Minimization by Majorization for a quadratic hinge loss adaptation of word2vec

Msc Econometrics and Management Science Thesis

Koen de Nijs - 412673

Supervisor: prof. Patrick Groenen

January 21, 2021

Abstract

The popular word2vec model allows us to learn vector representations of words that map their properties and similarities to other words. The resulting 'word embeddings' can show the closest matching synonyms to a particular word, and even allow us to solve analogical reasoning tasks such as 'King' stands to 'X' as 'Man' stands to 'Woman', where 'X' needs to be solved for (e.g. 'Queen'). Although already highly optimised, remaining inefficiencies may include the gradient descent method of optimization, and the required sigmoid function for calculating the cross-entropy loss. We propose an alternative algorithm for word2vec which optimizes the word embeddings with minimization by majorization, as well as an implementation of this algorithm which substitutes the cross-entropy loss for a quadratic hinge loss function. We compare the two specifications of the algorithm on computation time per iteration, as well as classification accuracy and an analogical reasoning task set. Several sampling methods of the choice space are proposed to scale the model to larger vocabularies. We find that the minimization by majorization algorithm converges to a parameter set with better classification accuracy than our implementation of the conventional word2vec method. The quadratic hinge implementation is found to offer better performance on analogical reasoning tasks than the cross-entropy loss model. We also find that subsampling of the choice set in training the model makes the model readily scaleable to sizeable vocabularies, and that uniform random sampling offers better performance than frequency-based sampling or sampling from the word context.

Contents

1 Introduction

In 2014, Google introduced word2vec, a shallow neural network classification algorithm for deriving word probabilities from their surrounding k words (Continuous Bag of Words), or vice versa, predicting the surrounding k words around a particular word (Skip-Gram). As it turned out, the word-specific estimated weight vectors (also named word embeddings) linking a particular word to the surrounding words can be used to derive semantic relationships. Specifically, similar, synonimic, words will have similar weight vectors, as the two words will be used interchangeably and may therefore be equally probable for a particular word context. Finding a synonym for a particular word is therefore as simple as finding the word with the most similar parameter vector. Besides synonimy, we may also use the weight vectors to derive more sophisticated word relations, such as "X stands to Spain" as "Paris stands to France", which may be solved by Vec(X) = Vec(Paris) - Vec(France) + Vec(Spain), for which a viable solution is 'Madrid'. If the text used to train the model contains all these terms and makes their relation apparent, the word2vec model will be able to solve this analogy. This means that the word2vec word embeddings may be used not just for filling in some blanks, but that they may also have ontological value.

A particular issue in estimating word embeddings for large vocabularies, is that of computation time. For a model with a word embedding size of c, estimated for a vocabulary of size M, we require the estimation of $2Mc$ parameters (for input- and output-weights). With c typically in the hundreds, and M likely to be in the ten-thousands for a sizeable text corpus, this poses a magnificent computation challenge, particularly when we train the model on text corpora numbering millions to billions of total words. Several alterations to accelerate word2vec have already been proposed. Particularly, to use hierarchical tree-based softmax for calculating word probabilities, as well as the use of negative sampling, which only adjusts the weights of a small subset of words for which we want a negative outcome.

We propose to further accelerate the word2vec model by borrowing from Support Vector Machine (SVM) optimization techniques, specifically the use of minimization by majorization, as well as the use of a quadratic hinge loss function. Minimization by majorization has the benefit of guaranteed descent of the loss function across iterations. The stochastic gradient descent method used in the conventional word2vec does not have this characteristic and may more easily get stuck in the likelihood space Wang (2008). The proposed quadratic hinge function is expected to be computationally less expensive than the sigmoid function used in the word2vec model. The quadratic hinge only requires that we evaluate a word 'score' z against a particular threshold, then square the difference between the score and the threshold. The cross-entropy loss function used by the conventional word2vec requires the calculation of $\exp(z)$, and then divides this exponent by the exponent of the scores of all the other possible words, which we expect to be relatively burdensome. The quadratic hinge also has the benefit of only attaching a loss to misclassified instances. This may help more efficient optimization efforts than the word2vec model, which incurrs a loss based on the estimated probability of correct classification. To test the benefits of both the MM algorithm and the quadratic hinge loss function, we aim to answer the research question:

Can we use a quadratic hinge loss function optimized with minimization by majorization to accelerate word2vec learning, without sacrificing word embedding quality?

To answer this research question, we describe and implement an MM-optimization of the 'continuous bag of words' version of word2vec, which predicts a word based on which words directly surround it in the text. Our research has three key contributions. First, we study whether minimization by majorization (MM) is a suitable method of optimization for the word2vec model. Second, we propose an alternative loss function: the quadratic hinge loss, which may be less expensive to compute. Third, we consider the issue of negative sampling for the MM-optimized algorithm, comparing different techniques of sampling to improve the scalability of the proposed models. We use an incremental approach, answering the following three research sub-questions:

RQ1: Can minimization by majorization be used to accelerate word2vec learning?

The traditional word2vec algorithm uses a method of stochastic gradient descent. For each word in the text corpus, the probabilities of positive classification for each word in the vocabulary are calculated. The gradient of the loss function with respect to the parameters is then calculated, and the parameters are increased with a given step size in the descent direction. This method requires a suitable step size, and risks overshooting the descent, or otherwise taking steps that are too small. Minimization by majorization performs iterative minimization of a larger function than the loss function. This allows the algorithm to achieve a decrease in the loss function with each iteration. The MM algorithm may therefore find it easier to converge to the best set of parameters for the text corpus. To answer the first research question, we implement a MM algorithm of the regular word2vec loss function, adapting a previous implementation from Groenen and Josse (2016), and compare its optimization to the stochastic gradient descent method. We observe progress in the loss function over time to test whether the MM algorithm makes quicker progress than the gradient descent algorithm.

RQ2: Does the use of a quadratic hinge loss function over the traditional cross-entropy loss accelerate parameter learning, and are the resulting embeddings of equal quality?

The word2vec loss function is normally a cross-entropy loss, which aims to maximize the sigmoid transformation of the score attached to the correct word. The cross-entropy loss can be interpreted as the log sum of the estimated probabilities of the correct words. This carries the risk that the loss function may be occupied with increasing the certainty of classifications that are already correct, rather than reclassifying incorrect observations. The quadratic hinge loss function attaches a positive loss only to incorrect classifications, and does not require the expensive sigmoid transformation. We are interested to know whether this may make optimization more efficient, and improve classification performance. We study the time per iteration for both the MM-optimized cross-entropy loss and the hinge loss function, and evaluate classification performance by means of in-sample and out-of-sample classification accuracy. This allows us to quantify whether the hinge loss function is equally suited to the text classification task performed by the word2vec model, and whether it can achieve a higher accuracy in a given amount of training time.

RQ3: Can we use negative sampling to scale the MM-optimized models to larger text corpora?

To extend the models to applications with large text corpora and vocabularies, it may be overzealous to consider the whole vocabulary as the classification space for each word instance. In these larger implementations we may employ negative sampling, in which only a selection of the vocabulary compete with the correct word. Mikolov et al. (2013b) show that their word2vec model benefits from a frequency-based negative sampling method in which frequent words are more often used as competing samples. We aim to test their result for the MM-optimized word2vec model as well as the hinge loss model, and compare it to random uniform sampling, as well as context sampling, in which we use the surrounding words as competitors to the correct words. We first test the models with increasing text-corpus and vocabulary sizes to observe the scalability of the sampled methods. We then test the sampled models on classification accuracy, as well as a word analogy task (e.g. solve 'X' stands to 'Amsterdam' as 'Belgium' stands to 'Brussels'), to ensure that the word embeddings show the same semantic characteristics as the traditional word2vec. We are interested in comparing the performance of the hinge loss and cross-entropy loss models in a sampled application, as well as finding which method of sampling best suits each model.

These three sub-questions together allow us to evaluate the primary research question, assessing the efficiency of an MM-optimized quadratic hinge word2vec model, and allow us to determine whether word2vec can benefit from MM-optimization, and whether it can benefit further from a quadratic hinge loss function.

We find that in smaller text corpora N, the MM-optimization of word2vec is faster to minimize the cross-entropy loss to within a relative convergence tolerance, and converges to a set of word embeddings with a higher classification accuracy.

We find that the MM-optimized quadratic hinge loss function offers an improvement in accuracy on the word analogy task when compared to the MM-optimized cross-entropy loss, but generally shows slightly lower classification accuracy. Training with the quadratic hinge loss results in word embeddings with intuitive word relations, and appears to present the same semantic properties offered by the conventional word2vec algorithm.

Negative sampling is found to greatly increase the scalability of the MM-optimized models to larger vocabularies. We find uniform random sampling to result in the best classification accuracy, as well as the best performance on the word analogy task.

On an application on a larger corpus of $N = 17{,}000{,}000$, we find that the MM-optimized models underperform on the word analogy task relative to a gensim implementation of stochastic gradient descent with identical hyperparameters. The quadratic hinge loss offers the most competitive performance on the word analogy task presented by Mikolov et al. (2013b) with 8.5% of tasks solved correctly per the provided solution, but this is still significantly less than the gensim gradient descent word2vec performance of 32.0%. Overall, the MM algorithm is found to be a thorough optimizer of the word2vec loss in smaller corpora, but not fast enough to compete with stochastic gradient descent on a larger scale. The MM-optimized quadratic hinge loss shows notable promise, it achieves faster iterations, and beats out the MM-optimized cross-entropy loss on the word analogy task on every scale of implementation.

We continue with a summary of the literature on word2vec, after which Section 2 defines our methods, and Section 3 reports the data sets used. Section 4 reports the results per research sub-question. Finally, Section 5 offers a conclusion, and Section 6 presents a discussion of our limitations and considerations for future research.

1.1 Word2vec in the Literature

Word2vec was first introduced in 2013 by Mikolov et al. (2013b), although the concept of word embeddings may be traced back to Rumelhart et al. (1986), who note the weights in a neural network as containing important features of the task domain. Mikolov et al. (2013b) describe two methods of obtaining vector representations of words by learning from their co-occurrence in a text corpus, the skip-gram and the continuous-bag-of-words models. The first model trains vector representations by finding the set of vectors that best predict the words surrounding a particular singular word. Inversely, the continuous-bag-of-words model, which we study in this thesis, is trained by finding the set of vector representations that best classify a target word, based on the surrounding 'bag' of context words. The classification is improved by performing stochastic gradient descent with a cross entropy loss function. This loss function has a probabilistic interpretation, and is identical to the loss function used in multinomial logistic regression, as introduced in econometrics by McFadden (1973). This essentially makes the word2vec model a large-scale multinomial regression problem, in which we aim to predict the choice out of M words in the vocabulary, with as our predictors the binary encoding of the word context of each of the other words in the vocabulary, although the dimensionality of the parameter space is reduced as not to require M^2 parameters.

Mikolov et al. (2013b) find that word2vec training, whilst already relatively fast for a neural network, can be sped up with undersampling of the most frequent words, to the benefit of task performance for the less frequent words in the vocabulary. Specifically, to drop a significant portion of common words such as 'the', prevents the model from overtraining on these words, and increases performance for rare words such as 'ninjutsu'. The authors also extend the model to consider common multiword phrases as singular words, to extend the vocabulary to personal, company and brand names. Additionally, the traditional softmax is supplanted by a hierarchical tree-based softmax calculation. This saves the model from having to evaluate each of M nodes in the output layer, reducing this to an average of $\log_2(M)$ nodes. By substituting the cross-entropy loss in word2vec by a hinge loss function, we negate the need for calculating the softmax entirely, which may further accelerate the word2vec as presented in Mikolov et al. (2013b).

The word2vec model has been adapted to domains outside of word pattern recognition. Notably, Chung et al. (2016) propose 'audio word2vec', finding that the word2vec may be successfully adapted for spoken term detection, outperforming conventional methods. The model may also be estimated with unsupervised learning, in which the model, similar to conventional word2vec, finds 'sound-alike' audio segments by comparing vector representations. Another adaptation of word2vec outside the natural language domain is presented by Ng (2017), who note the potential of the model for reducing the dimensionality of vector representations of long DNA strands. Although our application will be limited to the NLP domain, we note the potential value for a word2vec acceleration across other fields.

2 Methodology

We first define the theoretical framework of the encoding of a text corpus and the structure of the word2vec model in section 2.1. The stochastic gradient descent method of word2vec optimization, which will serve as a baseline, is given in section 2.2. To answer our first research question, evaluating the efficiency of MM-optimization for word2vec, we derive a majorizing function of the cross-entropy loss in Section 2.3. For our second research question, the evaluation of a quadratic hinge version of word2vec, we derive a majorization of the loss function in Section 2.4. To answer our third research question on the scalability of optimization with negative sampling, we present separate sampled majorizations of both loss functions as well as several negative sampling methods, in Section 2.5.

2.1 Word2vec Framework

The use case of the word2vec model is as follows. We consider the case of trying to predict a word instance i out of a total text corpus of N words, with M total words in the possible vocabulary. The text corpus may be considered a continuous space-separated string of words. For ease of interpretation, we will introduce the example text corpus '$a\ b\ a\ c\ b\ b\ d$', where the letters a to d take the place normally taken by complete words. The text corpus has length $N = 7$ and vocabulary size $M = 4$. We may encode such a corpus by a one-hot encoded matrix $\boldsymbol{Y}$ of dimensions $N \times M$, which in this case will be specified as:

$$\boldsymbol{Y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To estimate the word2vec model, we also need to denote the word context of a particular word i in the corpus. For a context size of k, this means defining a $N \times M$ matrix $\boldsymbol{Q}$ that reports in each row vector $\boldsymbol{q}_i$ which of the M words were in the context of word instance i.

Specifically, for our example '$a\ b\ a\ c\ b\ b\ d$' (if we let k = 2):

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Note that $q_{im} = 1$ for every word l for which $y_{jl} = 1$, with $j \in [\max(1,i-k/2), \min(M,i+k/2)]$. If we want to know only the surrounding words for word instance i, excluding the encoding for the word itself, we may take $\boldsymbol{X} = \boldsymbol{Q} - \boldsymbol{Y}$, where $\boldsymbol{X}$ is an $N \times M$ matrix encoding the surrounding words for each word instance. For our example, this would be:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Note that under this specification, we encode only the presence of a word in the context, not its frequency. The goal of the word2vec model, is to predict the outcomes encoded in $\boldsymbol{Y}$ with the word context given by $\boldsymbol{X}$. This may be done by defining an $M \times M$ matrix $\boldsymbol{B}$, with elements β_{ij}. The weights β_{ij} reflect the relation of co-occurrence of words i and j. A higher weight reflecting high co-occurrence, and a low weight reflecting very low co-occurrence. With these weights, we can calculate word probabilities from their context. For example, we may specify:

$$P[y_{im} = 1|\boldsymbol{x}_i] = \frac{\exp\left(\boldsymbol{x}_i'\boldsymbol{\beta}_m\right)}{\sum_l^M \exp\left(\boldsymbol{x}_i'\boldsymbol{\beta}_l\right)} = \pi_{im},$$

where $\boldsymbol{x}_i'$ is row i of matrix $\boldsymbol{X}$, and $\boldsymbol{\beta}_m$ is column m of matrix $\boldsymbol{B}$. However, this requires the estimation of M^2 parameters. For large corpora, with M often in the ten-thousands, this may require more parameters than the number of unique observations in the corpus. This creates a prohibitively large likelihood of overfitting the parameter set to the data, negating the practical use of the model. The word2vec specification therefore restricts the $M \times M$ parameter matrix to be the matrix product of two smaller parameter matrices ($\boldsymbol{U}$,$\boldsymbol{V}$), where both $\boldsymbol{U}$ and $\boldsymbol{V}$ are $M \times c$. With c typically in the range of 50-300, we define

$$\boldsymbol{B} = \boldsymbol{U}\boldsymbol{V}'.$$

This reduces the total number of weights to be estimated to $2Mc$ parameters, the elements of $\boldsymbol{U}$ and $\boldsymbol{V}$. This model definition can be interpreted, as it is presented in Mikolov et al. (2013b), as a neural network with a single hidden layer of size c. The first matrix $\boldsymbol{U}$ contains the parameters vectors $\boldsymbol{u}'_m$ that report how the occurrence of a particular word m activates the c nodes in the hidden layer. The second matrix $\boldsymbol{V}$ contains the parameter vectors $\boldsymbol{v}'_m$ that reflect how the activation of the hidden layer is related to the likelihood of a particular word m being chosen for the word context. Together, the parameter vectors ($\boldsymbol{u}'_m$,$\boldsymbol{v}_m$) give the word embedding of a particular word m. Word pairs (m,l) with high similarity between their word embeddings are likely to be synonymic or thematically similar. Dissimilarity between their word embeddings suggests low co-occurrence. We define the $N \times M$ matrix $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'$. For a given set of parameter matrices ($\boldsymbol{U}$, $\boldsymbol{V}$), the matrix entry $z_{im} = \boldsymbol{x}'_i\boldsymbol{U}\boldsymbol{v}_m$ reflects the score used in the exponent for calculating the estimated probability π_{im}. A high score z_{im} therefore reflects that for the word instance i, we attach a high probability to the word m being fitting, given the context $\boldsymbol{x}'_i$.

