Distributional Initialization of Neural Networks

Irina Sergienya

München 2016

Distributional Initialization of Neural Networks

Irina Sergienya

Inaugural-Dissertation
zur Erlangung des Doktorgrades der Philosophie
an der Ludwig–Maximilians–Universität
München

vorgelegt von
Irina Sergienya
aus Belarus

München 2016

Erstgutachter: Prof. Dr. Hinrich Schütze
Zweitgutachter: Prof. Dr. Peer Kröger
Gutachter: Dr. habil. Helmut Schmid
Datum der mündlichen Prüfung: 05.07.2016

П р е д с е д а т е л ь к о м и с с и и. Вы читаете на нескольких языках, знакомы с высшей математикой и можете выполнять кое-какие работы. Считаете ли вы, что это делает вас Человеком? О т а р к. Да, конечно. А разве люди знают что-нибудь еще?

(Из допроса отарка. Материалы Государственной комиссии)

Север Гансовский. "День гнева"

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

In Natural Language Processing (NLP), together with speech, text is one of the main sources of information. Computational systems that process raw text need to perform a transformation of text input into machine-readable format. Final performance of the NLP systems depends on the quality of these input representations, that is why the main objective for representation learning is to keep and highlight important features of the input tokens (characters, words, phrases, etc.).

Traditionally, for Neural Networks (NNs) such input representations are one-hot vectors, where each word is represented with a vector of all-but-one zeros, with value 1 on the position that corresponds to the index of the word in the vocabulary. Such a representation only helps to differentiate words, but does not contain any usable information about relations between them. Word representations that are learned by NNs – word embeddings – are then arranged in a matrix, where each row corresponds to a particular word in a vocabulary and is retrieved by multiplication of the corresponding one-hot vector and the embedding matrix.

These word embeddings are initialized randomly, and during training adjust their values to capture the contextual semantic information with respect to the training objective. When a word is frequent, it is seen often during training and its representation is updated frequently; for the same reason embeddings for rare words experience much less updates. This makes it difficult for NNs to learn good word embeddings for words that occur just several times in a corpus.

In this work, we propose a method to improve quality of word embeddings of rare words. The main idea is to initialize a NN that learns embeddings with sparse distributional vectors that are precomputed for rare words from a given corpus.

We introduce and investigate several methods for building such distributional representations: with different ways to combine one-hot representations of frequent and distributional representations of rare words, different similarity functions between distributional vectors, different normalization approaches applied to the representations in order to control the input signals' amplitude.

We evaluate the performance of our proposed models on two tasks. On a word similarity judgment task, the embeddings of words are used to compute similarity scores between two words in given pairs; then these similarity scores are compared with human ratings. With use of the same NN architecture, word embeddings that are trained using distributional initialization show significantly better performance than word embeddings trained with

traditional one-hot initialization.

On language modeling task, where models compete in predicting probability of a given sequence of words, models with distributional initialization show minor improvements over models with one-hot initialization.

We also study a very popular word2vec tool (Mikolov et al., 2013a) that is used to obtain word embeddings without supervision. The main question we ask is how much the quality of learned word embeddings depends on the initial random seed. The obtained results suggest that training with word2vec is stable and reliable.

Acknowledgments. This work was supported by Deutsche Forschungsgemeinschaft (grant DFG SCHU 2246/10-1, FADeBaC).

Zusammenfassung

Text ist neben Sprache eine der Hauptinformationsquellen in der natürlichen Sprachverarbeitung. Um Rohtexte zu verarbeiten, müssen Computer die Texteingabe zunächst in maschinenlesbares Format umwandeln. Von der Qualität dieser Eingaberepräsentation hängt die finale Leistung von Sprachverarbeitungssystemen ab. Hauptziele des Repräsentationslernens sind daher der Erhalt und die Hervorhebung wichtiger Eigenschaften der Eingabe (Buchstaben, Wörter, Phrasen, etc.).

Traditionelle Eingaberepräsentationen für neuronale Netze (NNs) sind sogenannte 1-aus-N Vektoren, die jedes Wort als einen Vektor darstellen, der nur aus Nullen und einer Eins an jener Position besteht, die dem Index des Wortes im Vokabular entspricht. Solche Repräsentationen können zwar Wörter differenzieren, enthalten aber keine weiteren Informationen, z.B. bezüglich Relationen zwischen Wörtern. Wortrepräsentationen können andererseits auch von NNs gelernt werden. Diese sogenannten Worteinbettungen werden meist in einer Matrix angeordnet, in der jede Zeile einem bestimmten Wort in einem Vokabular entspricht. Für das Training kann durch Multiplikation des zugehörigen 1-aus-N Vektors und der Einbettungsmatrix auf sie zugegriffen werden.

Worteinbettungen werden meist zufällig initialisiert und während des Trainings so angepasst, dass sie kontextabhängige semantische Informationen bezüglich des Trainingsziels widerspiegeln. Da häufige Wörter oft während des Trainings gesehen werden, werden ihre Repräsentationen mehrfach aktualisiert. Aus demselben Grund werden Einbettungen seltener Wörter weitaus weniger angepasst. Dies erschwert es NNs, gute Worteinbettungen für Wörter zu lernen, die nur wenige Male in einem Korpus auftreten.

