the linearly increasing smoothing function for the last 10 years, as described in equation (3.15). Thus the last 10 time periods are updated where time periods closer to t receive a bigger update. This model is labeled 10YS, for 10 year smoothing.

There is also one model that makes use of the regularization but not any smoothing. Thus the smoothing function is the one in equation (3.13) and the regularization parameter is set to fit the data. This model will be labeled REG, for regularization.

Lastly, a model using both the smoothing and regularization is tested. This model uses the same smoothing function as 10YS and a regularization parameter set to obtain the best results. This model is labeled SREG in the experiments.

It should be noted that most of the results will be hard to validate, however, it is possible to reason about them fairly well by evaluating and reason about the performance of the different models.

5.3 Results

When the models have been trained on the data it is time to get results for different tasks. It is important that the results show how words change meaning over time, both for some of the most and some of the least changed words. Also a human labeled data set will be evaluated for the different models in order to get a quantitative result.

First, the models are compared on one of the most changed words during the time period. In order to make sense of the word changes over time a word is picked and its similarity is plotted to some other words over time. Thus makes it fairly easy to understand how the words change their meaning by observing how similar the model thinks it is compared to other words. In figure 5.1 we observe exactly this. We have picked the word *gay* and plot the similarity between *gay* and other words. From this we can observe that *gay* is more similar to the words *bright* and *cheerful* in the beginning of the 20th-century and more similar to *bisexual* and *lesbian* at the end of the 20th-century. It is also possible to observe when the shift in similarities took place, which seems to agree with our intuition that the word *gay* changed its meaning during the 80s. The same plot is created for all the models for an easy comparison. All models are explained and given names in section 5.2.

The word 'gay' was chosen because it is one of the most changed words that also occurs a certain amount of times during the time period. The words to compare it with was chosen because they are the words that, according to the model, are most similar in some time period. The words 'bright' and 'cheerful' are among the 5 most similar words to gay in the year 1900 and the words 'lesbian' and 'bisexual' are among the 5 most similar words to the word gay in the year 2008.

It is easy to spot the similarities and the differences between the different models. It looks like all models basically models the same thing although their differences because of smoothing and regularization aer noticeable.

5.3.1 Comparing to human raters

Gulordava et al. [27] created a data set with human rated changes in word meaning over time. 100 words were rated on the scale 0-4 where the question was "How much has the given word changed its meaning in the last 40 years?". There were 5 raters, each giving a value 0-4 to every word. By taking the average of the raters we measure change in word meaning for that word and we try to correlate it to the measurements of our models. Thus for every word in their data set we measure the change in our models and then calculate the Pearson's correlation [28], a measure of linear correlation, between the two values for all words. The Pearson's correlation is

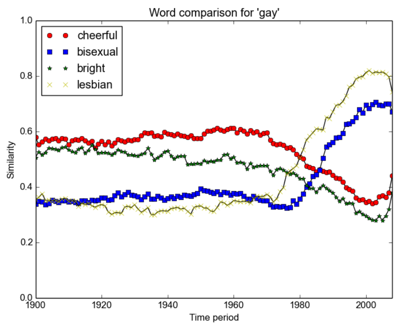


Figure 5.1: How STM models the semantic similarity between gay and some other words and how they have changed during the 20th-century.

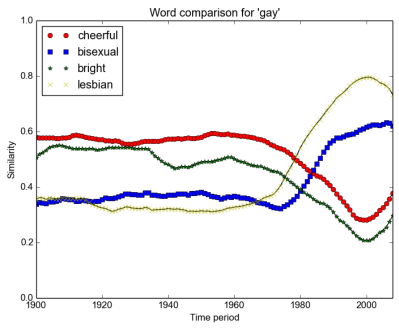


Figure 5.2: How 10YS models the semantic similarity between gay and some other words and how they have changed during the 20th-century.

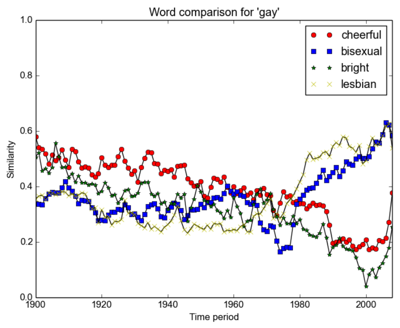


Figure 5.3: How BASE models the semantic similarity between gay and some other words and how they have changed during the 20th-century.

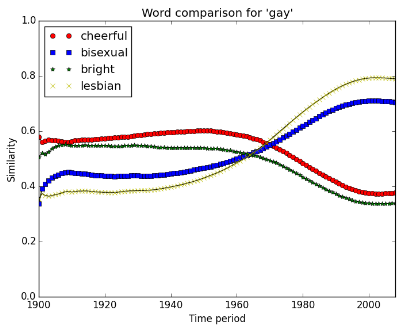


Figure 5.4: How FS models the semantic similarity between gay and some other words and how they have changed during the 20th-century.