2.2 Stochastic Gradient Descent word2vec

We now define the method by which the conventional word2vec model optimizes the parameter matrices $\boldsymbol{U}$,$\boldsymbol{V}$, as presented by Mikolov et al. (2013a) (another thorough explanation of the algorithm is given by Rong (2014)). The model framework given in the previous section can be used to predict word occurrences from their surrounding text, but requires a method to find the set of optimal parameters $\boldsymbol{U}$,$\boldsymbol{V}$ that attach a high probability to the correct words, and low probability to the wrong words across our corpus of N words. To this end, we need to define a loss function that reports goodness-of-fit for a parameter set. The conventional word2vec model makes use of the cross-entropy loss. For each word instance i, we have an estimated probability $P[y_{im} = 1|\boldsymbol{x}_i] = \pi_{im}$. The cross-entropy is derived from the total estimated likelihood of observing the text corpus, which we define as

$$\begin{aligned}\mathcal{L}(\boldsymbol{U},\boldsymbol{V}) = \prod_{i=1}^{N}\prod_{m=1}^{M}\pi_{im}^{y_{im}} &= \prod_{i=1}^{N}\prod_{m=1}^{M}\left(\frac{\exp\left(\boldsymbol{x}_i'\boldsymbol{\beta}_m\right)}{\sum_{l=1}^{M}\exp\left(\boldsymbol{x}_i'\boldsymbol{\beta}_l\right)}\right)^{y_{im}} \\ &= \prod_{i=1}^{N}\prod_{m=1}^{M}\left(\frac{\exp\left(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m\right)}{\sum_{l=1}^{M}\exp\left(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_l\right)}\right)^{y_{im}},\end{aligned}$$

where $\boldsymbol{v}_m$ is column m of matrix $\boldsymbol{V}$. Maximization of the likelihood function $\mathcal{L}(\boldsymbol{U},\boldsymbol{V})$ is equivalent to the minimization of the easier to compute negative log-likelihood

$$\ell(\boldsymbol{U},\boldsymbol{V}) = -\sum_{i=1}^{N}\sum_{m=1}^{M} y_{im}\log\left(\exp(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m)/(\sum_{l=1}^{M}\exp(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_l))\right).$$

The application of word2vec presented by Mikolov et al. (2013b) estimates the set of parameters $(\boldsymbol{U},\boldsymbol{V})$ by means of backpropagated stochastic gradient descent. The word2vec gradient descent method is described in detail by Rong (2014). The model employs a version of stochastic gradient descent, in which the gradient of the loss function for only a single instance at a time is used. For each word i of the N words in the corpus, the probabilities π_{im} are calculated, and the gradient of the instance-specific loss $\ell_i(\boldsymbol{U},\boldsymbol{V})$ relative to the parameter matrices $\boldsymbol{U}$, $\boldsymbol{V}$ is calculated. The parameter matrices are then adjusted with a particular step size $\eta > 0$ in the descent direction relative to $\ell(\boldsymbol{U},\boldsymbol{V})$. Appendix B.1 shows the full algorithm as it is implemented for this research. The benefit of this method is that only a single row $\boldsymbol{x}_i'$ is considered at a time. This makes the model very easily scalable to large datasets, since the matrix $\boldsymbol{X}$ does not need to be kept in memory but can be 'streamed' through the iteration as needed. A drawback of this method is that this large sequence of N parameter updates is unlikely to converge to a global minimum for $(\boldsymbol{U},\boldsymbol{V})$. Since each update only considers one word instance, the parameter updates are blind to their effect on the complete loss function, and may contradict previous parameter updates and even cause net increases in the loss $\ell(\boldsymbol{U},\boldsymbol{V})$.

For both outcomes, the adjustment happens with a particular learning rate $\eta > 0$. A higher learning rate η means that bigger steps are taken in the adjustment of the parameters.

However, this also increases the risk of overshooting the optimal parameter value. When this happens, the parameters may actually be adjusted to a value with a higher loss than the previous iteration. This is a known drawback of gradient descent, which may have trouble navigating valleys in the likelihood space (Wang, 2008). The proposed MM-algorithm adaptation of word2vec has the benefit of guaranteed descent over iterations, and does not require the specification of a suitable η.

In the commonly used stochastic gradient descent method of word2vec optimization, such as in Rong (2014) or Mikolov et al. (2013b), parameter identification is generally not enforced, which means that the estimated parameter set $(\boldsymbol{U},\boldsymbol{V})$ is not the unique global minimum to the cross-entropy loss. For example, if we were to use $(\boldsymbol{UT},\boldsymbol{VT}^{-1})$, with $\boldsymbol{T}$ some invertible $c \times c$ matrix, we would have $\boldsymbol{Z} = \boldsymbol{XUTT}^{-1}\boldsymbol{V} = \boldsymbol{XUV}$. This would result in an identical loss $\ell(\boldsymbol{UT},\boldsymbol{VT}^{-1}) = \ell(\boldsymbol{U},\boldsymbol{V})$. However, since the goal of the model is to observe the similarity between the word embeddings, rather than obtain a unique solution to the classification task, this is generally considered not to be an issue, but may be noted as a drawback of the word2vec framework.

2.3 MM-trained word2vec

We are interested in the use of an alternative method of optimisation of the word2vec loss function presented in the previous section. We therefore specify an algorithm for the optimization of the cross-entropy loss by Minimization by Majorization. Minimization by Majorization (MM) allows for the optimization of functions that have no direct analytical solution, by iteratively optimizing other functions that are similar, and strictly larger than the objective function. For a particular function $f(\boldsymbol{\theta})$ that may be difficult to optimize, and for which our current parameter set is $\boldsymbol{\theta}^0$, the idea is to find another function $g(\boldsymbol{\theta})$ for which $f(\boldsymbol{\theta}^0) = g(\boldsymbol{\theta}^0)$, and for which $f(\boldsymbol{\theta}) \leq g(\boldsymbol{\theta})$. If we can easily minimize $g(\boldsymbol{\theta})$, the resulting optimal solution $\boldsymbol{\theta}^*$ will satisfy $f(\boldsymbol{\theta}^*) \leq f(\boldsymbol{\theta}^0) = g(\boldsymbol{\theta}^0)$, since otherwise $g(\boldsymbol{\theta}^*)$ would not be the true minimum for $g(\boldsymbol{\theta})$. This particular characteristic of the method guarantees descent of the loss function across iterations. However, the method does require the availability of a good $g(\boldsymbol{\theta})$. For optimal performance, the majorizing function must be reasonably close to the objective function, and computationally easy to minimize. We consider the implementation of the MM-algorithm first on the conventional cross-entropy loss word2vec, after which we introduce an alternative model based on quadratic hinge loss.

We adapt the MM-algorithm for the cross-entropy loss function reported in Groenen and Josse (2016). The authors consider the issue of multinomial multiple correspondence analysis, to estimate the multinomial probability of a set of J categorical variables, each with K_j options, conditional on the $J-1$ other categorical variables. Our application differs in two regards. First, only one categorical variable is considered, the choice amongst M different words on each of N word instances. Second, we have a very large choice set M, which requires the restriction of our parameters to the linear combination $\boldsymbol{U}\boldsymbol{V}'$.

To majorize the function $\ell(\boldsymbol{U},\boldsymbol{V})$, we first need to find a proper majorizing function $g_i(\boldsymbol{\theta},\boldsymbol{\theta}^0)$, to majorize the loss $f_i(\boldsymbol{\theta})$ for a particular word instance i. Since the total loss associated with a parameter set is given by the summed loss over all word instances, the complete majorizing function can later be obtained by summing over i. Let $f_i(\boldsymbol{\theta})$ be our objective function, then the majorizing function must satisfy $g_i(\boldsymbol{\theta}^0,\boldsymbol{\theta}^0) = f_i(\boldsymbol{\theta}^0)$, and $g_i(\boldsymbol{\theta},\boldsymbol{\theta}^0) \geq f_i(\boldsymbol{\theta})$. Based on these conditions, the iterative minimization of the majorizing function will guarantee descent across iterations for our objective $f_i(\boldsymbol{\theta})$.

We require a majorizing function for our particular cross-entropy loss objective function. Groenen and Josse (2016) suggest the majorizing function

$$g_i(\boldsymbol{\theta},\boldsymbol{\theta}^0) = f_i(\boldsymbol{\theta}^0) + (\boldsymbol{\theta} - \boldsymbol{\theta}^0)'\nabla f_i(\boldsymbol{\theta}^0) + \frac{1}{4}||\boldsymbol{\theta} - \boldsymbol{\theta}^0||^2, \tag{1}$$

where $\boldsymbol{\theta}^0$ is the parameter set from the previous iteration of the algorithm. In our case we let $\boldsymbol{\theta} = \boldsymbol{Z}$, the $N \times M$ matrix of scores $\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'$ which are used in the exponents to calculate the word probabilities π_{im}. We denote the rows of $\boldsymbol{Z}$ as $\boldsymbol{z}_i' = \boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{V}'$, and the elements as $z_{im} = \boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m$. To find iteratively better fitting weight matrices $\boldsymbol{U}$,$\boldsymbol{V}$, we may majorize with as our parameters the elements of $\boldsymbol{Z}$ and solve for $\boldsymbol{U}$ and $\boldsymbol{V}$ from there on.

We define the instance-specific loss $f_i(\boldsymbol{\theta})$ as:

$$\begin{aligned} f_i(\boldsymbol{\theta}) = \ell_i(\boldsymbol{U},\boldsymbol{V}) &= -\sum_{m=1}^{M} y_{im} \log\left(\exp(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m)/(\sum_{l=1}^{M}\exp(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_l))\right) \\ &= -\sum_{m=1}^{M} y_{im} \log(\left(\exp(z_{im})/(\sum_{l=1}^{M}\exp(z_{il}))\right). \end{aligned}$$

The gradient $\nabla f_i(\boldsymbol{z}_i')$ is given by the first derivative of the instance-specific loss relative to the $1 \times M$ score vector $\boldsymbol{z}_i$. It can be shown that

$$\frac{\partial f_i(\boldsymbol{z}_i')}{\partial \boldsymbol{z}_i} = \frac{\exp(\boldsymbol{z}_i)}{\sum_{l=1}^{M} \exp(z_{il})} - \boldsymbol{y}_i = \boldsymbol{\pi}_i - \boldsymbol{y}_i,$$

which gives the $1 \times M$ vector of derivatives of the instance-specific loss relative to each of the elements of $\boldsymbol{z}_i$. Filling in the equation 1 with functions pertaining to our optimization problem we obtain

$$\begin{aligned} g_i(\boldsymbol{z}_i,\boldsymbol{z}_i^0) &= -\boldsymbol{y}_i' \log(\boldsymbol{\pi}_i^0) + (\boldsymbol{z}_i - \boldsymbol{z}_i^0)'(\boldsymbol{\pi}_i^0 - \boldsymbol{y}_i) + \frac{1}{4}||\boldsymbol{z}_i - \boldsymbol{z}_i^0||^2 \\ &= \sum_{m=1}^{M} \left(-y_{im} \log(\pi_{im}^0) + (z_{im} - z_{im}^0)(\pi_{im}^0 - y_{im}) + \frac{1}{4}(z_{im} - z_{im}^0)^2 \right). \end{aligned}$$

The complete majorizing function, over all word instances i, is then given by

$$g(\boldsymbol{Z},\boldsymbol{Z}^0) = \sum_{i=1}^{N} \sum_{m=1}^{M} \left(-y_{im} \log(\pi_{im}^0) + (z_{im} - z_{im}^0)(\pi_{im}^0 - y_{im}) + \frac{1}{4}(z_{im} - z_{im}^0)^2 \right).$$

To minimize $g(\boldsymbol{Z},\boldsymbol{Z}^0)$, we need to find the point where the gradient of the summed instance-specific losses is 0 relative to each of the elements of $\boldsymbol{U}$ and $\boldsymbol{V}$. We first separate the majorizing functions into all terms containing $\boldsymbol{U}$ and $\boldsymbol{V}$, and terms relating to elements we may regard as constant relative to $(\boldsymbol{U},\boldsymbol{V})$, such as π_{im}^0, z_{im}^0 and y_{im}. We group these into the constant term c_{im}, as they may be disregarded for optimization purposes. After expanding the brackets of the term $(z_{im} - z_{im}^0)^2$, this gives the reduced form expression

$$g(\boldsymbol{Z},\boldsymbol{Z}^0) = \sum_{i=1}^{N} \sum_{m=1}^{M} \left(z_{im}(\pi_{im}^0 - y_{im} - \frac{1}{2} z_{im}^0) + \frac{1}{4} z_{im}^2 + c_{im} \right).$$

To further simplify, we define $r_{im} = \pi_{im}^0 - y_{im} - \frac{1}{2} z_{im}^0$, the elements of the $N \times M$ matrix $\boldsymbol{R} = \boldsymbol{\Pi} - \boldsymbol{Y} - \frac{1}{2}\boldsymbol{Z}^0$. Substituting this into the equation gives the simplified expression

$$g(\boldsymbol{Z},\boldsymbol{Z}^0) = \sum_{i=1}^{N} \sum_{m=1}^{M} \left(z_{im} r_{im} + \frac{1}{4} z_{im}^2 + c_{im} \right).$$

To optimize with respect to the parameter matrices $\boldsymbol{U}$, $\boldsymbol{V}$ we replace z_{im} with its original definition of $\boldsymbol{x}_i' \boldsymbol{U} \boldsymbol{v}_m$ to obtain

$$\begin{aligned} g(\boldsymbol{Z},\boldsymbol{Z}^0) &= \sum_{i=1}^{N} \sum_{m=1}^{M} \left(\boldsymbol{x}_i' \boldsymbol{U} \boldsymbol{v}_m r_{im} + \frac{1}{4}(\boldsymbol{x}_i' \boldsymbol{U} \boldsymbol{v}_m)^2 + c_{im} \right) \\ &= \operatorname{tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'\boldsymbol{R}') + \frac{1}{4} \operatorname{tr}(\boldsymbol{V}\boldsymbol{U}'\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}') + c, \end{aligned}$$

where c contains the sum of all c_{im} over i,m. To find the update equations for our parameter matrices, we first take the derivative of the majorizing function with $\boldsymbol{U} = \boldsymbol{U}^0$ with respect to $\boldsymbol{V}$ (note that per the property of invariance with regards to cyclic permutations of the trace function, we can let $\text{tr}(\boldsymbol{X}\boldsymbol{U}^0\boldsymbol{V}'\boldsymbol{R}') = \text{tr}(\boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0\boldsymbol{V}'))$:

$$\frac{\partial g(\boldsymbol{Z},\boldsymbol{Z}^0)}{\partial \boldsymbol{V}} = \boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0 + \frac{1}{2}\boldsymbol{V}\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0 = \boldsymbol{0},$$

where $\boldsymbol{0}$ is a matrix of zeros that takes the dimension of whatever it is equated to. From this we may derive that

$$\boldsymbol{V} = -2\boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}.$$

We can then update the values of $\boldsymbol{\Pi}$, $\boldsymbol{Z}$, and subsequently $\boldsymbol{R}$ with the new $\boldsymbol{V}$, and then perform a new iteration in which we update $\boldsymbol{U}$. Similar to the update for $\boldsymbol{V}$, we obtain the derivative of the majorizing function with respect to $\boldsymbol{U}$ (again using the cyclic property, where $\text{tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}\boldsymbol{R}') = \text{tr}(\boldsymbol{U}\boldsymbol{V}'\boldsymbol{R}'\boldsymbol{X})$):

$$\frac{\partial g(\boldsymbol{Z},\boldsymbol{Z}^0)}{\partial \boldsymbol{U}} = \boldsymbol{V}^{0\prime}\boldsymbol{R}'\boldsymbol{X} + \frac{1}{2}\boldsymbol{V}^{0\prime}\boldsymbol{V}^0\boldsymbol{U}'\boldsymbol{X}'\boldsymbol{X}.$$