In dieser Arbeit schlagen wir eine Methode vor, um die Qualität von Worteinbettungen für seltene Wörter zu verbessern. Dazu wird ein NN, das Einbettungen lernt, mit dünnbesetzten verteilten Vektoren initialisiert, die für seltene Wörter aus einem gegebenen Korpus vorberechnet werden.

Wir führen verschiedene Methoden ein, solche verteilten Initialisierungsvektoren zu erstellen und untersuchen sie: Wir betrachten unterschiedliche Möglichkeiten, 1-aus-N Repräsentationen für häufige Wörter und verteilte Vektoren für seltene Wörter zu kombinieren, vergleichen Ähnlichkeitsfunktionen für verteilte Vektoren und stellen Normalisierungsansätze vor, die auf die Repräsentation angewandt werden können, um die Amplitude des Eingabesignals zu kontrollieren.

Zur Bewertung unserer vorgeschlagenen Modelle betrachten wir zwei Aufgaben. Die erste Aufgabe ist die Beurteilung von Wortähnlichkeiten. Dabei werden Worteinbettun-

gen verwendet, um Ähnlichkeiten zwischen den Wörtern eines gegebenen Wortpaares zu berechnen. Diese werden dann mit menschlichen Bewertungen verglichen. Bei Verwendung der gleichen NN Architektur zeigen Worteinbettungen, die mit verteilten Initialisierungen trainiert wurden, signifikant bessere Leistungen als Worteinbettungen, die mit traditionellen 1-aus-N Initialisierungen trainiert wurden.

Die zweite Aufgabe ist Sprachmodellierung, das heißt, die Vorhersage der Wahrscheinlichkeit einer gegebenen Wortsequenz. Dabei zeigen Modelle mit verteilter Initialisierung geringfügige Verbesserungen gegenüber Modellen mit 1-aus-N Initialisierungen.

Wir betrachten außerdem das weit verbreitete word2vec (Wort-zu-Vektor) Programm (Mikolov et al., 2013a), das verwendet wird, um unüberwacht Worteinbettungen zu lernen. Die Hauptfrage, die wir untersuchen, ist, wie stark die Qualität der gelernten Worteinbettungen von dem Startwert der Zufallszahlen abhängt. Die erhaltenen Ergebnisse deuten darauf hin, dass das Training mit word2vec stabil und zuverlässig ist.

Chapter 1

Introduction

1.1 Motivation

Starting from 1950s, with development of computer science, the question of creation of machines that are powerful enough to show intelligent behavior arose. At first, expectations of researches from the 1950-60s were extremely positive: fast progress was achieved with coarse modeling methods, on non-tedious tasks together with quite unsatisfactory solution of hard ones (see Dreyfus (1972)). But all the bright expectation in AI research were broken in mid 1960s[1], that slowed down artificial intelligence (AI) development. It also was shown that much more efforts should be put in order to make systems more human-like and applicable to real-world problems.

It appeared to be hard to tackle the problem of AI creation all at once. Thus AI research field got split in several sub-fields, where researchers try to solve specific problems depending on the type of input information and human cognitive functions (e.g., thought, perception, memory, speech) to imitate. For example, perception is addressed in pattern recognition, data and text mining, information extraction; psycholinguistics focuses on the interconnection of awareness, thought and linguistics; knowledge extraction and management, expert systems and knowledge bases try to extract, keep and use knowledge from different sources; etc.

The modeling of the complete human cognition is not possible without modeling of human language. Speech played an extremely important role in the formation and development of humankind. Language helps people to communicate their experience, accumulate and transfer it through the time. Speech is the basis of the human thoughts; moreover, reasoning and abstract thinking are not possible without speech.

Natural Language Processing (NLP) is a field that studies how to process – understand and generate – natural language by the computational means.

There is a big number of problems that NLP is occupied with:

- speech and acoustic signals processing are sorted out by *speech recognition*, *speech*

[1]For examples see ALPAC report (Pierce and Carroll, 1966) with critique of machine translation or book by Minsky and Papert (1972) on perceptrons/neural networks.

segmentation, *digital signal processing*;

- written text is digitized by means of *optical character recognition*;
- digitized representation is then *tokenized on sentence or word level* (this is extremely important for unsegmented languages like Chinese, Japanese, Thai);
- syntactic information is then acquired with help of *parsing*, *part-of-speech tagging* and *morphological segmentation*;
- *word sense disambiguation*, *sentiment analysis*, *relationship extraction*, *coreference resolution* extract semantics from the text;
- *machine translation* tries to represent semantics of a sentence in a source language with a sentence in a target language;
- when search is needed, *information retrieval*, *information extraction*, *question answering* start to play role; and so on.

One of the popular toolkits to work on NLP problems is provided by *Machine Learning (ML)*. The idea behind ML is to make machines (computers) able to learn – "to improve automatically with experience"(Mitchell, 1997).