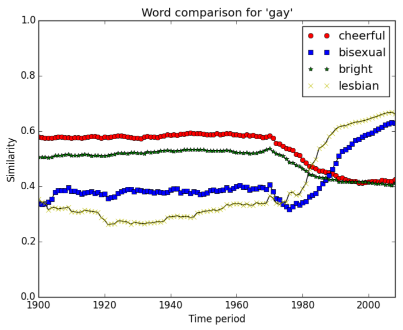


Figure 5.5: How REG models the semantic similarity between gay and some other words and how they have changed during the 20th-century.

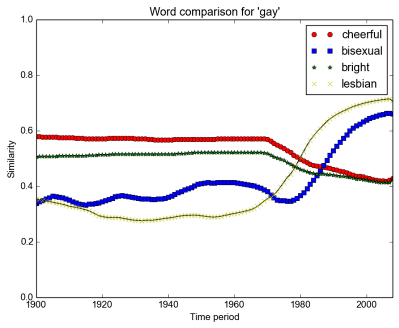


Figure 5.6: How SREG models the semantic similarity between gay and some other words and how they have changed during the 20th-century.

chosen because Guilordava et al. uses it to measure how well their models are doing.

The table below shows the Pearson's correlation between the human raters and the different models used during testing.

Model	**Correlation**
STM [24]	0.6035
BASE	0.5436
FS	0.5283
10YS	0.5914
REG	0.3931
SREG	0.4813
sim-HR [27]	0.386
freq-HR [27]	0.301

The sequentially trained model achieves the best result, closely follow by the model with the smoothing function for the last 10 years. The models that use regularization gets the worst results, though all models perform better than those in the dataset's original paper.

In this test we try to compare the correlation between a cosine similarity and the average of values from an ordinal scale, which is not a perfectly fair comparison. It is unclear how this gets affected by properties such as regularization but the cosine function is not linear, which should affect the result as a linear correlation is calculated.

5.3.2 Words without changes

The figures from 5.7 to 5.12 show how much some common words change during the time span. These words should not change at all, since their meaning is the same today as it was in the year 1900. By comparing the vector for time period t to the vector for the first time period for the same word it is possible to measure its change during the period. As we can see the delta model's vectors tend to change quite a bit. The smoothing seem to stop this to some extent, while the regularization stop the changes completely. The sequentially trained model changes approximately as much as the FS model, showing that even though it has a lot of noise the words do not drift away from its original meaning.

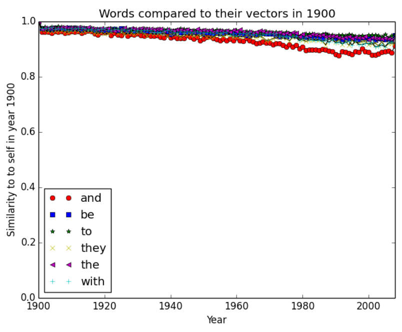


Figure 5.7: Word similarities compared to their vectors for the year 1900 for the model STM.

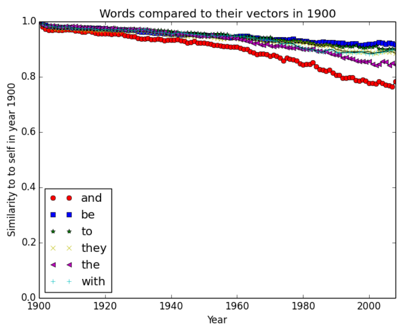


Figure 5.8: Word similarities compared to their vectors for the year 1900 for the model 10YS.

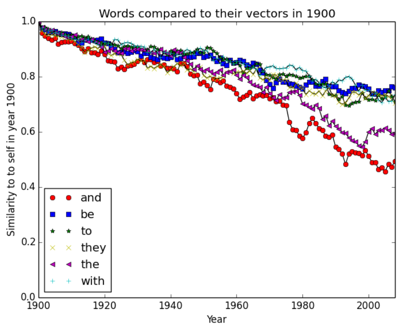


Figure 5.9: Word similarities compared to their vectors for the year 1900 for the model BASE.

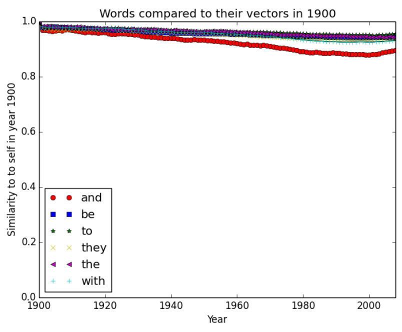


Figure 5.10: Word similarities compared to their vectors for the year 1900 for the model FS.