This gives us the update equation

$$\boldsymbol{U} = -2(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{R}\boldsymbol{V}^0(\boldsymbol{V}'^0\boldsymbol{V}^0)^{-1}.$$

Algorithm 1 defines the optimization procedure. A particular issue in optimizing the cross-entropy loss, is that we $-log(\pi_{im})$ is susceptible to encouraging drift in our score matrix $\boldsymbol{Z}$ to extreme values, as to optimise the π_{im} to become as close to 0 or 1 as possible. This can lead to trouble when we try to achieve convergence of the loss function to within a tolerance ε. We consider the option of regularizing the cross-entropy loss function, and present a method for doing so in Appendix A. However, we find it to carry a prohibitive computational burden, and therefore do not conclude it as one of our main methods. Alternatively, we opt to set both a relative convergence tolerance ε, and a time-limit on optimisation, which forces termination of the algorithm when it continues to make small increments in $\ell(\boldsymbol{U},\boldsymbol{V})$ by drifting elements of $\boldsymbol{Z}$ to extreme values.

```
Algorithm 1: MM-trained word2vec
  Result: Optimization of paramater matrices $\boldsymbol{U}$, $\boldsymbol{V}$
  Input: Text corpus $\boldsymbol{d}$, convergence tolerance $\varepsilon$
  Parameters: Context size $k$, embedding size $c$, vocabulary size $M$.
  Let $\boldsymbol{d}$ be the text corpus as a vector of strings $[str1, str2, ..., strN]$.
  Let $\boldsymbol{l}$ be the vocabulary vector, the $M$ most frequent strings in $\boldsymbol{d}$.
  Encode $N \times M$ matrix $\boldsymbol{Y}$ from text corpus, where $y_{im} = 1$ if $d_i = l_m$, 0 otherwise.
  Encode $N \times M$ matrix $\boldsymbol{Q}$ from text corpus, where $q_{im} = 1$ if
   $max(y_{i-k/2,m}, ..., y_{im}, ..., y_{i+k/2,m}) = 1$, 0 otherwise.
  Set $\boldsymbol{X} = \boldsymbol{Y} - \boldsymbol{Q}$.
  Initialize $M \times c$ matrices $\boldsymbol{U}$, $\boldsymbol{U}^0$, $\boldsymbol{V}$, $\boldsymbol{V}^0$ with each of the elements drawn from a
   $N(0,1/c^2)$ distribution.
  Initialize $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'$.
  Initialize $\pi_{im} = \exp(z_{im})/(\sum_{l=1}^{M} \exp(z_{lm}))$ for each $i$, $m$, to construct $\boldsymbol{\Pi}$.
  Initialize $f(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N} \sum_{m=1}^{M} -log(\pi_{im}) y_{im}$, $f(\boldsymbol{U}^0,\boldsymbol{V}^0) = \infty$.
  while $|(f(\boldsymbol{U},\boldsymbol{V}) - f(\boldsymbol{U}^0,\boldsymbol{V}^0))/f(\boldsymbol{U}^0,\boldsymbol{V}^0)| > \varepsilon$ do
    $\boldsymbol{U}^0 = \boldsymbol{U}$.
    $\boldsymbol{V}^0 = \boldsymbol{V}$.
    $\boldsymbol{Z}^0 = \boldsymbol{Z}$.
    $\boldsymbol{\Pi}^0 = \boldsymbol{\Pi}$.
    $\boldsymbol{R} = \boldsymbol{\Pi}^0 - \boldsymbol{Y} - 1/2\boldsymbol{Z}^0$.
    On odd iterations, set $\boldsymbol{V} = -2\boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}$.
    On even iterations, set $\boldsymbol{U} = -2(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{R}\boldsymbol{V}^0(\boldsymbol{V}^{0\prime}\boldsymbol{V}^0)^{-1}$.
    Update $\boldsymbol{Z}$, $\boldsymbol{\Pi}$, $f(\boldsymbol{U},\boldsymbol{V})$.
  end
```

2.4 MM-trained word2vec with quadratic hinge loss

We propose to further amend the word2vec model by using a quadratic hinge loss function. The conventional cross-entropy loss requires the expensive calculation of the sigmoid for deriving π_{im}. This is needed in both the regular stochastic gradient descent evaluation, as well as in the proposed MM-algorithm alternative. The benefit of such a sigmoid function is that it gives exact probability values for each of the possible outcomes M for each instance i. These probabilities give an accessible intuition to the model, and their maximization for the correct word in each instance will definitely converge to a sensible set of parameters $(\boldsymbol{U},\boldsymbol{V})$. However, in the word2vec setting these probabilities may be overzealous. Given the large vocabulary size M, and the even larger corpus size N, estimated specific probabilities are likely not needed for any use other than optimization, and their calculation may take up valuable resources. We consider that a more parsimonious specification that directly targets the scores $\boldsymbol{Z}$ rather than their sigmoid transformation, may achieve equal performance to the cross-entropy specification at a quicker rate. To this end we employ a hinge loss function that only evaluates whether $z_{im} > 1$ for the correct words in an instance, and whether $z_{im} < -1$ for all other words.

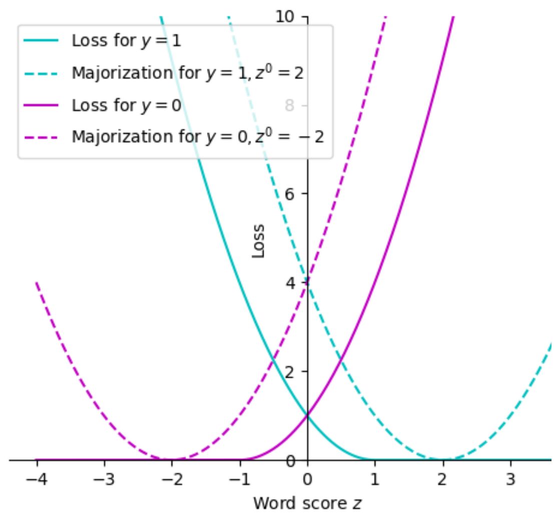
For our specification, we define the following loss function, for a parameter set $(\boldsymbol{U},\boldsymbol{V})$, words i to N in the corpus:

$$\mathcal{H}(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N}\sum_{m=1}^{M}\left(y_{im}\max(1 - z_{im},0)^2 + (1 - y_{im})\max(z_{im} + 1,0)^2\right).$$

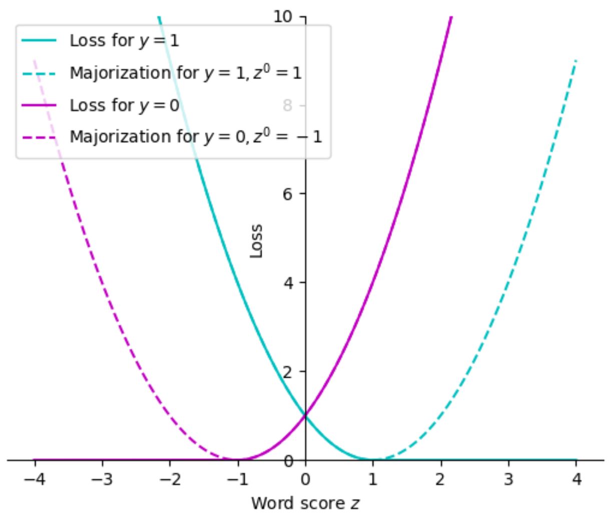
This loss function sets a target score z_{im} of -1 for each incorrect word and a target score of 1 for the correct word. Figure 1 shows this loss function for one word instance in both the case that $y_{im} = 1$ and for $y_{im} = 0$. We see that there is a positive loss associated for a score z_{im} being below the value of 1 when $y_{im} = 1$, or being above the value of -1 when $y_{im} = 0$. A score between -1 and 1 always incurs a positive cost, as to encourage differences in the word-specific parameters $(\boldsymbol{u}_m, \boldsymbol{v}_m)$ across the vocabulary M.

To find a majorizing function, we borrow from Groenen et al. (2008), who give majorizing functions for the absolute, quadratic and huber hinge loss function in a Support Vector Machine application. The authors note that the first component of the hinge loss: $f^{+}(q) = \max(1 - q,0)^2$, can be majorized by $g_1(q) = (1 - q)^2$ when $q^0 \leq 1$ and by $g_2(q) = (q^0 - q)^2$ when $q^0 \geq 1$. This piecewise specification achieves the required conditions for majorization that $g(q^0) = f(q^0)$, as well as achieving the same slope in q^0 and having $g(q) \geq f(q)$ for all q. We see that when q^0 has not yet attained the optimal value in the range greater than 1, the majorizing function is minimized at $q = 1$. When q^0 is already in

Figure 1: Majorization of quadratic hinge loss function by value of y_{im}



(a) Majorization for correct classifications



(b) Majorization for incorrect classifications

the optimal range, there is no further incentive for the majorization to change the value of q, since the majorization is minimized at q^0.

Similarly, for the cost associated with instances for which $y_{im} = 0$, of the form $\max(q + 1{,}0)^2$, Groenen et al. (2008) suggest the majorization $g_1^-(q) = \max(q + 1{,}0)^2$ when $q^0 < -1$, and $g_2^-(q) = \max(q - q^0)^2$. For these instances we can also see that the majorization pushes for a smaller value when q^0 is not yet below -1, attaining the minimum at $q = -1$, and providing no incentive for change when q^0 is already in the optimal range.

The majorizations are given in Figure 1. In Figure 1 (a) we see that for already correct classifications, the minimum is attained at the previous value z^0. In Figure 1 (b) we see that for incorrect classifications the majorization coincides exactly with the loss function itself, reflecting the quadratic loss associated with those instances.

We apply this majorization of our quadratic hinge loss for word embeddings, with respect to the word scores z_{im}. By finding the proper majorization of $\mathcal{H}(\boldsymbol{Z})$, we can then derive an improvement on the parameter set $(\boldsymbol{U},\boldsymbol{V})$. First, we need to know for a given initial parameter set $(\boldsymbol{U}^0,\boldsymbol{V}^0)$, the scores $\boldsymbol{Z}^0$, and whether these scores are in the proper range for their respective values in $\boldsymbol{Y}$. For the first part of the quadratic hinge loss function, we define $s_{im} = 1$ if $z_{im}^0 < 1$, 0 otherwise. For the second part of the loss function, we define $t_{im} = 1$ if $z_{im}^0 > -1$, 0 otherwise. Together, s_{im} and t_{im} denote whether we already have a suitable word score for the combination i,m, (in which case they are both 0), or whether there is a positive loss associated with i,m (in which case either of them are 1). This gives us the majorizing function

$$G(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N}\sum_{m=1}^{M} \left(y_{im}\left(s_{im}(1-z_{im})^2 + (1-s_{im})(z_{im}^0 - z_{im})^2\right) + \right.$$
$$\left.(1-y_{im})\left(t_{im}(1+z_{im})^2 + (1-t_{im})(z_{im} - z_{im}^0)^2\right)\right).$$

We see that the function can be split into the majorization for when $y_{im} = 1$ and when $y_{im} = 0$. Those two parts can then each be split into the majorization for when the classification is already correct ($s_{im} = 0$ or $t_{im} = 0$, respectively) or for when the classification, per the most recent score z_{im}^0 is still incorrect ($s_{im} = 1$ or $t_{im} = 1$, respectively).

Since we are interested in the derivative with respect to z_{im} we want to rewrite the majorization as a quadratic function of z_{im}, expanding the brackets in our majorization and grouping all the coefficients for z_{im}^2 and the linear part z_{im} together:

$$\begin{aligned}G(\boldsymbol{U},\boldsymbol{V}) = &\sum_{i=1}^{N}\sum_{m=1}^{M}(z_{im}^2(y_{im}s_{im} + y_{im}(1-s_{im}) + (1-y_{im})t_{im} + (1-y_{im})(1-t_{im}))\\
&+ z_{im}(-2y_{im}s_{im} - 2y_{im}(1-s_{im})z_{im}^0 + 2(1-y_{im})t_{im} - 2z_{im}^0(1-y_{im})(1-t_{im}))\\
&+ {z_{im}^0}^2(y_{im}(1-s_{im}) + (1-y_{im})(1-t_{im}))\\
&+ y_{im}s_{im} + (1-y_{im})t_{im}).\end{aligned}$$

To aid optimization with respect to z_{im}, we may note that the coefficient for the term z_{im}^2 can be worked out to 1, since always exactly one of the terms evaluates to 1. We can also perform the substitution $a_{im} = y_{im}s_{im} - (1-y_{im})t_{im} + z_{im}^0(y_{im}(1-s_{im}) + (1-y_{im})(1-t_{im}))$ to reduce the linear term to a single coefficient. We can also group the terms that do not contain z_{im}, since these are not relevant for optimization with respect to z_{im}. We then obtain

$$G(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N}\sum_{m=1}^{M}(z_{im}^2 - 2a_{im}z_{im} + c_{im}),$$

where c_{im} contains all the terms not dependent on z_{im}. To facilitate the matrix notation of $G(\boldsymbol{U},\boldsymbol{V})$, we define the $N \times M$ matrix $\boldsymbol{A}$, containing the elements a_{im}. We can then rewrite the majorization to

$$\begin{aligned}G(\boldsymbol{U},\boldsymbol{V}) &= -2\,\mathrm{tr}(\boldsymbol{Z}\boldsymbol{A}') + \mathrm{tr}(\boldsymbol{Z}'\boldsymbol{Z}) + c \\ &= -2\,\mathrm{tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'\boldsymbol{A}') + \mathrm{tr}(\boldsymbol{V}\boldsymbol{U}'\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}') + c.\end{aligned}$$

Due to the quadratic nature of both this majorizing function, and the majorization used for the cross-entropy loss, we may note the parallel in the two majorizations. Both have a linear component and a quadratic component, with the primary difference between the two being in the linear term, $\mathrm{tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'\boldsymbol{R}')$ for the cross-entropy loss, and $\mathrm{tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'\boldsymbol{A}')$ for the hinge loss function. The potential benefit to hinge loss majorization may be in the linear term being less expensive to calculate, since it no longer requires the calculation of $\boldsymbol{\Pi}$. Similar to the optimization of the regular word2vec loss, we first optimise $G(\boldsymbol{U}^0,\boldsymbol{V})$ relative to $\boldsymbol{V}$. To this end, we derive

$$\frac{\partial G(\boldsymbol{U}^0,\boldsymbol{V})}{\partial \boldsymbol{V}} = 2\boldsymbol{V}\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0 - 2\boldsymbol{A}'\boldsymbol{X}\boldsymbol{U}^0 = \boldsymbol{0},$$

from which we may obtain the update equation

$$\boldsymbol{V} = \boldsymbol{A}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}.$$

We may then re-estimate $\boldsymbol{Z}$ to obtain an updated $\boldsymbol{A}$ before updating $\boldsymbol{U}$. We then take the derivative of $G(\boldsymbol{U},\boldsymbol{V}^0)$ with respect to $\boldsymbol{U}$:

$$\frac{\partial G(\boldsymbol{U},\boldsymbol{V}^0)}{\partial \boldsymbol{U}} = 2\boldsymbol{X}\boldsymbol{X}'\boldsymbol{U}\boldsymbol{V}^{0\prime}\boldsymbol{V}^0 - 2\boldsymbol{X}'\boldsymbol{A}\boldsymbol{V}^0,$$

which gives us the update equation

$$\boldsymbol{U} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{A}\boldsymbol{V}^0(\boldsymbol{V}^{0\prime}\boldsymbol{V}^0)^{-1}.$$

The resulting Algorithm 2 for finding hinge-loss minimized word embeddings is similar to the former, save for the differences in the weight updates and loss calculations.