ML algorithms can be divided into 3 categories based on the feedback that is available for them during the training:

- supervised algorithms, when the input and desired output are given;
- unsupervised algorithms, when only input signals are given;
- algorithms with reinforcement, when some feedback is available during the training, but the relations between input and output are not well-defined.

One of the main disadvantages of supervised methods is that the correct outputs should be available for the algorithm to use, and to provide these correct outputs huge amount of human labor is needed. On the other hand, unsupervised methods can learn from raw data with no manual work involved.

There are many algorithms and models proposed by ML researchers: decision trees, support vector machines, linear classifiers, hidden Markov models to name a few. One of them, artificial neural networks, is in our particular interest.

Neural Networks (NNs) were proposed as an attempt to emulate human brain. Its main basic element is a neuron, that corresponds to a unit in a NN. Units are organized in interconnected layers: from the input layer where the input signals come from, through a number of hidden layers inside the network, to the output layer where the output of the network is produced. Signals undergo non-linear transformations passing through the network that emulates neural activity in a human brain.

Deep Learning (DL) is a part of ML that attempts to discover structure in the input data with employment of NNs with multiple layers and non-linear transformations. With a

success in vision and speech processing fields, DL also achieves good results in NLP tasks, e.g., question answering (dos Santos et al., 2016), part of speech tagging (Huang et al., 2015), paraphrase identification (Cheng and Kartsaklis, 2015).

The big area where DL plays an important role is *representation learning* (also called *feature learning*). In NLP, we rarely work with raw data itself; usually, the input is tokenized and each token (e.g., word or letter) receives its digital representation. Constructing this representation, we would like it to reflect properties that are helpful in solving particular NLP tasks. Many types of word representations exist: cluster-based representations, distributional representations, distributed representations (also known as *word embeddings*).

In Chapter 2, we would like to discuss existing approaches to build distributed word representations in more details and propose an extension of a well-known tool word2vec that allows to increase quality of learned word embeddings for rare words.

Another NLP task where word representations are useful is *language modeling* – task of predicting probability of a given word sequence. In decades of research, different approaches were proposed to tackle language modeling and its smoothing objective, such as n-gram models, factored language models, neural language models. Different DL architectures also were applied to solve this task. We will discuss them in detail in Chapter 3, together with our proposed extension of an existing log-bilinear language model.

word2vec, a very popular tool to learn distributed word representations, was developed by (Mikolov et al., 2013a). Together with its modifications, it is widely used by NLP researchers nowadays. Despite of the popularity, only several works attempts to study how and why word2vec works. In Chapter 4, we would like to shed some light on how usage of different initial random seeds influences on learned with word2vec word embeddings. We compare the structure of learned embedding spaces and perform qualitative analysis of the obtained embeddings.

1.2 Thesis outline

The main question we are investigating in the scope of this work is how to improve performance of NLP tasks through improving quality of word representations learned by NNs.

In Chapter 2, we talk about existing word representations and importance of learning good representations for rare words. We propose to address this problem by introducing distributional representations of words at the input layer of NN architecture. Then we describe a base of our work: word2vec tool and construction of distributional representations. Experiments follow, with description of training corpus, evaluation task, evaluation data sets and use of hyper-parameter θ that is responsible for determining which words are rare. Discussion of the results highlights the important findings and canvasses the role of chosen hyper-parameters (θ values and types of distributional models), variability of the results and scalability of the proposed approach. Further, related works are listed, with conclusion and possible extension of our approach at the end.

In Chapter 3, we talk about application of our suggestion – initialization of NNs with

distributional words representations – to language modeling task. We start introducing existing language models, continue with more detailed and formal description of language modeling framework with proposed distributional representations and their integration at the input layer of log-bilinear language model. We also talk about models training, interpolation and evaluation. Experimental setup describes corpus, target words vocabularies, scope of rare words threshold – hyper-parameter θ, and training regime of NN. In the Experiments, we give a detailed description of the set of experiments that we conduct in order to test our proposal: for each experiment, we first give a short motivation, followed by the used distributional representations and training parameters; results and their discussion follow. Conclusions that lead us to certain decisions in further experiments are mentioned at the end. The Discussion section sheds light on the performance with respect to the change of hyper-parameters and high-level decisions: choice of θ, distributional schemes, learning rate. Conclusion summarizes the results of experiments; and future work sketches the further research directions. Related work section briefly talks about works in the scope of our research topic.

In Chapter 4, we investigate the performance of word2vec depending on the initial random seed that is used to initialize the word embedding matrix. We first give overview of the different model types word2vec can train with the hyper-parameters involved. Then we describe the sources of randomization in word2vec: those variables and architecture decisions that result in different learned embeddings. We introduce our modification of the source code that allow us to change the random seed, and report the values of other hyper-parameters used to launch word2vec. Corpus and evaluation metrics are described right before we proceed to the experiments. Related work section sheds some light on the existing works that study word2vec tool. Conclusion summarizes the obtained results, and future work shares some ideas about the extensions of the current investigation.

1.3 Contributions of the Thesis

In our work, we present several original ideas:

1. Distributional initialization of neural networks (by the example of word2vec and vLBL).

2. Different treatment of words with different frequency by combination of distributional and one-hot vector representations.