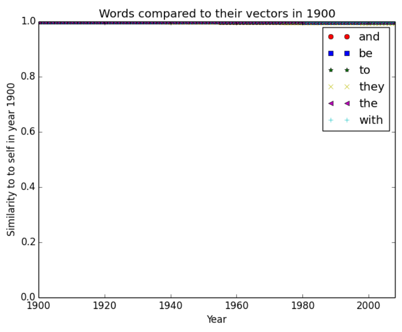


Figure 5.11: Word similarities compared to their vectors for the year 1900 for the model REG.

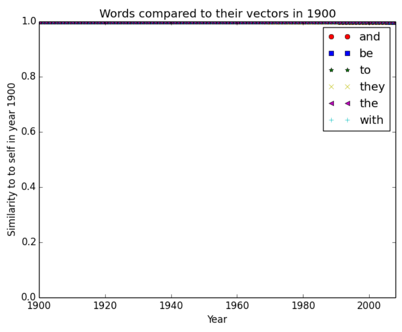


Figure 5.12: Word similarities compared to their vectors for the year 1900 for the model SREG.

6 Discussion

In this thesis a model able to track the temporal evolution in word meaning for a set of words has been created. The model is very general in the sense that it can adapted in the terms of smoothing and regularization, both of which can be changed in ways that effectively creates different results.

Creating the simplest of models, and not using any regularization or smoothing, gives results which show that the model is able to observe changes in word meaning over time and also pinpoint approximately when these changes happened. However, such a model is very noisy, making it hard to interpret it correctly. It also shows that common words, which should not have changed word meaning, tend to change more and more from its original meaning the longer the time goes. This suggests that something happens to the entire vector space making comparisons between words in time periods far apart not applicable.

There seems to be several ways of overcoming these flaws, one such way is to train the model sequentially. By doing this we improve all our measured results. The model correlates its measured changes better to that of human raters, there is less noise and the words that should not change does not change as much as in the simplest model, while still changing to some extent. Overall the results are good, but can they be made even better?

By taking advantage of additional information known about language we are able to create models which should be closer to a "true" model of word meanings over time. Smoothing takes advantage of the fact that the underlying function that models changes in word meaning is smooth and not noisy, while regularization helps to model the fact that only a few words change meaning from one time period to another. By adding these concepts to the model we hope to get a better representation of word meanings over time.

Adding a constant smoothing function that introduces a dependency between all time periods has a big impact on the results. Changes in word meaning tend to be a lot less noisy which was the purpose of the smoothing. However, it also has other effects. While studying when changes takes place, this model tend to have its changing points earlier than the other models. It also has some edge effects, where there often are more changes in the first time periods and less changes in the last ones. This is understandable, but not correct, because earlier time periods will be updated more often.

One way to get rid of some of these effects is to claim that the dependency between time periods does not extend through the entire time span. An example of this is the model that utilizes a linearly increasing smoothing function for the last 10 years. Thus, only time periods within the last 10 years are affected by changes from some example. This model still has more smooth change than the simple model, however, the changes are not as smooth as before. The edge effects are as good as gone, as can be observed through graphs, and the result from the correlation to human raters is better than that of the simple model.

The other technique used to tried to overcome some of the flaws of the simple model is regularization, which is introduced in the objective function. By doing this we are able to limit the amount of changes that is done between two time periods, thus trying to prevent the noise that is otherwise created. The regularization seems to give the expected results, where the noise is significantly less and lots of change vectors become 0, as wanted. The regularization also prevents changes in the words that should not change over the entire time span which should make a word w_1 comparable to a word w_2 even in different time periods. The regularized models, however, does yield significantly worse results when correlating their measured changes to that of human raters. This is however not entirely unexpected. By introducing regularization we do not only shrink all changes linearly, we try to enforce sparsity which means that words with little change will be modeled as having no change at all. This should affect the results of the correlations, since we try to fit a linear correlation where it is not perfectly applicable.

When combining smoothing and regularization we get some of the wanted features from both, however with some unexpected effects. Changes tend to be more smooth, as done by the smoothing function, and words that should not have any change does not have any change over time. Though some words, such as the word "bisexual" when compared to "gay", seem to change more when using both regularization and smoothing (see figure 5.12), which is somewhat unexpected. Though it should be noted that the word "bisexual" has a very low frequency in that time period, but the effect that frequency has on the results will not be investigated further.