Algorithm 2: MM-trained word2vec with quadratic hinge loss.

```
**Result:** Optimization of paramater matrices $\boldsymbol{U}$, $\boldsymbol{V}$.
**Input:** Text corpus $\boldsymbol{d}$, convergence tolerance $\varepsilon$.
**Parameters:** Context size $k$, embedding size $c$, vocabulary size $M$.
Let $\boldsymbol{d}$ be the text corpus as a vector of strings $[str1, str2, ..., strN]$.
Let $\boldsymbol{l}$ be the vocabulary vector, the $M$ most frequent strings in $\boldsymbol{d}$.
Encode $N \times M$ matrix $\boldsymbol{Y}$ from text corpus, where $y_{im} = 1$ if $d_i = l_m$, 0 otherwise.
Encode $N \times M$ matrix $\boldsymbol{Q}$ from text corpus, where $q_{im} = 1$ if
  $max(y_{i-k/2,m},...,y_{im},...,y_{i+k/2,m}) = 1$, 0 otherwise.
Set $\boldsymbol{X} = \boldsymbol{Y} - \boldsymbol{Q}$.
Initialize $M \times c$ matrices $\boldsymbol{U}$, $\boldsymbol{U}^0$, $\boldsymbol{V}$, $\boldsymbol{V}^0$ with each of the elements drawn from a
  $N(0,1/c^2)$ distribution.
Initialize $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'$.
Initialize $f(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N}\sum_{m=1}^{M}(y_{im}\max(1 - z_{im},0)^2 + (1 - y_{im})\max(z_{im} + 1,0)^2)$,
  $f(\boldsymbol{U}^0,\boldsymbol{V}^0) = \infty$.
**while** $|(f(\boldsymbol{U},\boldsymbol{V}) - f(\boldsymbol{U}^0,\boldsymbol{V}^0))/f(\boldsymbol{U}^0,\boldsymbol{V}^0)| > \varepsilon$ **do**
    $\boldsymbol{U}^0 = \boldsymbol{U}$.
    $\boldsymbol{V}^0 = \boldsymbol{V}$.
    $\boldsymbol{Z}^0 = \boldsymbol{Z}$.
    Construct $\boldsymbol{S}$, where $s_{im} = 1$ if $z^0_{im} < 1$, 0 otherwise.
    Construct $\boldsymbol{T}$, where $t_{im} = 1$ if $z^0_{im} > -1$, 0 otherwise.
    Construct $\boldsymbol{A}$, where
      $a_{im} = y_{im}s_{im} - (1 - y_{im})t_{im} + z^0_{im}(y_{im}(1 - s_{im}) + (1 - y_{im})(1 - t_{im}))$.
    On odd iterations, set $\boldsymbol{V} = \boldsymbol{A}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}$.
    On even iterations, set $\boldsymbol{U} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{A}\boldsymbol{V}^0(\boldsymbol{V}^{0\prime}\boldsymbol{V}^0)^{-1}$.
    Update $\boldsymbol{Z}$, $f(\boldsymbol{U},\boldsymbol{V})$.
**end**
```

2.5 Sampling

Our final research sub-question concerns the scalability of the algorithms proposed thus far by means of negative sampling. To this end, we present three distinct methods of sampling, and derive the MM-optimizations for the word2vec model with a sampled, sparse $\boldsymbol{Z}$. So far, the methods for training the word embeddings, for both the traditional cross-entropy- and hinge-loss functions, has relied on the computation of the full $N \times M$ score matrix $\boldsymbol{Z} = \boldsymbol{XUV}'$. This matrix is dense, in that every element is expected to be non-zero, and for the cross-entropy loss it also needs to be transformed with the sigmoid function to calculate $\boldsymbol{\Pi}$. These factors together make the calculation of scores or probabilities for each algorithm computationally very expensive. The efforts for calculating all M scores for each word instance i may also be needlessly thorough, as it evaluates the whole choice set given by the vocabulary at each instance. A possible gain in efficiency may be reached by significantly reducing the density of $\boldsymbol{XUV}'$ by evaluating the score of only a sample of the vocabulary. This changes the training task from classifying the correct word from the whole vocabulary, to trying to find the correct word relative to a small selection of competing words. Mikolov et al. (2013b) also suggest negative sampling to speed up their stochastic gradient descent implementation of word2vec. The authors suggest negative sampling of the incorrect words in m, for each instance i, and find that the optimal sampling method uses the most frequent words in the vocabulary (e.g. 'the', 'and', 'of') more often as negative samples.

To get meaningful word embeddings, this sample should contain the target word for which $y_{im} = 1$, and a small selection of other words in the vocabulary. We define the $N \times M$ matrix $\boldsymbol{P}$ as the sampling matrix, which has p non-zero elements in each row i to denote which of the M words in the vocabulary are included in evaluating the instance-specific loss. Here, $p_{im} = 1$ to denote the word m being sampled for instance i, and 0 otherwise, where $p_{im} = 1$ always if $y_{im} = 1$. To evaluate the effects of sampling we evaluate three methods, specifically:

- Uniform random sampling: we set $p_{im} = 1$ if $y_{im} = 1$, and otherwise the value of p_{im} is determined by a uniform random distribution, with $P[p_{im} = 1] = (p - 1)/M$, 0 otherwise. May be performed repeatedly with every iteration.

- Frequency-based weighted sampling: Mikolov et al. (2013b) suggest a weighted sampling method, making use of the unigram distribution (the sample frequency of the word in the corpus) raised to the power 3/4 as the negative sampling distribution Specifically, the probability of being included as the negative sample is given by $P[p_{im} = 1] = (\sum_{i=1}^{N} y_{im})^{3/4})/(\sum_{l=1}^{M}(\sum_{i=1}^{N} y_{il})^{3/4}))$. Using popular words more frequently may balance out their frequent occurrence as positive samples.

- Context sampling: we let $p = k + 1$ and $\boldsymbol{P} = \boldsymbol{Q}$, in which the context words are evaluated against the target word.

The increase in the sparsity in the score matrix $\boldsymbol{Z}$ will allow quicker calculation of the loss function, as well as quicker parameter updates. We implement uniform random sampling, weighted frequency-based random sampling and context sampling, for both the MM-hinge and the MM-word2vec algorithms and evaluate the resulting performance on word analogy tasks, and in- and out-of-sample classification accuracy.

2.5.1 Sampling MM-trained word2vec

The update equations for our MM algorithms change slightly when we include negative sampling. In this section, we derive the parameter update equations for the MM-optimised word2vec model when we include negative sampling. The application of a sampling matrix p_{im} to our loss function means that we evaluate the loss only for the combinations i,m for which $p_{im} = 1$. This means that we only need to evaluate the score z_{im} and probabilities π_{im} for those instances for which $p_{im} = 1$. If we define

$$z^*_{im} = p_{im}\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m = p_{im}z_{im},$$

we obtain the instance-specific loss function

$$f_i(\boldsymbol{U},\boldsymbol{V}) = -\sum_{m=1}^{M} p_{im}y_{im}\log(\left(\exp(z^*_{im})/(\sum_{l=1}^{M} p_{im}\exp(z^*_{il}))\right) = -\sum_{m=1}^{M} y_{im}\log(\pi^*_{im}),$$

where we used the fact that when $y_{im} = 1$, $p_{im} = 1$, such that $y_{im}p_{im} = y_{im}$. Note that the probability π^*_{im} now reports the estimated probability of the word m being the correct word relative to all the other words for which $p_{im} = 1$, rather than as evaluated against the entire vocabulary. This has the benefit of being easier to compute, but with the drawback that the estimated probability may not accurately reflect the position of the embedding relative to the rest of the vocabulary. There may be another word that has a higher score z_{im} than our target word, but this is not accounted for in the loss function, or the majorization.

The majorization of this function, using the same steps as used in the previous majorization of the cross-entropy loss, is given by

$$g_i(\boldsymbol{z}_i,\boldsymbol{z}_i^0) = \sum_{m=1}^{M} \left(-y_{im}\log(\pi_{im}^{*0}) + (z_{im}^* - z_{im}^{*0})(\pi_{im}^{*0} - y_{im}) + \frac{1}{4}(z_{im}^* - z_{im}^{*0})^2 \right).$$

However, the quadratic part of the majorizing function now contains the term $1/4(z_{im}^* - {z_{im}^*}^0)^2$. The dependence on the sampling matrix $\boldsymbol{P}$ makes this term difficult to optimize with respect to our parameter matrices $\boldsymbol{U}$,$\boldsymbol{V}$. We may however perform a further majorization by substituting z_{im} for z_{im}^* in the quadratic part only, to obtain:

$$h_i(\boldsymbol{z}_i,\boldsymbol{z}_i^0) = \sum_{m=1}^{M} \left(-y_{im}\log(\pi_{im}^{*0}) + (z_{im}^* - z_{im}^{*0})(\pi_{im}^{*0} - y_{im}) + \frac{1}{4}(z_{im} - z_{im}^{0})^2 \right).$$

Since the quadratic term is centered around z_{im}^0, this majorization still satisfies $h_i(\boldsymbol{z}_i^0,\boldsymbol{z}_i^0) = f_i(\boldsymbol{z}_i^0)$, as well as $f_i(\boldsymbol{z}_i) \leq h_i(\boldsymbol{z}_i,\boldsymbol{z}_i^0)$ and $\partial f_i/\partial \boldsymbol{z}_i(\boldsymbol{z}_i^0) = \partial h_i/\partial \boldsymbol{z}_i(\boldsymbol{z}_i^0,\boldsymbol{z}_i^0)$. This majorization has the benefit of p_{im} only occurring in the linear term, where it can be easily included in the partial derivative of the majorizing function.

Summing this instance-specific majorization $h_i(\boldsymbol{z}_i,\boldsymbol{z}_i^0)$ over all instances i, and grouping by the terms independent- and dependent of z_{im}^* gives

$$g_i(\boldsymbol{z}^*,\boldsymbol{z}^{*0}) = \sum_{i=1}^{N}\sum_{m=1}^{M} \left(z_{im}^*(\pi_{im}^{*0} - y_{im}) - \frac{1}{2}z_{im}z_{im}^0 + \frac{1}{4}z_{im}^2 + c_{im} \right).$$

We may then substitute $r_{im}^* = p_{im}(\pi_{im}^{*0} - y_{im}) - \frac{1}{2}z_{im}^0$, and rewrite to matrix notation to obtain

$$g(\boldsymbol{Z}^*,\boldsymbol{Z}^{*0}) = \mathrm{tr}((\boldsymbol{XUV}')\boldsymbol{R}^{*\prime}) + \frac{1}{4}\mathrm{tr}((\boldsymbol{XUV}')'(\boldsymbol{XUV}')) + \sum_{i=1}^{N}\sum_{m=1}^{M} c_{im}.$$

Note that this is the same majorization we obtained for the unsampled MM-optimized cross-entropy loss function, save for the filtered equivalent $\boldsymbol{R}^*$. By the same steps outlined in the section 2.3, we can obtain the parameter update equations

$$\boldsymbol{V} = -2\boldsymbol{R}^{*\prime}\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1},$$

and

$$\boldsymbol{U} = -2(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{R}^*\boldsymbol{V}^0(\boldsymbol{V}^{0\prime}\boldsymbol{V}^0)^{-1}.$$

If we write out $\boldsymbol{R}^* = (\boldsymbol{\Pi}^* - \boldsymbol{Y}) - \frac{1}{2}\boldsymbol{Z}^0$, we obtain as our update equation for $\boldsymbol{V}$

$$\begin{aligned}\boldsymbol{V} &= \boldsymbol{Z}^{0\prime}\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1} - 2(\boldsymbol{\Pi}^* - \boldsymbol{Y})'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}\\ &= \boldsymbol{V}^0\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1} - 2(\boldsymbol{\Pi}^* - \boldsymbol{Y})'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}\\ &= \boldsymbol{V}^0 - 2(\boldsymbol{\Pi}^* - \boldsymbol{Y})'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}.\end{aligned}$$

Similarly for $\boldsymbol{U}$, writing out $\boldsymbol{R}^*$ and substituting $\boldsymbol{X}\boldsymbol{U}^0\boldsymbol{V}^{0\prime}$ for $\boldsymbol{Z}^0$ reduces the update equation for $\boldsymbol{U}$ to

$$\boldsymbol{U} = \boldsymbol{U}^0 - 2(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'(\boldsymbol{\Pi}^* - \boldsymbol{Y})\boldsymbol{V}(\boldsymbol{V}'\boldsymbol{V})^{-1}.$$

2.5.2 Sampling MM-trained word2vec with quadratic hinge loss

For our hinge loss function we may take the same steps to obtain the sampled parameter update equations. First, we note the instance-specific loss

$$f_i(\boldsymbol{U},\boldsymbol{V}) = \sum_{m=1}^{M}\left(y_{im}\max(1 - z_{im},0)^2 + (1 - y_{im})p_{im}\max(z_{im} + 1,0)^2\right).$$

We again invoke the definition of the matrices $\boldsymbol{S}$, $\boldsymbol{T}$, where $s_{im} = 1$ if $z^0_{im} < 1$ and $t_{im} = 1$ if $z_{im} > -1$, and use this to define the majorization

$$\begin{aligned}G(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N}\sum_{m=1}^{M}&\left(y_{im}\left(s_{im}(1 - z_{im})^2 + (1 - s_{im})(z^0_{im} - z_{im})^2\right) + \right.\\ &\left.(p_{im} - y_{im})\left(t_{im}(1 + z_{im})^2 + (1 - t_{im})(z^0_{im} - z_{im})^2\right)\right).\end{aligned}$$

We now again substitute the linear component to obtain a reduced form expression of the form $G(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N}\sum_{m=1}^{M} z^2_{im} - 2a_{im}z_{im} + c_{im}$, but work this out a bit further by setting

$$\begin{aligned}a^*_{im} &= y_{im}s_{im} - (p_{im} - y_{im})t_{im} + z^0_{im}(y_{im}(1 - s_{im}) + (p_{im} - y_{im})(1 - t_{im}))\\ &= z^0_{im} - y_{im}s_{im}(1 - z^0_{im}) + (p_{im} - y_{im})t_{im}(1 + z^0_{im})\\ &= z^0_{im} + d_{im},\end{aligned}$$

where $d_{im} = -y_{im}s_{im}(1 - z^0_{im}) + (p_{im} - y_{im})t_{im}(1 + z^0_{im})$. We may then write this out in matrix notation by defining $\boldsymbol{D}$ as the $N \times M$ matrix containing the elements d_{im}:

$$G(\boldsymbol{U},\boldsymbol{V}) = -2\,\mathrm{tr}((\boldsymbol{Z}^0 + \boldsymbol{D})'\boldsymbol{XUV}') + \mathrm{tr}(\boldsymbol{VUX}'\boldsymbol{XUV}') + c,$$

where c contains all the terms constant with respect to $\boldsymbol{Z}$. From this point on, we can use the same steps as in section 2.4 to obtain the update equations

$$\begin{aligned}\boldsymbol{V} &= (\boldsymbol{Z}^0 + \boldsymbol{D})'\boldsymbol{XU}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{XU}^0)^{-1}\\ &= \boldsymbol{V}^0 + \boldsymbol{D}'\boldsymbol{XU}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{XU}^0)^{-1},\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{U} &= (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'(\boldsymbol{Z}^0 + \boldsymbol{D})\boldsymbol{V}^0(\boldsymbol{V}^{0\prime}\boldsymbol{V}^0)^{-1}\\ &= \boldsymbol{U}^0 + (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'(\boldsymbol{D})\boldsymbol{V}^0(\boldsymbol{V}^{0\prime}\boldsymbol{V}^0)^{-1}.\end{aligned}$$

Due to the quadratic nature of the loss function, we may observe a similarity in the update equations to the solution of ordinary least squares regression. Particularly, for the update equation to $\boldsymbol{V}$, we see that the update performs multiple linear regression of the elements of $\boldsymbol{D}$ on the $N \times c$ matrix $\boldsymbol{XU}^0$, finding which parameter update for $\boldsymbol{V}$ best minimizes the square of the errors in $(\boldsymbol{D} - \boldsymbol{XUV}')$, where each element of $\boldsymbol{D}$ reports the ideal change in z_{im}. Note that per the sparsity of $\boldsymbol{D}$, the target value for the update to z_{im} is 0 for any value for which $p_{im} = 0$. Otherwise, the target $\boldsymbol{D}$ is $(1 - z^0_{im})$ if $y_{im} = 1$ and $z^0_{im} < 1$ (i.e. the score is not yet high enough to warrant correct classification of i,m). The ideal update to $\boldsymbol{U}$,$\boldsymbol{V}$ increases z_{im} by d_{im}, which for this case would be $(1 - z^0_{im})$. For any instance for which $y_{im} = 0$ and $z^0_{im} > -1$ (i.e. the score is not yet low enough to correctly classify i,m), the target update to z_{im} is $(-1 - z^0_{im})$. Ideally, the sparsity of $\boldsymbol{D}$ increases with every iteration, since its sparsity is directly proportional to the number of correctly classified values in the i,m domain. Any value which has $p_{im} = 0$ or any value which is correctly classified has $d_{im} = 0$. This allows the updates to focus on the instances for which we do not yet have a correct score, rather than further increasing the value of already correct values. This may be a potential upside of the quadratic hinge loss over the cross-entropy loss, which maintains the $(\boldsymbol{\Pi} - \boldsymbol{Y})$ loss matrix in the update equation throughout optimization, which has a constant density proportional to the density of $\boldsymbol{P}$. It redirects optimization to the problem cases, and also allows for quicker iterations, since we only have to use the non-zero elements in $\boldsymbol{D}$ to perform our parameter updates.