3. Proposal of distributional representations with use of different combination schemes (mixed, separate, ONLY), different association functions (co-occurrence in a window, with use of positioned PPMI and positioned letter 3-grams), and different normalization schemes (non-diagonal elements were replaced with constant; scaled; rows or columns got normalized with different scaling coefficients).

4. Analysis of the word2vec tool: analysis of the learned word embeddings with respect to different initial random seeds used in the word embedding matrix initialization.

We present empirical evaluation of ideas 1–3 on word similarity judgment task and language modeling task.

Analysis of the proposed models is provided for different hyper-parameters and distributional representations, with the discussion of the achieved performance.

Analysis of the word2vec tool is performed by means of the models trained for different random seeds: we compare the number of the common words in the top 10 nearest neighbors, the distances between embedding vectors for words, and the difference in a quality of the learned embeddings employing word similarity judgment task.

Chapter 2

Learning Better Embeddings for Rare Words Using Distributional Representations

This chapter covers work already published at international peer-reviewed conferences. The relevant publication is (Sergienya and Schütze, 2015). The research described in this chapter was carried out in its entirety by myself. The other author(s) of the publication(s) acted as advisor(s) or were responsible for work that was reported in the publication(s), but is not included in this chapter.

2.1 Summary

In this chapter, we are going to talk about our investigation on learning better word embeddings for rare words using distributional representations for Neural Network (NN) training.

The main goal of this work is to improve distributed words representations, with emphasis on representations of rare words. There are two main types of word representations: low-dimensional dense word embeddings that capture the contextual semantic information and high-dimensional distributional vectors in which each dimension corresponds to a context word. We propose to initialize an embedding-learning model with distributional vectors. Evaluation on word similarity tasks shows that this initialization significantly increases the quality of embeddings learned for rare words.

2.2 Introduction

Machine Learning (ML) provides a great toolkit to solve real-world problems of different nature. One of the main challenges of the ML is to find a way to represent real-world objects in a machine-readable format, namely find an approach to build their explicit

numerical representations. Since results of ML algorithms deeply depend on the quality of their input, learning better representations for objects of interest is a way to improve the quality of ML systems in general.

In a course of ML research, many ways to represent raw input data were developed. The simplest one is *symbolic representation*: arrange objects in a list and then address them by their index in this list. Other widely used approaches start with decisions on substantial characteristics of objects of interest (*feature engineering*) and assess the presence of such characteristics on a given scale (*feature learning*).

Application-dependent feature learning is widely used in fields with perceptive input: to represent audio and visual signals. For audio signals, most popular are RASTA-PLP features [Hermansky (1990), Hermansky and Morgan (1994)] and MFCC features (Mermelstein, 1976) that capture energy in a signal at various frequencies. For visual processing, extracted features usually include information about contours (active contour model), edges, corners and points of interest, ridges, regions with constant properties, such as brightness or color (blob detection); image histograms and SIFT features (Lowe, 1999) are also widely used. In Natural Language Processing (NLP), a word feature vector usually includes a word's morphological structure, part-of-speech tags, shape features (e.g., capitalization, hyphenation, use of numbers).

The modern trend in ML is to reduce manual work needed for feature engineering by introduction of *representation learning* – an unsupervised way to automatically detect features from big amounts of unlabeled data. Taken into account the speed of growth of computational power, representation learning becomes a more and more interesting area of research.

Currently, big collections of unlabeled text data are available for language processing, so it is not surprising that many ways of automatic construction of word representations were proposed targeting NLP applications. They can be divided into 3 main groups: clustering-based, distributional word representations, and distributed word embeddings.

Clustering-based methods propose to cluster words from a corpus according to the contexts in which they occur. The most famous example of clustering approach was introduced by Brown et al. (1992) and is based on the idea that vocabulary partitioning should maximize the average mutual information. This representation is used in language modeling and represents the probability of the next word as a product of the probability of the next class and the probability of the next word given its class. The clustering approach is successfully used in many NLP tasks, e.g. name entity recognition (Miller et al., 2004), dependency parsing (Suzuki et al., 2009).

Distributional methods in modeling word representations rely on the idea called the *distributional hypothesis* originated from work (Harris, 1954): "difference of meaning correlates with difference of distribution." To put this another way, distributional regularities in the text correlate with its meaning; and distributional methods aim to detect these regularities. These methods represent words as vectors of a co-occurrence matrix in which rows correspond to target words and columns correspond to contexts. The main differences between proposed implementations of the distributional hypothesis arise from the definition of context, the association measure between word and its context, and the function

to measure similarity between two given words. Context representations include a window of words of a fixed size, or a sentence, or a paragraph, or a whole document that contains or corresponds to a target word; context can also reflect structural relationships: it can indicate whether a target word is in syntactic or semantic relationships with contexts, e.g. is a a head of noun phrase or is a subject of a predicate. Association measure between a target word and its context can be represented, e.g., with an indicator variable (0 or 1), a co-occurrence count or its modification (e.g. PMI, PPMI), or account for frequency of a target word and contexts (e.g. tf–idf). Similarity of given words is usually measured as a cosine or the Euclidean distance between their vectors. The most well-known examples of distributional models include:

- Hyperspace Analogue to Language (HAL) (Lund and Burgess, 1996): in this model, context vocabulary is the same as target vocabulary, context is a window of a fixed size; a target-context matrix contains a number of co-occurrences of that pair; similarity measure is the Euclidean distance.