Regularization, when set with a parameter high enough, limits the end result of the model. Even the most changed words does not adjust as much as they should, creating a model with worse results. The regularization is in that sense very sensitive and it can be expected that this happens to words with less change if the parameter is somewhat high. Also, the parameter values for the regularization, and to some extent also the smoothing, has mostly been set from observing the results in graphs such as those in section 5. This could be made more rigorous where the resulting models are analyzed more closely in order to get a model with both regularization and smoothing that better represents the changes. This could probably improve the results of some of the models and is possibly the easiest future work in order to yield better results.

7
Conclusion and Future Work

The goal of this thesis was to create a dynamic model able to track the temporal evolution in word meaning for a set of words. The model had some focuses that it should be able to do and some key results that should be evaluated.

The model created is dynamic in the sense that the smoothing function and the regularization can be adapted which has a shown effect on the results. By making use of time varying word embeddings the model is able to track the temporal evolution in word meaning, although maybe not perfectly. The model is able to measure the amount of change in a given word's meaning and can thus find the words with the most change automatically. The amount of change in the words is shown to correlate with that of human raters. The model lives up to the goals set, however it is very hard to analyze how well. More analysis could be done on the model in general.

The model could be further developed in several ways. First, a more rigorous evaluation could be done on the effects of regularization and smoothing. Also attempts of training sequentially using regularization could be made. Techniques such as updating the training examples in batches, instead of updating for every example, could also be tried.

The model could somehow take frequency into account. It is unclear how a change in frequency (over time) of a word changes its semantic meaning, or the observation of the change in semantic meaning. The regularization parameter could be dependent on a words frequency, and not be constant. It could also be dependent on the time period, where different time periods could allow for more change, by a lower lambda, since we know that more change happened in that time period.

Throughout this thesis, the phrase "word *meaning*" has been used. However, it is unclear what word meaning actually implies, and maybe word usage would be a better term. Though by training a model where the data has POS-tagging or even word sense disambiguation, we could get a unique identifier for different meanings of the same word. Other things such as word stemming could be done on the input, using only a common base form for all words in order to improve results.

Overall, this thesis creates a model that is able to do a lot of things and opens up for opportunities to do them better. Through more rigorous analysis, better annotated input data or by extending the model further.

Bibliography

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *The Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[3] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pp. 873–882, Association for Computational Linguistics, 2012.

[4] G. E. Hinton, "Distributed representations," 1984.

[5] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954.

[6] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[8] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 384–394, Association for Computational Linguistics, 2010.

[9] P. D. Turney, P. Pantel, *et al.*, "From frequency to meaning: Vector space models of semantics," *Journal of artificial intelligence research*, vol. 37, no. 1, pp. 141–188, 2010.

[10] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.

[11] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *arXiv preprint arXiv:1309.4168*, 2013.

[12] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 142–150, Association for Computational Linguistics, 2011.

[13] W. Y. Zou, R. Socher, D. M. Cer, and C. D. Manning, "Bilingual word embeddings for phrase-based machine translation.," in *EMNLP*, pp. 1393–1398, 2013.

[14] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in Neural Information Processing Systems*, pp. 2177–2185, 2014.

[15] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[16] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[17] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[18] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight, "Sparsity and smoothness via the fused lasso," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 1, pp. 91–108, 2005.

[19] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[20] C. M. Alaíz, Á. Barbero, and J. R. Dorronsoro, "Group fused lasso," in *Artificial Neural Networks and Machine Learning–ICANN 2013*, pp. 66–73, Springer, 2013.

[21] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[22] K. Bleakley and J.-P. Vert, "The group fused Lasso for multiple change-point detection." Technical Report, June 2011.

[23] V. Kulkarni, R. Al-Rfou, B. Perozzi, and S. Skiena, "Statistically significant detection of linguistic change," in *Proceedings of the 24th International Conference on World Wide Web*, pp. 625–635, International World Wide Web Conferences Steering Committee, 2015.

[24] Y. Kim, Y.-I. Chiu, K. Hanaki, D. Hegde, and S. Petrov, "Temporal analysis of language through neural language models," *arXiv preprint arXiv:1405.3515*, 2014.

[25] Y. Lin, J.-B. Michel, E. L. Aiden, J. Orwant, W. Brockman, and S. Petrov, "Syntactic annotations for the google books ngram corpus," in *Proceedings of the ACL 2012 system demonstrations*, pp. 169–174, Association for Computational Linguistics, 2012.

[26] F. Ginter and J. Kanerva, "Fast training of word2vec representations using n-gram corpora," 2014.

[27] K. Gulordava and M. Baroni, "A distributional similarity approach to the detection of semantic change in the google books ngram corpus," in *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, pp. 67–71, Association for Computational Linguistics, 2011.

[28] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*, pp. 1–4, Springer, 2009.