2.6 Model Accuracy

To compare the models presented in the previous sections, we use as a performance metric their classification accuracy. That is, the proportion of instances i in the text corpus for which the model suggests the correct word $m*$ for which $y_{im^*} = 1$. We also use the top-10 classification accuracy, which reports the proportion of instances i for which the correct word appears among the 10 first suggestions offered by the model.

We can first define the accuracy metric as

$$\mathcal{A}(\boldsymbol{U},\boldsymbol{V}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{M}(y_{im}I[m = \arg\max_{l}\, z_{il}]),$$

where $I[\cdot]$ is an indicator function that equals to 1 if the condition in brackets is true, 0 otherwise. In this case, it reports whether the word m has the maximum score z_{im} for instance i. Similarly, we can define the top-10 accuracy as

$$\mathcal{A}_{10}(\boldsymbol{U},\boldsymbol{V}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{M}(y_{im}I[m \in L_i]),$$

where L_i is the set of 10 words with the largest z_{im} in $\boldsymbol{z}_i$.

We calculate in-sample classification accuracy, but also out-of-sample classification accuracy (i.e. word instances i which do not appear in our training data). Unless stated otherwise, we use an out-of-sample data set that is of the same size N as our training data, with word instances i being randomly allocated to either the training set or the out-of-sample test set.

3 Data

The primary dataset used to train the model will be 'text8'. This dataset contains the first 10^8 bytes of text on the English wikipedia as of March 2006. Interpunction and HTML tags are removed to produce a raw text corpus of 17 million words, containing 253 thousand unique words. This includes both conventional English words, as well as various types of names that may appear more sparsely in the corpus. For the estimation of word embeddings, the vocabulary size is commonly truncated to a particular value (e.g. 10,000), keeping only the most frequent words, so as not to waste computation efforts on the sparse occurrences of particular names or misspellings of regular words, for which robust word embeddings are unlikely to be calculable.

The data is freely available online [1]. The text8 document contains a space-separated string of 17,005,208 words. A total of 253,855 unique strings are contained in the document. Across all unique strings, the median number of occurrences in the document is 2, with a mean number of occurrences of 67. 47,314 words have an occurrence of 10 or more, 11,815 occur 100 times or more. Table 1 shows the top 10 most frequent words, as well as a few others. Overall, we find that numbers, prepositions and articles take primacy in the text corpus.

Table 1: Most frequent words in the 'text8' corpus

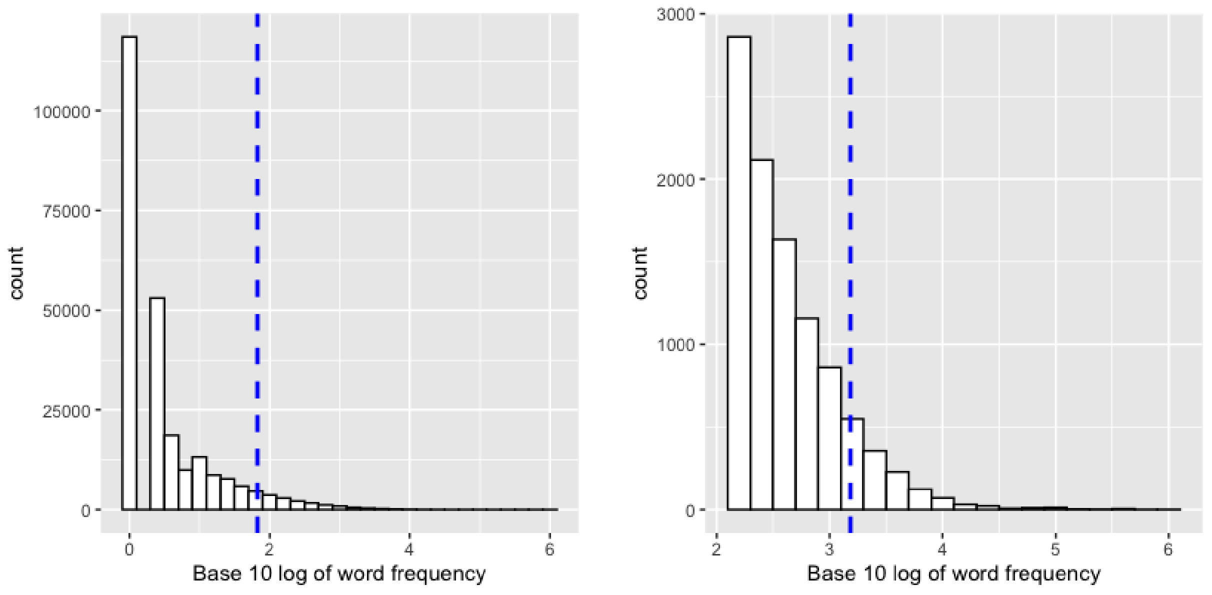
Rank	Word	Count
1	the	1,061,396
2	of	593,677
3	and	416,629
4	one	411,764
5	in	372,201
6	a	325,873
7	to	316,376
8	zero	264,975
9	nine	250,430
10	two	192,644
100	where	12,347
1000	takes	1,783
10000	beria	126

Figure 2a shows the distribution of the logarithm of the word frequencies. We see

[1]http://mattmahoney.net/dc/textdata.html

a clear left skew of the data, even after logarithmic transformation. This indicates that a vast majority of the text consists of infrequently used words. These may be personal names, technical subject-specific terms or misspellings of more common words. Restriction of the vocabulary M to 10,000 words results in the word frequency distribution shown in Figure 2b. This subsample of the vocabulary contains all words with at least 126 occurrences. Together, this subsample of the total vocabulary in the dataset accounts for 15.3 million words in 'text8', or 89.8% of the whole dataset. In testing the models presented in the previous section, we consider various subsets of the text8 dataset. For the unsampled model estimations, even a subset of 100,000 words leads to a significant pressure on memory, since we keep the $N \times M$ matrix Π in memory. Subsampling drastically reduces this memory pressure, allowing for the use of larger subsets of the dataset.

Figure 2: Distribution of word frequencies in the full 'text8' dataset (a), and for the 10,000 most frequent words (b). The blue line shows the base 10 logarithm of the mean word frequency.



(a) Full data

(b) 10^4 most frequent words

3.1 Word Analogy Task

The quality of the obtained word embeddings can be assessed by testing their accuracy in predicting words from the surrounding context, the task which the word embeddings are trained to perform. However, we would ideally also like to see some of the word embedding behaviours shown in the literature, in which relations between words are reflected in the similarities in their word embeddings, such that we can solve tasks like ('girl' stands to 'boy' as 'woman' stands to 'X') by solving $\text{vec}('girl') - \text{vec}('boy') + \text{vec}('woman') = \text{vec}(X)$, with the closest match for $\text{vec}(X)$ in our embedding set. To test this type of analogical reasoning in our obtained embeddings, we use the task set set forth by Mikolov et al. (2013a). The set is freely available online[2] and contains 19,544 questions. Each task has 4 words, with the first three words setting up the analogy: w_1 stands to w_2 as w_3 stands to x, and with the fourth word w_4 giving the solution to x. In line with the literature (e.g. Mikolov et al. (2013b), Pennington et al. (2014)), we solve for x by finding the closest embedding m in our vocabulary M by measure of the cosine similarity[3] of $\text{vec}(\boldsymbol{u}_m, \boldsymbol{v}_m)$ to $\text{vec}(w_1) - \text{vec}(w_2) + \text{vec}(w_3)$. The idea is to find the word which most closely matches w_1, but without the properties defining w_2, and with the properties that define w_3 added to it. For example, for 'father' stands to X as 'man' stands to 'woman', we subtract the 'man' properties from 'father', and add the 'woman' properties, such that the best match to the resulting vector representation is, hopefully, 'mother'. The task set has several distinct categories:

1. Capital-Country pairs: e.g. Paris - France, Brussels - Belgium.
2. Gender pairs: e.g. Man - Woman, Son - Daughter
3. Grammatical pairs: Calm - Calmly, Quiet - Quietly
4. Opposite pairs: Certain - Uncertain, Decided - Undecided

[2]http://download.tensorflow.org/data/questions-words.txt

[3]The cosine similarity of two vectors $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ of length L is given by

$$sim(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \frac{\sum_{l=1}^{L} \theta_{1l}\theta_{2l}}{(\sqrt{\sum_{l=1}^{L} \theta_{1l}^2})(\sqrt{\sum_{l=1}^{L} \theta_{2l}^2})}$$

5. Comparative pairs: Big - Bigger, Bad - Worse
6. Present participle pairs: Read - Reading, Run - Running
7. Plural pairs: Child - Children, Road - Roads

For a given embedding set and vocabulary, we evaluate what percentage of the tasks can be executed exactly, and what percentage of tasks have the correct answer within the closest 10 matches. The latter metric allows us to distinguish whether a model shows promise, even when the allocated text corpus size or training time has not resulted in high performance in exact task execution. We note that the entire task set may not be completed for every embedding set, since the vocabulary taken from the text corpus must overlap with the words in the task set. Especially for the more obscure capital-country pairs, the vocabulary may not cover the whole task. In this case, we report the number of tasks covered by the vocabulary, and evaluate performance as the percentage of tasks for which the correct word is found.

4 Results

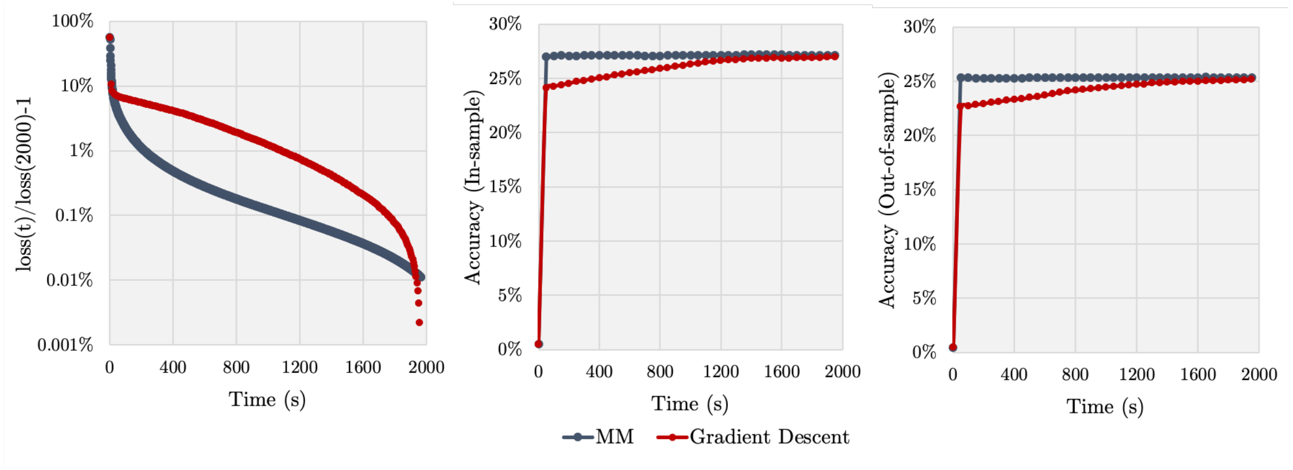
4.1 Minimization by Majorization in word2vec

We first test the performance of the MM-optimization of word2vec, relative to the stochastic gradient descent method. We look at computation time per iteration, as well as the progress over time in minimizing the cross-entropy loss. We also observe how the time per iteration scales with respect to the hyperparameters N, M, and c. Since the mm-optimized algorithm primarily uses matrix operations to update the parameters, it is implemented completely in numpy, which efficiently handles matrix multiplications. The gradient descent word2vec algorithm loops over the complete corpus in order to update the parameters. It is therefore implemented in cython to achieve a performance increment relative to conventional python, which would be particularly slow in performing such a looped operation.

4.1.1 Time to convergence with $N = 100{,}000$, $M = 200$

For the first 100,000 words in the text8 corpus, we run word2vec both through regular stochastic gradient descent, and with minimization by majorization. We use a gradient descent step size $\eta = 0.025$ for the stochastic gradient descent algorithm, since it is the default value for the popular ‘gensim‘ implementation. The step size is shrunk by 1% every iteration, to a minimum value of 0.001. Both algorithms are run for a period of 2,000 seconds, with a word embedding size of $c = 25$. Figure 3 shows the progression in the loss function across iterations. Convergence to within the relative convergence criterion $\varepsilon = 10^{-5}$ is reached after 506 seconds for the MM optimization algorithm, and after 1996 seconds for the gradient descent optimized algorithm. We find that the stochastic gradient descent

Figure 3: Optimization progress towards minimum loss and classification accuracy for MM and stochastic gradient descent algorithms, for $N = 100{,}000$, $M = 200$, $k = 10$, $c = 25$.



algorithm achieves a quicker initial minimization of the loss function within the first iteration. However, the MM-optimization quickly surpasses the gradient descent method. This may be symptomatic of the stochastic gradient descent algorithm performing updates for every word instance, performing N updates to the parameter set even in just one iteration. This may be beneficial for the first few iterations, but inefficient for long-term optimization, in which we find the MM algorithm performing better. By considering the corpus as a whole with each update, the MM algorithm converges to a set of word embeddings with slightly better classification accuracy of 27.1%, relative to 26.9% for the gradient descent implementation.

Out-of-sample, accuracy at convergence is 25.3% and 25.2% for MM and stochastic gradient descent, respectively. We see no drop in out-of-sample performance across iterations, which would be a sign of harmful overfitting.

Average time per iteration was .35 seconds for the MM algorithm, and 6.17 seconds for the stochastic gradient descent algorithm. This also reflects the elaborate computational requirement of calculating word probabilities and updating the parameter set upon each word instance, rather than aggregating the parameter update for the complete corpus. We find that optimization progress by measure of classification accuracy mirrors the progress by measure of the loss function.

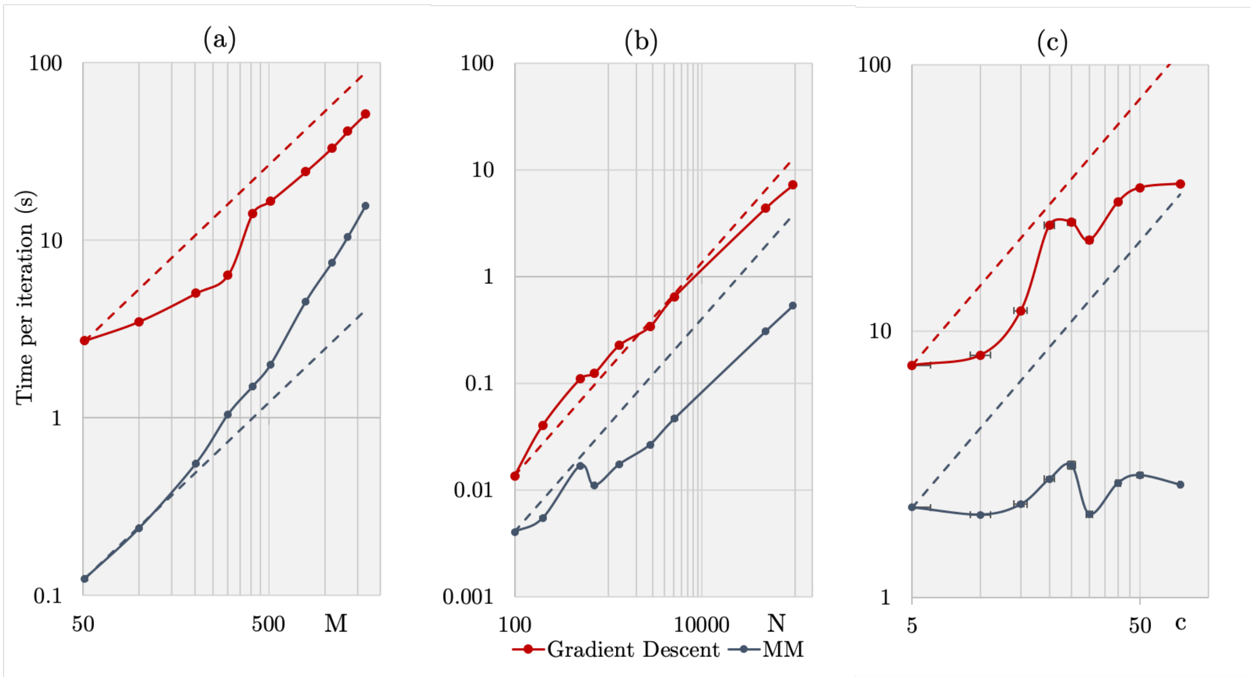
4.1.2 Scaling relative to M, N and c

We repeat the previous experiment, but vary the hyperparameters to observe how each method of optimization scales with respect to the corpus size N, the vocabulary size M, and the dimension c of the parameter matrices $\boldsymbol{U}, \boldsymbol{V}$. We perform 100 iterations of each algorithm for increasing values of the hyperparameters and note the mean time taken for parameter updates per iteration.