- Latent Semantic Analysis (LSA) (Dumais et al., 1988) and Latent Semantic Indexing (LSI) (Deerwester et al., 1990) analyze relationships between a set of documents (contexts) and the terms (targets) they contain. The term-document matrix consists of tf–idf values, and singular value decomposition (SVD) is used to reduce the number of rows in it. Similarity between documents is measured by cosine between column vectors.

- Latent Dirichlet allocation (LDA) (Blei et al., 1993) is a generative probabilistic model of a corpus, where every document is seen as a mixture of latent topics, and each topic is characterized by a distribution over words.

- Construction of a co-occurrence matrix can be memory and time consuming, that is why several incremental approaches where proposed. Řehůřek and Sojka (2010) describe LSA matrix construction for document streaming. Random indexing by (Sahlgren, 2005) applies Johnson–Lindenstrauss lemma about random projections from high- to low-dimension Euclidean space to reduce dimensionality of co-occurrence matrix.

Distributed word embeddings are low-dimensional (usually from 50 to 500) dense vector representations of target vocabulary, where space geometry agrees with linguistic regularities. The main idea behind distributed word representations is to beat the *curse of dimensionality* that distributional methods suffer from: a test sequence of words is often never seen during the training, that leads to zero probability of that sequence; and in order to avoid such cases exponential number of parameters needed to be trained. Use of dense word representations with proper probability function of such representations reduces the number of trained parameters (makes it proportional to the size of the vocabulary) and helps to avoid zero probabilities of test sequences. Common ways to obtain word embeddings are to use NNs or via dimensionality reduction of co-occurrence matrices. Deep

Learning (DL), a branch of ML that uses multiple layer NNs with complex non-linear transformations for data processing, is currently one of the most popular approaches to apply NNs for embedding learning:

- Neural Network Language Model (NNLM) by Bengio et al. (2003) introduce NN architecture for language modeling task (predict the next word of a sequence given $n-1$ previous words). Words in the vocabulary are represented with low-dimensional real-valued feature vectors. This model shows better results than traditional n-gram language models.

- Collobert et al. (2011) use a DNN architecture where input is trainable word embeddings and output is a score per possible tag for a given task: part of speech tagging, name entity recognition, chunking, or semantic role labeling. They compare performance of this model with a model where initial word embeddings are trained in a language modeling manner on a large volume of unlabeled data, and show that use of pre-trained embeddings improves performance. They also explore an idea of multi-task learning – where embedding level of the NN architecture is shared between models with different supervised tasks.

- Log-Bilinear language model (LBL) by Mnih and Hinton (2007), learn word representations by predicting a target word embedding given embeddings of its context. It does not have non-linearities, but perform comparatively well to n-grams and other NNLMs. This model was optimized by means of hierarchical clustering (Mnih and Hinton, 2008) and negative sampling (Mnih and Teh, 2012) training procedure to hierarchical LBL.

- Turian et al. (2010) embeddings are trained based on combination of existing embeddings of Brown clusters, Collobert et al. (2011) and Mnih and Hinton (2008) with tuning on name entity recognition and chunking tasks.

- word2vec tool (Mikolov et al., 2013a) allows training of two log-linear models that became a popular baseline in NLP research. Proposed architectures aim to increase training speed while giving up the non-linearity. They have projection layer that is shared for all word positions in each context, followed by an output layer. Word embeddings are trained in an unsupervised manner by predicting a word given its context or predicting a context given middle word. In greater detail, word2vec models are described in Section 2.3.1.

Several techniques of dimensionality reduction of co-occurrence matrices exist, and their result are sometimes seen as a distributed representation formation:

- Statistical and algebraic methods include SVD, Principal Component Analysis (PCA, Pearson (1901)), Independent component analysis (ICA), Canonical correlation analysis (CCA, Hotelling (1936)), Two Steps CCA (TSCCA, Dhillon et al. (2012)), Sparse Random Projections (Li et al., 2006).

- Hellinger PCA proposed by Lebret et al. (2013) induces initial representations using Hellinger distance for PCA technique and then specializes them for tasks of named entity recognition and movie review scoring. NNs with such initialization perform similar or even outperform embeddings of Collobert et al. (2011), Turian et al. (2010), and Mnih and Hinton (2008).

- Another method of dealing with large dimensionality or co-occurrence matrix – Stochastic Low-Rank Approximation – was proposed by Lebret and Collobert (2015). It encodes a distribution of probabilities of words given their contexts by reducing reconstruction error of low-rank approximation.

- GloVe (Pennington et al., 2014) is a log-bilinear regression model that captures global corpus statistics directly from a given corpus by factorizing word-to-word co-occurrence matrix to obtain target and context embeddings.