Figure 4(a) shows the performance of both optimisation methods relative to the vocabulary size M. Here we observe a potential bottleneck for the MM optimisation of word2vec. Time taken per iteration appears to increase more than linearly relative to M. For the gradient descent implementation, performance appears to be linear in M. We also observe this in the difference in computation time per iteration between the two methods. For $M = 50$, the stochastic gradient descent algorithm took 21.9 times longer, on average, per iteration. For $M = 1500$, the algorithm only took 3.3 times longer per iteration. From this we conclude that traditional gradient descent scales better relative to the vocabulary size M. A quadratic complexity in M will make the MM-optimization unsuitable for implementations with M in the thousands.

Figure 4(b) shows the performance of the algorithms for increasing values of N. We see that for both algorithms, computation time is roughly linear in N, making both applications suitably scaleable over large corpora, provided that no memory issues arise. We see that for each value of N, our implementation of the stochastic gradient descent word2vec

Figure 4: Computation time per iteration by optimization method, relative to vocabulary size M (a), corpus size N (b) and embedding size c (c), with base values N = 100,000, M = 100, $c = 25$, linear extrapolations of computation time given in dotted lines.



has a longer computation time per iteration than the MM algorithm.

Figure 4(c) reports performance relative to the embedding size c. We find that both algorithms scale well with c, showing increases in computation time per iteration that trend less than linearly with c. The required matrix multiplications are efficient enough in c that an increase in c carries very little penalty for these values of M and N. For purposes of optimization, it is therefore most important which vocabulary size M is selected, rather than the size of the embedding c, which can be increased without much additional computation cost. For the sake of obtaining interpretable word embeddings, it may be advisable to increase c in tandem with M, to allow the larger vocabulary more room to express the differences between the constituent words.

4.2 MM-trained embeddings with quadratic hinge loss

In this section we study the second research question: ”Does the use of a quadratic hinge loss function over the traditional cross-entropy loss accelerate parameter learning, and are

Table 2: Classification accuracy for gradient-descent word2vec, as well as MM-optimized cross-entropy (CE) and Quadratic Hinge (QH) loss, in- and out-of-sample, $N = 200{,}000$, $M = 1000$, $c = 50$

Algorithm	Time Until Convergence (m)	Accuracy	Top 10 Accuracy	Out-of-sample Accuracy	Out-of-sample Top 10 Accuracy
Word2vec	-	21.8%	54.2%	18.7%	45.4%
MM CE	-	25.1%	55.9%	20.9%	48.9%
MM QH	79	24.1%	55.5%	21.4%	48.0%

the resulting embeddings of equal quality?". We train the model on a subsample of the text8 dataset of $N = 200{,}000$ words. We set a vocabulary size of $M = 1000$, and $c = 50$ as the size of the parameter matrices $\boldsymbol{U}$,$\boldsymbol{V}$. We train the quadratic hinge model until relative convergence of $\varepsilon = 10^{-5}$. Table 2 reports the results. Convergence is reached after 79 minutes. In-sample classification accuracy at convergence is 24.1%. The correct word is in the 10 highest values of z_{im} for 55.5%. Out-of-sample, classification accuracy is 21.4%, and 48.0% of observations have the correct word within the 10 best estimates offered by the model. The word2vec algorithm fails to converge within $\varepsilon = 10^{-5}$ in two hours of optimisation, we therefore compare performance after an equal optimization time of 79 minutes.

We find that the quadratic hinge model has slightly lower in-sample classification accuracy than the MM-optimized cross-entropy loss model at 25.1%. However, its out-of-sample performance is slightly better (21.4% for the quadratic hinge loss, 20.9% for the cross-entropy loss), suggesting a lower propensity for overfitting the sample. Both models outperform our implementation of the conventional gradient descent trained word2vec model. This suggests that MM-optimization may provide better classification performance in equal training time. However, we have also found that the MM implementation does not scale well with M. This suggests that the higher observed level of accuracy offered by MM may not extend to larger vocabularies.

We find no evidence in this implementation that the quadratic hinge loss offers a

Table 3: Most similar word embedding to a selection of words and analogical tasks, for traditional word2vec and MM-models with cross-entropy (CE), and Quadratic Hinge (QH) loss. Estimated with $N = 200{,}000$, $M = 1{,}000$, $c = 50$.

	Most Similar Word to:				
	‘eight’	‘october’	‘president’	‘american’ - ‘america’+ ‘europe’ (goal : ‘european’)	‘our’ - ‘we’ + ‘they’ (goal: ‘their’)
Word2vec	six, five, four	january, july, september	election, public, capital	american, national, other	than, links, they
MM CE	o, open, son	give, january, continent	party, museum, center	europe, persian, g	replaced, politics, troops
MM QH	five, september, december	november, december, koestler	election, over, wife	europe, late, amsterdam	there, they, these

performance improvement over the cross-entropy loss, but note that it is very competitive in terms of classification accuracy, and that it has the benefit of converging to within a relative convergence tolerance within a reasonable time.

For the quadratic hinge to compete with the cross-entropy loss model, we also need to know that the word embeddings have some semantic quality, allowing us to find word relations by comparing their embeddings. Table 3 shows the most similar word embedding to a small sample of words and two analogical reasoning tasks. We find that both algorithms offer thematically similar words to ‘october’ and ‘president’. The analogical task is not yet solved by any of the models at this corpus size N. To extract the semantic properties of the word embeddings, we likely need a larger corpus size to train on.

More exact testing of the semantic quality of the word embeddings requires the evaluation of the word embeddings relative to a labeled set of semantic tasks. To estimate the embeddings for the necessary vocabulary size M to include the words contained in the 19,544 standardized tasks also studied by Mikolov et al. (2013b), we evaluate the potential of sampling for extending the vocabulary size.

4.3 Sampling methods compared

We now consider the third research question, the issue of negative sampling to improve scalability of the models considered in the previous sections. We first observe computation time per iteration under three sampling methods, as compared to the unsampled models to observe whether the sampling method sufficiently extends scalability. We then run the MM-optimized word2vec model with cross-entropy loss, and with the quadratic hinge loss, for each of the subsampling methods. We compare classification accuracy in- and out-of-sample to study the classification quality of the embeddings obtained, and also consider performance on a word analogy task to observe semantic interpretability of the models.

4.3.1 Effect of sampling on scalability of MM-optimized word2vec

Figure 5: Computation time per iteration of word2vec MM-optimization, relative to vocabulary size M, by sampling method and loss function ((a): Cross Entropy, (b): Quadratic Hinge), for $N = 100{,}000$, $k = 10$, $c = 25$

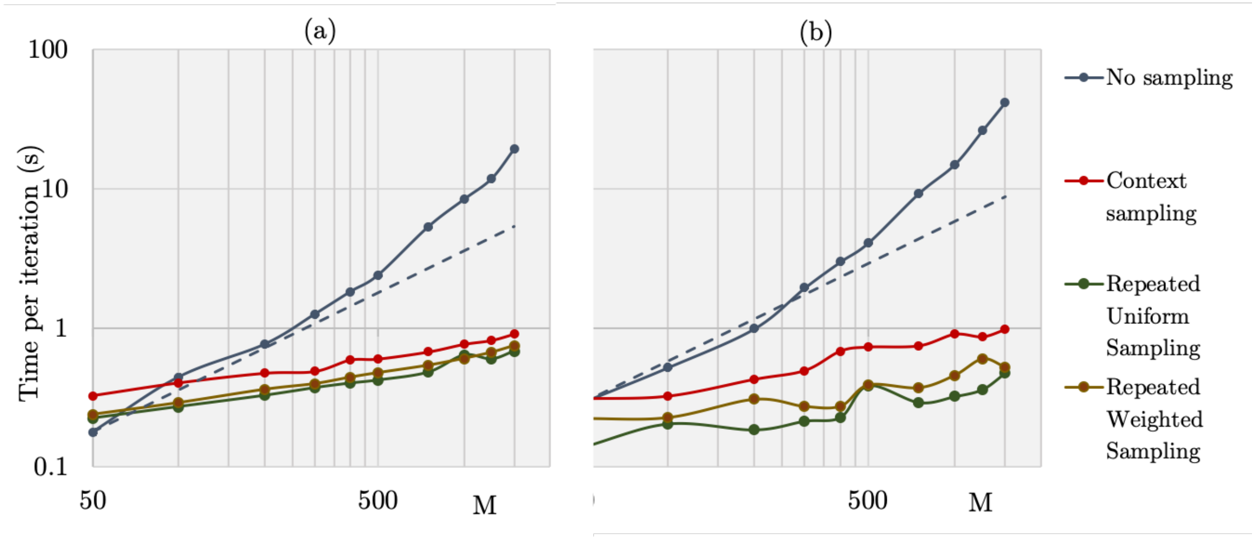
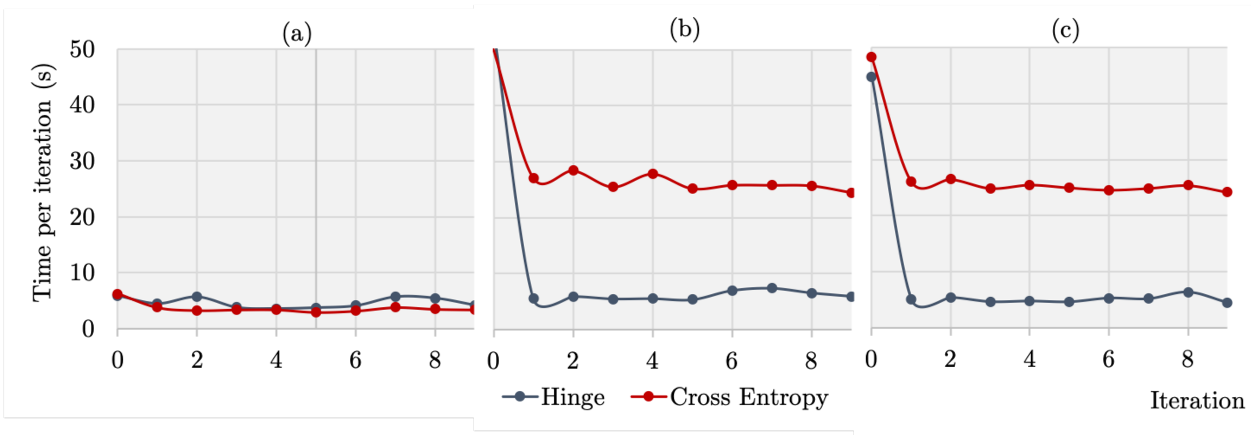


Figure 5 shows computation time per iteration for the MM-optimized word2vec algorithm for differing sampling methods, relative to the vocabulary size M. We find that sampling greatly increases scalability of the algorithm relative to the vocabulary size, with computation time trending less than proportionally with M. For this particular parameterisation, we also find no computation time penalty for repeated sampling, relative to one-off sampling by using the context matrix $\boldsymbol{Q}$ as our sampling matrix $\boldsymbol{P}$. This suggests that the computational overhead of repeated sampling is small relative to the other steps in the algorithm, and may be easily implemented if the resulting embeddings show an improvement over one-off sampling. Especially for applications with a smaller corpus N relative to the vocabulary size M, repeated sampling may benefit the robustness of the word embeddings and prevent overfitting on $\boldsymbol{Q}$.

We find that the negative sampling methods are successful in making the models more scaleable in M. This makes the computational complexity of the models linear in N, and less-than linear in M, which allows us to train the models on larger datasets, and particularly, larger vocabularies.

Figure 6: Sampled MM-optimization computation time for updating Z (a), U (b), V (c) by loss function, for $N = 5{,}000{,}000$, $c = 50$, $M = 5000$, $k = 10$.



In the methodology, we also hypothesized that the quadratic hinge model may benefit from additional sparsity in its update equations. For any correctly classified word instance, the entry d_{im} in matrix $\boldsymbol{D}$, which is used for the updates of $\boldsymbol{U}$ and $\boldsymbol{V}$ is equal to 0, meaning those elements can be skipped in calculating the updated parameters. The cross-entropy model does not have this benefit, since it still tries to optimise the probabilities of already correct classifications. Figure 6 shows the computation time per important step in the algorithm: updating the score matrix $\boldsymbol{Z}$, updating the parameter matrix $\boldsymbol{V}$ and updating $\boldsymbol{U}$. We see here that the quadratic hinge algorithm, particularly for this application on $N = 5{,}000{,}000$ words, is able to perform the updates to $\boldsymbol{U}$ and $\boldsymbol{V}$ up to 4 times faster. For the first iteration, most instances are incorrectly classified, and the updates take roughly equally long for both models. For this particular run, the $\boldsymbol{P}$ matrix has 50 million non-zero elements. However, the matrix $\boldsymbol{B}$ used for the updates of the parameters only has 4 million non-zero elements from the second iteration onwards. Given that the parameter updates take up the most time in each iteration, this allows the quadratic hinge algorithm to iterate much more frequently than the MM-optimized cross-entropy loss.

4.3.2 Negative sampling methods compared

With the sampling methods implemented, we may now scale our implementation of the algorithms to a larger corpus, which will also allow for testing the word embeddings on a word analogy task. We are interested to observe the performance for each sampling method, but also to observe any differences in performance between the model with the conventional cross-entropy loss, and the quadratic hinge loss. We implement the MM-optimization for 5,000,000 words from the 'text8' dataset, and set the vocabulary to the $M = 5{,}000$ most frequent words in the dataset. We employ the word analogy task set forth by Mikolov et al. (2013a). The task set includes 19,000 analogical reasoning tasks, out of which 3,403 overlap with our vocabulary. We test whether the word embeddings can solve the task exactly, but also evaluate whether the correct answer is within the top 10 most similar suggestions offered by the word embedding set, to observe whether the task is within the realm of solvability, given perhaps an even larger corpus to train on.

Table 4 shows classification performance and analogy task performance for our three sampling methods for each of the two loss functions implemented. We find that performance with context sampling in both classification and the word analogy task is very poor. For the quadratic hinge loss function in particular, even top-10 accuracy is only 3%. The cross-entropy loss fares better with this sampling method, but also underperforms relative to the other subsampling methods. This suggests that the context words are not suitable negative samples to train on. We may need to show the model negative samples that are further outside of its context of use to cluster the words in the vocabulary into distinct subjects. Context sampling may prohibit this by only having the words compete with words which are thematically similar, which would not allow us to distinguish a cluster of words from

Table 4: Classification accuracy and word analogy accuracy by loss function (CE: cross-entropy, QH: Quadratic Hinge), and sampling method, for N = 5,000,000 and M = 5,000 after 2 hours of training

Loss Function	Sampling	Accuracy	Top-10 Accuracy	Out-of-sample Accuracy	Out-of-sample Top-10 Accuracy	Word Analogy Accuracy	Word Analogy Top-10 Accuracy
CE	Context	13.1%	39.3%	12.8%	38.3%	3.2%	19.4%
QH	Context	2.7%	3.0%	2.6%	2.9%	0.0%	0.3%
CE	Uniform Rand.	**18.9%**	44.6%	**18.6%**	**43.6%**	3.4%	21.3%
QH	Uniform Rand.	14.5%	**44.8%**	14.2%	43.6%	**5.8%**	**30.5%**
CE	Weighted Rand.	17.3%	44.3%	16.8%	43.0%	3.4%	21.7%
QH	Weighted Rand.	13.8%	44.8%	13.5%	43.5%	5.1%	29.4%

another, since words in different clusters never compete for the same word instance.