(Levy et al., 2015) shows that despite of a confrontation between distributed and distributional methods for learning better word representations, popular word2vec and GloVe models can be reformulated in a matrix factorization problem with a specific choice of hyper-parameters. If such hyper-parameter values are applied to an SVD of PPMI co-occurrence matrix, the performance of SVD becomes comparable to the mentioned distributed models.

2.2.1 Rare words in NLP

Why should we care about rare words? According to Zipf's law, the frequency of any word in a given text corpus is inversely proportional to its rank in the frequency table. For example, in a ukWaC+WaCkypedia corpus (see Section 2.4.1), 82% of its vocabulary have frequency ≤ 5. Rare words are often words of interest for NLP applications: in languages with rich morphology, some word forms can appear very limited number of times in a training corpus; new words (or new word senses) appear all the time, (e.g., *застабил* (from slogan during the President elections in 2010 "За стабильность!" – "For stability!"): person that supports political course of the President in Belarus; *ватник* (from "ватник" – extremely pragmatic and cheap cotton-filled coat popular in USSR): person who is fanatic Russian/Soviet patriot, since 2011), or get borrowed from another language. It is tempting to fix vocabulary and ignore words below some frequency thresholds, but ignoring such words would mean to throw away useful and important information as well. Thus good estimation of rare words' embeddings can result in a better and richer performance of NLP applications.

2.2.2 Problem statement

In NLP with DL, traditional initialization of a NN is one-hot initialization, where all words are referred with their index in the vocabulary [e.g., Bengio et al. (2003), Collobert et al. (2011), Mikolov et al. (2013a)]. This index representation can be seen as a vector with

all-but-one zeros and one 1 at the word's index position. The main drawback of this one-hot representation is that it does not contain any usable information about a word except for its identity. For example, words like *laptop* and *laptops* have indices 6626 and 11769 respectively, that completely ignores the fact that these words are semantically the same and differ only in number. Hence one-hot representation makes it hard for NN to learn good word representations just from a few examples seen during the training. Adding some extra knowledge definitely could help the trainer to improve learned representations.

This work investigates one possible way to incorporate extra knowledge by means proposed in distributional semantics: for every target word in a vocabulary build a distributional vector and provide this vector as an input to a NN.

We study two ways to build distributional vectors, together with two association measurement schemes.

2.3 Methods

2.3.1 word2vec

The tool word2vec was introduced by Mikolov et al. (2013a). It provides two log-linear models, continuous bag-of-words and continuous skip-gram, to learn distributed word representations. One of the big advantages of the word2vec tool is that it avoids dense matrix multiplication; that makes its training extremely efficient and suitable to be applied to large amounts of unstructured text data.

The **continuous bag-of-words model (CBOW)** architecture consists of an input layer, one projection layer and an output layer as shown in Figure 2.1 (left). This model is called "bag-of-words" because it ignores the order of the words in the context at the projection layer: embeddings of the context words influence equally on the combined representation. Also words from the context are represented with the same embedding vector disregarding their position in the current context.

The **continuous skip-gram model (Skip-gram)** architecture consists of an input layer, one projection layer and an output layer as shown in Figure 2.1 (right). This model is a bag-of-words model: the projection layer is shared among all words, so the position information is not taken into account during the training. The model tries to maximize prediction of a word based on another word from the context. Precisely, given an input sequence of words from a training corpus, $w_1, w_2, ..., w_T$, the skip-gram model tries to maximize the average log probability:

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

where c is the size of the context window.

To estimate $p(w|w_I)$, Mikolov et al. (2013b) use a *hierarchical softmax*: a reduced version of a full softmax that benefits by structuring the possible output options in a

binary tree. This reduces the number of nodes for evaluation from *vocab_size* output nodes to $\log_2 vocab\_size$ output nodes. A Huffman binary tree (Huffman, 1952) is used in the implementation.

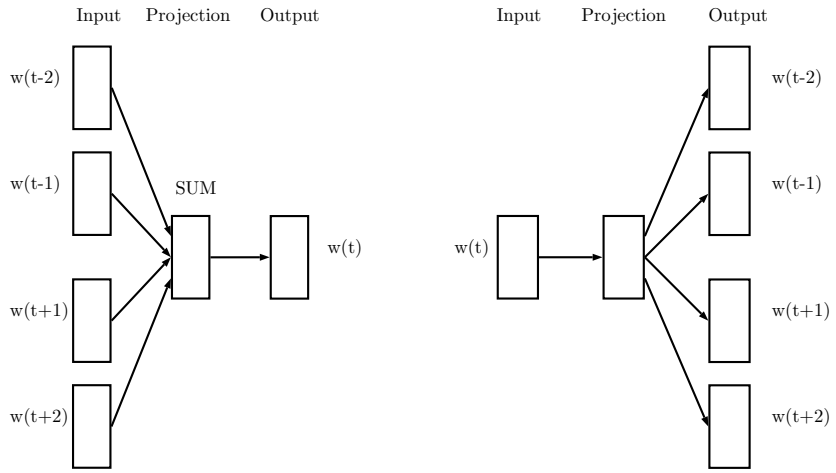


Figure 2.1: word2vec architectures: (*left*) CBOW predicts the current word based on the context of this word; (*right*) Skip-gram predicts surrounding words given the current word

This architecture achieves competitive performance on a number of NLP tasks (word similarity, relational analogy) and is used as a baseline to compare with.