We find that we cannot replicate the result presented by Mikolov et al. (2013b) that a frequency-based weighted negative sampling method results in the best performance. We test the suggested sampling distribution of the unigram frequency raised to the $3/4th$ power, but find it slightly underperforms relative to a simple random uniform. This suggests that the result may be particular to the stochastic gradient descent optimization algorithm and does not extend to MM optimization. It may also be that the result is particular to the larger scale of implementation. For our implementation, we give preference to the random uniform negative sampling distribution for its simplicity and better performance in both classification accuracy and the word analogy task.

In comparing the performance offered by each of the loss functions, we find that the cross-entropy loss model shows better classification accuracy, even out-of-sample. However, the quadratic hinge loss model shows better performance on the word analogy task. We hypothesize that this is symptomatic of the quadratic hinge loss giving more consideration to the misclassified words, which would allow for more sophisticated embeddings for each word in the vocabulary. The cross-entropy loss, which optimizes classification probabilities, may focus more on the few frequent words in the vocabulary, even if they are generally already correctly classified.

4.3.3 Sensitivity to subsampling

Mikolov et al. (2013b) suggest the subsampling of the matrix $\boldsymbol{Y}$, throwing away incidences of the most frequent words in the vocabulary, which may prohibit productive parameter learning. They find a performance improvement by randomly discarding a row of $\boldsymbol{Y}$ with the probability $P[\text{discard } \boldsymbol{y}_i] = 1 - \min[1, \sqrt{t/f(m_i*)}]$, with $f(m_i*)$ the frequency of the word m_i* for which $y_{im} = 1$ in row i, and t a chosen threshold, typically in the range $[10^{-5}, 10^{-3}]$. For any word with a frequency in the corpus lower than t, this probability reduces to 0. Words with a higher frequency than t are undersampled, with the undersampling rate positively related to the word frequency. The method aggressively undersamples very frequent words such as 'the', 'in' and 'of', and keeps all words in the vocabulary with a frequency lower than t. This is said to spread optimization efforts more equally over the vocabulary and prevents a model fit that only predicts a small set of words. To test whether the benefit of subsampling extends to our implementation, we apply their subsampling method with $t = 10^{-3}$ to our implementation of the MM-optimized models, for both the cross-entropy and the quadratic hinge loss. We do not find a replication of their improvement by this method, finding that the addition of subsampling of Y actually detriments the performance of the embeddings. For the MM-optimized model with cross-entropy loss, performance on the word analogy task drops from 3.4% to 3.3%, for the quadratic hinge model the drop is even more dramatic, from 5.8% to 0.2%. We hypothesize that the subsampling method benefits the stochastic gradient descent method of optimization, since it makes a sequence of N updates to the parameter matrices during an iteration. If a large number of these updates are concerned with the small subset of most frequent words, the embedding space may become imbalanced. Imbalance of word frequencies in the text corpus may also affect the MM-optimization, but it may be more robust to the imbalance since it considers the whole corpus with each update. Since the MM updates are not blind to their effect on the complete likelihood the way the stochastic gradient descent updates are, the updates will not focus on the most frequent words to an extent that may hurt the fit of the parameters for the complete vocabulary.

4.4 Application on full 'text8' dataset

For a final comparison of the MM-optimized models, we run each model for 4 hours on the complete text8 dataset and observe in-sample classification accuracy and performance on the word analogy task. We use $c = 100$, $k = 4$, and $M = 10{,}000$. We use the uniform random sampling method, which we found to perform best for $N = 5{,}000{,}000$. The larger vocabulary allows us to cover 7,211 tasks in the word analogy task set. Within the 4 hours of optimization, none of the models converge to within a tolerance of $\varepsilon = 10^{-4}$. For a sizable vocabulary and a large text corpus, there is likely still residual information on word relations in the data, even after a long period of optimization. We again find that the hinge loss majorization is able to perform quicker iterations, since it only needs to consider the instances with incorrect classifications. The average iteration for the quadratic hinge model takes 43.3 seconds, while the cross-entropy loss implementation takes 105.5 seconds per iteration.

For sake of comparison to the fastest publicly available models, we also run the implementation of gradient descent word2vec offered by the python package 'gensim'. The implementation is highly optimised, and makes use of the fact that the gradient descent method only considers one instance at a time, by streaming the dataset from memory. We specify the model to be as similar as possible to our MM-models, using $c = 100$, $k = 4$, $M = 10{,}000$ and $N = 17{,}000{,}000$, and uniform random sampling. We train the model for 5 complete passes of the data, and test its accuracy on the word analogy task.

We find that the cross-entropy loss function produces better in-sample classification results, but that the quadratic hinge model performs better on the word analogy task, achieving a task accuracy of 8.5%, relative to 5.4% for the cross-entropy loss function. This supports the previously obtained results on $N = 5{,}000{,}000$ words, suggesting that the quadratic hinge model is better suited for extracting semantic relations from the words. We find that both MM models underperform significantly relative to the gensim word2vec implementation on the word analogy task, which is able to achieve a word analogy task accuracy of 32%.

We show some examples of the obtained embeddings in Table 6, which shows the closest synonyms to a selection of words from the vocabulary. We see that there is significant overlap between the extracted word clusters by each of the models, which are of course primarily determined by the word co-occurrences in the text corpus. Table 7 shows the solutions offered by both models to a selection of word analogy tasks. The solutions again show significant overlap, with some minor differences in the rank of the solutions in the closest matches to the word task.

Table 5: Classification accuracy and word analogy task performance for the full ‘text8’ dataset, by loss function ($N = 17{,}000{,}000$, $M = 10{,}000$, $c = 100$, $k = 5$)

Loss function	Accuracy	Top-10 Accuracy	Word Analogy Accuracy	Word Analogy Top-10 Accuracy
cross-entropy	**18.1%**	44.2%	5.4%	24.6%
Quadratic Hinge	15.6%	**44.6%**	8.5%	**31.7%**
’gensim’ word2vec (5 epochs)	-	-	**32.0%**	-

Table 6: Most similar word embedding to a selection of words for MM-models with cross-entropy, and Quadratic Hinge loss. Estimated with $N = 17{,}000{,}000$, $M = 10{,}000$, $c = 100$.

	Most Similar Word to:				
	‘eight’	‘october’	‘president’	‘settlers’	‘nietzsche’
MM cross-entropy	two, zero, five	november, june, december	bill, presidential, administration	immigrants, invaders, colonists	kant, leibniz, aristotle
MM Quadratic Hinge	two, five, zero	november, june, december	chairman, administration, presidents	immigrants, colonists, invaders	hegel, kant, leibniz

Table 7: Most similar word embedding to a selection of word analogy tasks for MM-models with cross-entropy, and Quadratic Hinge loss. Estimated with $N = 17{,}000{,}000$, $M = 10{,}000$, $c = 100$, correct solutions in bold.

	Closest solution to the word analogy task:				
	father - man + woman	king - man + woman	heavier - heavy + small	running - run + dance	physician - physics + philosophy
MM cross-entropy	**mother**, brother, wife	henry, pope, charles	large, single, **smaller**	folk, musical, jazz	**philosopher**, historian, theologian
MM Quadratic Hinge	**mother**, wife, brother	henry, emperor, pope	large, **smaller**, simple	folk, musical, jazz	**philosopher**, physician, historian

5 Conclusion

In this thesis, we investigated the application of minimization by majorization for the word2vec algorithm, as well as the implementation of a quadratic hinge loss function. We considered three research sub-questions.

RQ1: Can minimization by majorization be used to accelerate word2vec learning?

We find that minimization by majorization, although initially slower in minimizing the cross-entropy loss, is more thorough in optimizing the word2vec loss than the conventional stochastic gradient descent algorithm. For smaller datasets in which the absolute best set of parameters is wanted for the particular corpus, it may achieve better minimization than the gradient descent algorithm. Without a negative sampling method applied, we find the method to have a complexity that is quadratic in M, making it poorly suitable for large applications without negative sampling. For applications on very large corpora, the data streaming method of the word2vec stochastic gradient descent may still be preferable, since our MM algorithm performs operations with large matrix representations of the text corpus. An acceleration of word2vec learning is possible with minimization by majorization, but will be dependent on the application, particularly the size of the vocabulary M and the corpus N.

RQ2: Does the use of a quadratic hinge loss function over the traditional cross-entropy loss accelerate parameter learning, and are the resulting embeddings of equal quality?

The quadratic hinge loss function has shown good performance on finding vector representations of words, that also have the popular semantic properties of word2vec embeddings. The resulting embeddings of quadratic hinge loss optimization show sensible word synonyms, and outperform the MM implementation with the conventional word2vec loss function in an analogical reasoning task. We find that the quadratic hinge model benefits from increased sparsity in its update equations across iterations, allowing it to iterate more frequently than the cross-entropy loss implementation, and focusing optimization efforts on misclassified cases. By measure of computation time per iteration, the quadratic hinge model offers a clear acceleration over the cross-entropy loss function. We also find it is quicker in extracting word embeddings with semantic properties. The only measure by which the quadratic hinge loss slightly lags the cross-entropy loss is the classification accuracy. We also find the quadratic hinge more likely to converge to within a relative convergence tolerance for corpora up to $N = 200{,}000$. For applications on a larger corpus, we find relative convergence unfeasible for any of the models implemented. For these applications, a set number of iterations or a cut-off time may be preferable.

RQ3: Can we use negative sampling to scale the MM-optimized models to larger text corpora?

We find that sampling greatly increases the scalability of the model to larger corpora and vocabularies, and find that a random uniform sampling method offers the best performance on both classification accuracy and the word analogy task, for both the cross-entropy loss and quadratic hinge loss models. We do not find a benefit to undersampling frequent words in $\boldsymbol{Y}$ for the MM-optimized model, which we hypothesize does not require such data preparation to spread its computation efforts over the vocabulary. This benefits the ease of use of the model, since it does not require the specification of a subsampling threshold, nor does it need a gradient descent step size or an optimized sampling distribution.

Our central research question was given by

Can we use a quadratic hinge loss function optimized with minimization by majorization to accelerate word2vec learning, without sacrificing word embedding quality?

Overall, we find an MM-optimized quadratic hinge loss function to offer a semantic performance improvement relative to a similarly optimized cross-entropy loss function. However, we find that both models do not compete with an efficient implementation of the stochastic gradient descent method. Although differences in computational overhead in the implementations are certain to play a role in this performance gap, we may also hypothesize that the parameter update structure of the MM algorithm is ill-suited for large corpora. Although the consideration of the whole corpus with each update allows the algorithm to be more thorough in the minimization of the loss function, this becomes inefficient for larger N. The stochastic gradient descent method is learning throughout a single iteration, allowing it to quickly learn a good set of word embeddings, rather than slowly learning the perfect set of embeddings. Although we note the promise of a quadratic hinge loss for word embedding learning, which has shown to be more efficient than an equally optimized cross-entropy loss model, we have not succeeded in accelerating word2vec learning.

6 Discussion

A few key shortcomings of this work, as well as potential extensions for future research can be identified.

We note the limited scale of our application, which does not have the size of implementations in the industry, where text corpora may have N in the billions. The performance of the quadratic hinge loss word embeddings relative to the cross-entropy embeddings may be different when trained with a larger corpus N relative to the vocabulary size M. However, the algorithm has the potential to be altered for further scalability. We may consider batched learning in which a sequence of smaller texts is used to update the parameter set $(\boldsymbol{U},\boldsymbol{V})$ in steps, similar to the stochastic gradient descent algorithm of conventional word2vec, which performs N smaller updates.

The MM-optimization algorithm with negative sampling may also be adapted for streaming the text data from disk. This would allow the algorithm to borrow from achievements in the highly optimised 'gensim' repository of word embedding models, likely offering a significant acceleration to our implementation.

In this thesis, we also limited ourselves to the estimation of the continuous-bag-of-words application of word2vec. The skip-gram model, which aims to predict the context words from the target word rather than vice versa, may also benefit from an alternative optimization method or loss version. A natural extension to this research would be to evaluate the MM optimization and the quadratic hinge loss in the context of the skip-gram model.

In investigating different sampling methods, we found that we could not replicate the benefit of frequency-based sampling as proposed by Mikolov et al. (2013b). The MM-optimized quadratic hinge loss function clearly reacts differently to the corpus composition than the word2vec model, and may require a particular sampling solution. Further research may pursue whether the uniform random sampling used in this thesis consistently outperforms frequency-based sampling in different scales of implementation, or whether another sampling distribution can be found that best supports our quadratic hinge adaptation of word2vec.

We also note that the quadratic hinge used in this research is only one of many alternative loss functions that may produce accurate word embeddings without requiring the calculation of choice probabilities. We may consider the absolute hinge loss, as well as the Huber loss function as substitutes for the quadratic hinge loss function. Moreover, we studied a quadratic hinge specification in which a separate loss term is associated with a positive classification ($y_{im}(\max(1-z_{im},0)^2)$), and a negative classification ($(1-y_{im})(\max(1+z_{im},0)^2))$). A more parsimonious model could be limited to a loss for a positive classification, and a parameter penalty term that encourages sparsity in $(\boldsymbol{U},\boldsymbol{V})$, which may further reduce the computational requirements of the hinge-loss word2vec implementation.

References

Bartels, R. H. and Stewart, G. W. (1972). Solution of the matrix equation $AX + XB = C$. *Communications of the ACM*, 15(9):820–826.

Chung, Y.-A., Wu, C.-C., Shen, C.-H., Lee, H.-Y., and Lee, L.-S. (2016). Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder. *arXiv preprint arXiv:1603.00982*.

Golub, G., Nash, S., and Van Loan, C. (1979). A Hessenberg-Schur method for the problem $AX + XB = C$. *IEEE Transactions on Automatic Control*, 24(6):909–913.

Groenen, P. J. F. and Josse, J. (2016). Multinomial multiple correspondence analysis. *arXiv preprint arXiv:1603.03174*.

Groenen, P. J. F., Nalbantov, G., and Bioch, J. C. (2008). SVM-Maj: a majorization approach to linear support vector machines with different hinge errors. *Advances in Data Analysis and Classification*, 2(1):17–43.

McFadden, D. (1973). Conditional logit analysis of qualitative choice behavior. *Frontiers in Econometrics*, pages 105–142.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Ng, P. (2017). dna2vec: Consistent vector representations of variable-length k-mers. *arXiv preprint arXiv:1701.06279*.

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Rong, X. (2014). Word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Wang, X. (2008). Method of steepest descent and its applications. *IEEE Microwave and Wireless Components Letters*, 12:24–26.

A Regularizing MM-optimized word2vec

The cross-entropy loss function of word2vec, as defined in our methodology, is given by $\ell(\boldsymbol{U},\boldsymbol{V}) = -\sum_{i=1}^{N}\sum_{m=1}^{M} y_{im} log(\pi_{im})$. Being based on the log of the estimated probabilities π_{im}, the cross-entropy loss may have difficulty converging to within a relative tolerance ε, since there may always be remaining gains in optimising the probabilities π_{im} to obtain a lower $-log(\pi_{im})$. For implementations in which we have a large set of parameters $2Mc$ relative to the size of the corpus N, it may benefit the loss function to keep increasing particular values of $\boldsymbol{Z}$ to extreme values, which may prohibit convergence without contributing to a sensible parameter set. In this appendix to our main research, we present a method of regularization which circumvents this issue. Because the regularization comes at a computational cost that prohibits its efficient use for larger vocabularies M, we do not include it as one of our main methods. However, we can show for a smaller application with N = 50,000, M = 500, that regularization can lead to faster convergence without sacrificing classification accuracy.