Skip-gram model is considered to perform better for infrequent words[1](Mikolov et al., 2013b), therefore we choose this model as a basis for our experiments.

2.3.2 Association measurement schemes

Association measurement schemes decide which association function is used in the creation of distributional representations.

The two schemes described in this work employ functions that differ in the information they provide about the target word: the BINARY scheme concentrates on whether the target word and context words appear in the same window in the corpus; the PPMI scheme measures a positive pointwise mutual information (PPMI) – a degree of association between the target and context words.

BINARY. Let $v_1, \ldots, v_n$ be the vocabulary of context words. In BINARY scheme, every entry $1 \leq i \leq n$ in the distributional vector of a target word w is set to 1 iff w and v_i co-occur at a distance of at most ten words in the corpus and to 0 otherwise.

PPMI. As an alternative to binary $\{0, 1\}$ values, PPMI can be used:

$$PMI(w, v_i) = \log \frac{P(w,v_i)}{P(w)P(v_i)},$$

[1] `code.google.com/p/word2vec/`

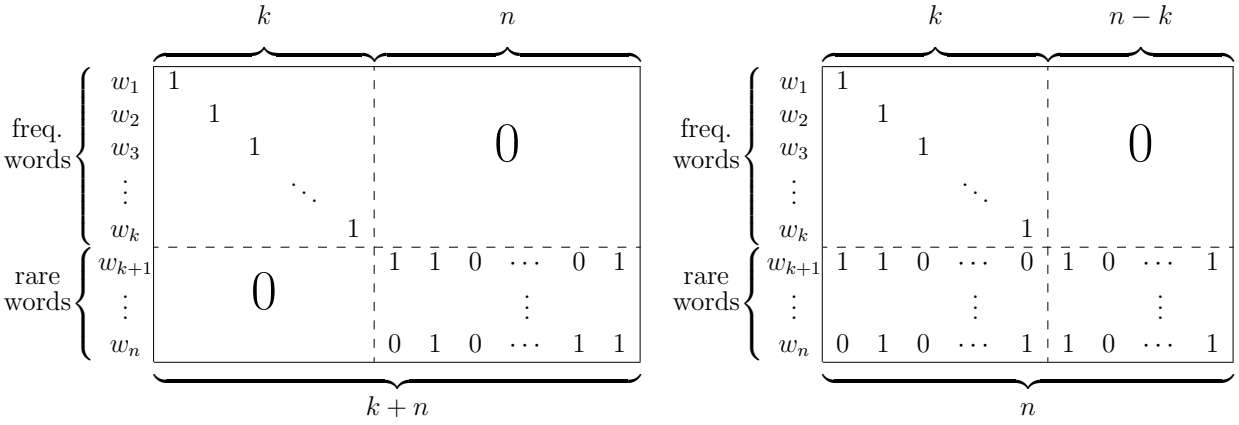


Figure 2.2: One-hot vectors of frequent words and distributional vectors of rare words are separate in *separate initialization* (left) and overlap in *mixed initialization* (right). This example is for BINARY weighting.

$$PPMI(w, v_i) = \begin{cases} 0, & PMI(w, v_i) < 0, \\ PMI(w, v_i), & \text{otherwise}, \end{cases}$$

where $P(w, v_i)$ is the probability to observe a target word w in the window of 10 words to the left and 10 words to the right with respect to a context word v_i, and $P(w)$ and $P(v_i)$ are the probabilities to observe a word w and a context word v_i in the corpus respectively.

Pointwise mutual information (PMI) is a measure of association used in information theory and statistics. It was first used in the distributional similarity area by Church and Hanks (1990) and popularized by Turney (2001). The PPMI variant – positive PMI – was known since (Niwa and Nitta, 1994a), and is a common association measure used in NLP [Bullinaria and Levy (2007), Baroni et al. (2014), S and Kaimal (2012)].

In PPMI scheme, frequent words receive one-hot representations and rare words receive PPMI representations: entry $1 \leq i \leq n$ in the distributional vector of a target word w is set to the PPMI of w and v_i. In order to keep all values in the initialization matrix in the same range, we rescale values in the PPMI vectors to ensure they are in $[0, 1]$, dividing them by the maximum value from the initial vectors of all words:

$$PPMI_{rescaled}(w, v_i) = \frac{PPMI(w, v_i)}{\max_{k,j} PPMI(w_k, v_j)}.$$

As all PPMI values for following experiments were rescaled, we will refer to them as "PPMI" for simplicity.

2.3.3 Combination schemes

Created distributional vectors for rare words are then combined with one-hot vectors for frequent words in two different manners: separate and mixed (see Figure 2.2). Recall that n is the dimensionality of the distributional vectors. Let k be the number of words with frequency $> \theta$, where the frequency threshold θ is a parameter.