We append the MM-optimization of the cross-entropy loss presented in Section 2.3, by adding a penalty for the elements of $\boldsymbol{U}$ and $\boldsymbol{V}$. If calibrated correctly, this makes increments of individual values of $\boldsymbol{Z}$ to keep increasing the probabilities $\boldsymbol{\Pi}$ too expensive, and will allow for convergence of $\ell(\boldsymbol{U},\boldsymbol{V})$. Adding a ridge penalty to both $\boldsymbol{U}$ and $\boldsymbol{V}$ gives us the revised loss function

$$\ell(\boldsymbol{U},\boldsymbol{V}) = -\sum_{i=1}^{N}\sum_{m=1}^{M}\left(y_{im}\log(\exp(z_{im})/(\sum_{l=1}^{M}\exp(z_{il})))\right) + \lambda\,\mathrm{tr}(\boldsymbol{U}'\boldsymbol{U}) + \lambda\,\mathrm{tr}(\boldsymbol{V}'\boldsymbol{V}).$$

The additional terms can be carried over directly to the majorizing function. We skip a few steps already shown in section 2.3 to obtain the majorizing function

$$\begin{aligned} g(\boldsymbol{Z},\boldsymbol{Z}^0) &= \sum_{i=1}^{N}\sum_{m=1}^{M}\left(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m r_{im} + \frac{1}{4}(\boldsymbol{x}_i'\boldsymbol{U}\boldsymbol{v}_m)^2 + c_{im}\right) + \lambda\,\mathrm{tr}(\boldsymbol{U}'\boldsymbol{U}) + \lambda(\mathrm{tr}(\boldsymbol{V}'\boldsymbol{V})) \\ &= \mathrm{tr}(\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'\boldsymbol{R}') + \frac{1}{4}\mathrm{tr}(\boldsymbol{V}\boldsymbol{U}'\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}\boldsymbol{V}') + \lambda\,\mathrm{tr}(\boldsymbol{U}'\boldsymbol{U}) + \lambda\,\mathrm{tr}(\boldsymbol{V}'\boldsymbol{V}) + c. \end{aligned}$$

If we optimize this function relative to the parameters in $\boldsymbol{V}$, we obtain

$$\frac{\partial g(\boldsymbol{Z},\boldsymbol{Z}^0)}{\partial \boldsymbol{V}} = \boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0 + \frac{1}{2}\boldsymbol{V}\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0 + 2\lambda\boldsymbol{V} = \boldsymbol{0},$$

which results in the update equation

$$\boldsymbol{V} = -2\boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0 + 4\lambda\boldsymbol{I})^{-1},$$

where $\boldsymbol{I}$ is an identity matrix of size M. We see that this is almost exactly equivalent to the regular update equation. For the matrix of input weights $\boldsymbol{U}$ the update equation is less straight forward. We can derive the derivative of the majorizing function

$$\frac{\partial g(\boldsymbol{Z},\boldsymbol{Z}^0)}{\partial \boldsymbol{U}} = \boldsymbol{V}^{0\prime}\boldsymbol{R}'\boldsymbol{X} + \frac{1}{2}\boldsymbol{V}^{0\prime}\boldsymbol{V}^0\boldsymbol{U}'\boldsymbol{X}'\boldsymbol{X} + \lambda\boldsymbol{U} = \boldsymbol{0}.$$

Rewriting this further gives us a Sylvester equation (of the form $\boldsymbol{A}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{B} = \boldsymbol{C}$):

$$\frac{1}{2}\boldsymbol{V}^{0\prime}\boldsymbol{V}^0\boldsymbol{U} + 2\lambda\boldsymbol{U}(\boldsymbol{X}'\boldsymbol{X})^{-1} = -\boldsymbol{V}'\boldsymbol{R}'\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}$$

which can be solved by means of the Bartels and Stewart (1972) algorithm. This carries some additional computational cost relative to the unregularized update equations, and therefore requires a tradeoff between the desire to achieve a convergence criterion and the speed of optimization. Golub et al. (1979) show that the computational complexity of solving the Sylvester equation is of a complexity $\mathcal{O}(M^3 + c^3)$. This means that for every ten-fold increase in M, the complexity of our regularized parameter updates increase by up to 10^3. However, for smaller texts, for which one is interested in obtaining embeddings for a M in the hundreds, the implementation is feasible. To demonstrate, we apply the regularized algorithm to a subset of 'text8' of N = 50,000 words, and set $M = 500$, $c = 25$. Table 8 shows time to convergence for increasing values of λ. We find that higher values in λ aid in reducing the time to convergence. However, the regularized updates are 2 to 3 times slower per iteration than the unregularized algorithm. For a lower value of $\lambda = 8$, this results in the algorithm actually being slower to converge to within $\varepsilon = 10^{-5}$. We find that for $\lambda = 16$, the algorithm converges 3 times faster than the unregularized model, with equal out-of-sample classification accuracy. For larger values of λ, we find that the parameter cost starts to interfere with classification performance.

For these smaller applications, regularized word2vec can therefore be value in achieving convergence at a quicker rate, but the computational complexity quickly increases with M. For larger vocabularies, we may give preference to the quadratic hinge adaptation

Table 8: Time-to-convergence and classification accuracy for regularized word2vec for varying values of the parameter cost λ, using $N = 50{,}000$, $M = 500$, $c = 25$.

λ	Time to convergence (s)	Iterations	Time per iteration (s)	In-sample Accuracy	Out-of-sample accuracy
0	1648	2298	0.72	28.1%	21.8%
8	2282	965	2.36	24.6%	22.0%
16	988	721	1.37	24.6%	21.9%
32	799	550	1.45	23.0%	21.5%
64	1085	490	2.21	20.6%	20.2%

of word2vec, which does not have this issue of parameter drift. Because it attaches a zero loss to any correctly classified z_{im}, it will converge when it can no longer make gains in classification, rather than continue to optimise probabilities pi_{im}. An alternative solution is to rely on running the algorithm to a set number of iterations, or limiting optimization to a particular learning time. For very large corpora, the computational burden of going through the whole corpus even once means that convergence to within a small ε is likely not even feasible to start with. For example, Mikolov et al. (2013a) train their model on a corpus of 6 billion words by passing through the corpus only three times. For our research, we consider regularization an interesting but suboptimal extension of the MM-optimized word2vec model. Future research may explore methods of regularization that offer a smaller computation cost.

B Algorithms

B.1 Gradient Descent word2vec

Algorithm 3: Gradient Descent word2vec with cross-entropy loss.

```
Result: Optimization of paramater matrices $\boldsymbol{U}$, $\boldsymbol{V}$.
Input: Text corpus $\boldsymbol{d}$, gradient descent step size $\eta$, step size decay $\nu$, convergence tolerance $\varepsilon$,
  context size $k$, embedding size $c$, vocabulary size $M$.
Let $\boldsymbol{d}$ be the text corpus as a vector of strings $[str1, str2, ..., strN]$.
Let $\boldsymbol{l}$ be the vocabulary vector, the $M$ most frequent strings in $\boldsymbol{d}$.
Encode $N \times M$ matrix $\boldsymbol{Y}$ from text corpus, where $y_{im} = 1$ if $d_i = l_m$, 0 otherwise.
Encode $N \times M$ matrix $\boldsymbol{Q}$ from text corpus, where $q_{im} = 1$ if $max(y_{i-k/2,m},...,y_{im},...,y_{i+k/2,m}) = 1$,
  0 otherwise.
Set $\boldsymbol{X} = \boldsymbol{Y} - \boldsymbol{Q}$.
Initialize $M \times c$ matrices $\boldsymbol{U}$, $\boldsymbol{U}^0$, $\boldsymbol{V}$, $\boldsymbol{V}^0$ with each of the elements drawn from a $N(0,1/c^2)$
  distribution.
Initialize $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{U}\boldsymbol{V}'$.
Initialize $\pi_{im} = \exp(z_{im})/(\sum_{l=1}^{M} \exp(z_{lm}))$ for each $i$, $m$, to construct $\boldsymbol{\Pi}$.
Initialize $f(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N} \sum_{m=1}^{M} -log(\pi_{im})y_{im}$, $f(\boldsymbol{U}^0,\boldsymbol{V}^0) = \infty$.
while $|(f(\boldsymbol{U},\boldsymbol{V}) - f(\boldsymbol{U}^0,\boldsymbol{V}^0))/f(\boldsymbol{U}^0,\boldsymbol{V}^0)| > \varepsilon$ do
    for $i = 1$ to $N$ do
        $\boldsymbol{z}_i = \boldsymbol{x}_i\boldsymbol{U}\boldsymbol{V}'$.
        $\boldsymbol{\pi}_i = \exp(\boldsymbol{z}_i)/(\sum_{m=1}^{M} \exp(z_{im}))$.
        $\boldsymbol{e} = \boldsymbol{y}_i - \boldsymbol{\pi}_i$.
        $\partial f(\boldsymbol{U},\boldsymbol{V})/\partial \boldsymbol{U} = \boldsymbol{x}_i(\boldsymbol{V}'\boldsymbol{e}_i)'$.
        $\partial f(\boldsymbol{U},\boldsymbol{V})/\partial \boldsymbol{V} = \boldsymbol{e}_i(\boldsymbol{U}'\boldsymbol{x}_i)'$.
        $\boldsymbol{U} = \boldsymbol{U} - \nu(\partial f(\boldsymbol{U},\boldsymbol{V})/\partial \boldsymbol{U})$.
        $\boldsymbol{V} = \boldsymbol{V} - \nu(\partial f(\boldsymbol{U},\boldsymbol{V})/\partial \boldsymbol{V})$.
    end
    Update $\boldsymbol{Z}$,$\boldsymbol{\Pi}$,$f(\boldsymbol{U},\boldsymbol{V})$.
    Shrink $\eta$ by multiplying by $\nu$.
end
```

B.2 MM-optimized cross-entropy word2vec with subsampling

Algorithm 4: MM-trained word2vec with quadratic hinge loss.

```
Result: Optimization of paramater matrices $\boldsymbol{U}$, $\boldsymbol{V}$.
Input: Text corpus $\boldsymbol{d}$, convergence tolerance $\varepsilon$, context size $k$, embedding size $c$, vocabulary size
  $M$, sampling method.
Let $\boldsymbol{d}$ be the text corpus as a vector of strings $[str1, str2, ..., strN]$.
Let $\boldsymbol{l}$ be the vocabulary vector, the $M$ most frequent strings in $\boldsymbol{d}$.
Encode $N \times M$ matrix $\boldsymbol{Y}$ from text corpus, where $y_{im} = 1$ if $d_i = l_m$, 0 otherwise.
Encode $N \times M$ matrix $\boldsymbol{Q}$ from text corpus, where $q_{im} = 1$ if $max(y_{i-k/2,m}, ..., y_{im}, ..., y_{i+k/2,m}) = 1$,
  0 otherwise.
Set $\boldsymbol{X} = \boldsymbol{Y} - \boldsymbol{Q}$.
Initialize $M \times c$ matrices $\boldsymbol{U}$, $\boldsymbol{U}^0$, $\boldsymbol{V}$, $\boldsymbol{V}^0$ with each of the elements drawn from a $N(0,1/c^2)$
  distribution.
Initialize $N \times M$ sampling matrix $\boldsymbol{P} = \boldsymbol{0}$, set $p_{im} = 1$ if $y_{im} = 1$, otherwise:
    if using context sampling: set $p_{im} = 1$ if $x_{im} = 1$,
    if using random uniform sampling: $P[p_{im} = 1] = 1/M$,
    if using weighted frequency-based sampling: $P[p_{im} = 1] = (\sum_{j=1}^{N} y_{jm})^{3/4} / (\sum_{l=1}^{M} (\sum_{j=1}^{N} y_{jl})^{3/4})$.
Initialize $\boldsymbol{Z}$, with $z_{im} = p_{im} x_i \boldsymbol{U} \boldsymbol{v}'_m$.
Construct $\boldsymbol{\Pi}^*$, where $\pi^*_{im} = p_{im} \exp(z_{im}) / (\sum_{l=1}^{M} p_{im} \exp(z_{im}))$. Initialize
  $f(\boldsymbol{U},\boldsymbol{V}) = -\sum_{i=1}^{N} \sum_{m=1}^{M} y_{im} \log(\pi^*_{im})$, $f(\boldsymbol{U}^0,\boldsymbol{V}^0) = \infty$.
while $|(f(\boldsymbol{U},\boldsymbol{V}) - f(\boldsymbol{U}^0,\boldsymbol{V}^0)) / f(\boldsymbol{U}^0,\boldsymbol{V}^0)| > \varepsilon$ do
    $\boldsymbol{U}^0 = \boldsymbol{U}$.
    $\boldsymbol{V}^0 = \boldsymbol{V}$.
    $\boldsymbol{Z}^0 = \boldsymbol{Z}$.
    $\boldsymbol{R} = \boldsymbol{P} \odot (\boldsymbol{\Pi}^* - \boldsymbol{Y}) - 1/2\boldsymbol{Z}^0$.
    $\boldsymbol{V} = \boldsymbol{V}^0 + \boldsymbol{R}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}$.
    $\boldsymbol{U} = \boldsymbol{U}^0 + (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'(\boldsymbol{R})\boldsymbol{V}(\boldsymbol{V}'\boldsymbol{V})^{-1}$.
    Redraw $\boldsymbol{P}$ if using random uniform or weighted random sampling
    Update $\boldsymbol{Z}$, $\boldsymbol{\Pi}^*$ $f(\boldsymbol{U},\boldsymbol{V})$.
end
```

B.3 MM-optimized quadratic hinge word2vec with subsampling

Algorithm 5: MM-trained word2vec with quadratic hinge loss.

```
Result: Optimization of paramater matrices $\boldsymbol{U}$, $\boldsymbol{V}$
Input: Text corpus $\boldsymbol{d}$, convergence tolerance $\varepsilon$, context size $k$, embedding size $c$, vocabulary size
  $M$, sampling method.
Let $\boldsymbol{d}$ be the text corpus as a vector of strings $[str1, str2, ..., strN]$
Let $\boldsymbol{l}$ be the vocabulary vector, the $M$ most frequent strings in $\boldsymbol{d}$
Encode $N \times M$ matrix $\boldsymbol{Y}$ from text corpus, where $y_{im} = 1$ if $d_i = l_m$, 0 otherwise
Encode $N \times M$ matrix $\boldsymbol{Q}$ from text corpus, where $q_{im} = 1$ if $max(y_{i-k/2,m},...,y_{im},...,y_{i+k/2,m}) = 1$,
  0 otherwise
Set $\boldsymbol{X} = \boldsymbol{Y} - \boldsymbol{Q}$
Initialize $M \times c$ matrices $\boldsymbol{U}$, $\boldsymbol{U}^0$, $\boldsymbol{V}$, $\boldsymbol{V}^0$ with each of the elements drawn from a $N(0,1/c^2)$
  distribution.
Initialize $N \times M$ sampling matrix $\boldsymbol{P} = \boldsymbol{0}$, set $p_{im} = 1$ if $y_{im} = 1$, otherwise:
    if using context sampling: set $p_{im} = 1$ if $x_{im} = 1$,
    if using random uniform sampling: $P[p_{im} = 1] = 1/M$,
    if using weighted frequency-based sampling: $P[p_{im} = 1] = (\sum_{j=1}^{N} y_{jm})^{3/4}/(\sum_{l=1}^{M}(\sum_{j=1}^{N} y_{jl})^{3/4})$,
Initialize $\boldsymbol{Z}$, with $z_{im} = p_{im} x_i \boldsymbol{U} \boldsymbol{v}'_m$
Initialize $f(\boldsymbol{U},\boldsymbol{V}) = \sum_{i=1}^{N} \sum_{m=1}^{M} \left(y_{im} \max(1 - z_{im},0)^2 + p_{im}(1 - y_{im}) \max(z_{im} + 1,0)^2\right)$,
  $f(\boldsymbol{U}^0,\boldsymbol{V}^0) = \infty$
while $|(f(\boldsymbol{U},\boldsymbol{V}) - f(\boldsymbol{U}^0,\boldsymbol{V}^0))/f(\boldsymbol{U}^0,\boldsymbol{V}^0)| > \varepsilon$ do
    $\boldsymbol{U}^0 = \boldsymbol{U}$
    $\boldsymbol{V}^0 = \boldsymbol{V}$
    $\boldsymbol{Z}^0 = \boldsymbol{Z}$
    Construct $\boldsymbol{S}$, where $s_{im} = 1$ if $z^0_{im} < 1$, 0 otherwise
    Construct $\boldsymbol{T}$, where $t_{im} = 1$ if $z^0_{im} > -1$, 0 otherwise
    Construct $\boldsymbol{B}$, where $b^*_{im} = -y_{im} s_{im}(1 - z^0_{im}) + (p_{im} - y_{im}) t_{im}(1 + z_{im})$
    $\boldsymbol{V} = \boldsymbol{V}^0 + \boldsymbol{B}'\boldsymbol{X}\boldsymbol{U}^0(\boldsymbol{U}^{0\prime}\boldsymbol{X}'\boldsymbol{X}\boldsymbol{U}^0)^{-1}$
    $\boldsymbol{U} = \boldsymbol{U}^0 + (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'(\boldsymbol{B})\boldsymbol{V}(\boldsymbol{V}'\boldsymbol{V})^{-1}$
    Redraw $\boldsymbol{P}$ if using random uniform or weighted random sampling
    Update $\boldsymbol{Z}$, $f(\boldsymbol{U},\boldsymbol{V})$,
end
```