SEPARATE. In separate initialization, the input representation for a word is the concatenation of a k-dimensional vector and an n-dimensional vector. For a word with frequency $> \theta$, the k-dimensional vector is a one-hot vector and the n-dimensional vector is zero vector. For a word with frequency $\leq \theta$, the k-dimensional vector is a zero vector and the n-dimensional vector is its distributional vector.

MIXED. In mixed initialization, the input representation for a word is an n-dimensional vector: a one-hot vector for a word with frequency $> \theta$ and a distributional vector for a word with frequency $\leq \theta$.

In summary, separate initialization uses separate representation spaces for frequent words (one-hot space) and rare words (distributional space). Mixed initialization uses the same representation space for all words; and rare words share weights with the frequent words that they co-occur with.

2.4 Experimental setup

2.4.1 Training corpus

For training, we use a corpus created by Baroni et al. (2009): a concatenation of ukWaC and WaCkypedia[2]. ukWaC is a 2 billion word corpus constructed from the Web limiting the crawl to the `.uk` domain and using medium-frequency words from the British National Corpus (BNC) as seeds. WaCkypedia is a 2009 dump of the English Wikipedia (about 800 million tokens), cleaned using the Wikipedia extractor. The corpus is preprocessed based on the procedure described in (Turian et al., 2010): we remove sentences that are less than 90% lowercase; lowercase; replace URLs, email addresses and digits with special tokens; tokenize with (Schmid, 2000); replace words of frequency 1 with `<unk>`; and add end-of-sentence tokens. After preprocessing, the corpus contains 2.4 billion tokens and 2.7 million word types.

In our experiments, target vocabulary and context vocabulary consist of the set of unique words from the training corpus.

2.4.2 Evaluation task

To estimate the quality of the trained word embeddings, the *word similarity judgment task* is employed. The task setting consists of a similarity data set(s) that contains pairs of words together with similarity rating. The tested system then tries to predict the similarity rating given pairs of words from the data set. The Spearman's correlation coefficient between provided and predicted ratings is measured, and a high correlation score is treated as evidence of high quality of the representations created by the system.

[2] `wacky.sslmit.unibo.it`

	RG	MC	MEN	WS	RW	SL
# pairs	65	30	3000	353	2034	999
# words	48	39	751	437	2942	1028

Table 2.1: Number of words and pairs in the six similarity data sets used for evaluation.

2.4.3 Evaluation data sets

To evaluate trained word embeddings, we use six word similarity judgment data sets:
RG (Rubenstein and Goodenough, 1965) data set contains 65 word pairs with one score per pair on a scale from 0 to 4. These scores are the means of judgments made by 51 subjects on the degree to which paired words are synonymous.

Ex.: {bird, cock} = 2.63; {graveyard, madhouse} = 0.44.

MC (Miller and Charles, 1991) data set is a 30 pairs subset of RG data set, rated by 38 undergraduate subjects. The pairs of nouns were chosen to cover different levels of similarity and were rated on a scale from 0 to 4. The average rating was reported.

Ex.: {journey, voyage} = 3.84; {graveyard, forest} = 0.84.

MEN[3] was introduced by (Bruni et al., 2012) to test multimodal systems. It contains 3,000 word pairs with semantic relatedness ratings provided by Amazon Mechanical Turk and normalized to $[0, 50]$. These words were randomly selected from combination of ukWaC and WaCkypedia corpora with occurring frequency ≥ 700.

Ex.: {dance, dancers} = 49; {bikini, pizza} = 1.

Word**S**im353 (WS)[4] (Finkelstein et al., 2001) data set contains 353 pairs of words that were rated by 13 subjects on an 11-point scale to assess words attributional similarity.

Ex.: {professor, doctor} = 6.62; {professor, cucumber} = 0.31.

The Stanford **R**are **W**ord (RW)[5] (Luong et al., 2013) data set contains 2034 pairs of rare words. Word candidates, found in Wikipedia with frequencies up to 10000, were selected randomly from 5 frequency bins and paired with interesting words picked from WordNet. Then created pairs were rated by 10 humans on a scale of [0, 10].

Ex.: {casteless, unwanted} = 7.50; {radiators, beginning} = 0.

Sim**L**ex-999 (SL)[6] is a data set introduced by (Hill et al., 2015) to focus on capturing similarity, rather than relatedness or association. It contains 999 pairs: 666 noun-noun, 222 verb-verb and 111 adjective-adjective pairs. The scores were obtained from 500 native English speakers, and lay on a scale of [0, 10].

Ex.: {cow, cattle} = 9.52; {portray, notify} = 0.78; {guilty, ashamed} = 6.38.

Data sets statistics are presented in the Table 2.1.

16 pairs in RW that contain at least one word that is not covered by the corpus vocabulary were excluded from the evaluation.

[3] clic.cimec.unitn.it/~elia.bruni/MEN
[4] alfonseca.org/eng/research/wordsim353.html
[5] www-nlp.stanford.edu/~lmthang/morphoNLM/
[6] cl.cam.ac.uk/~fh295/simlex.html

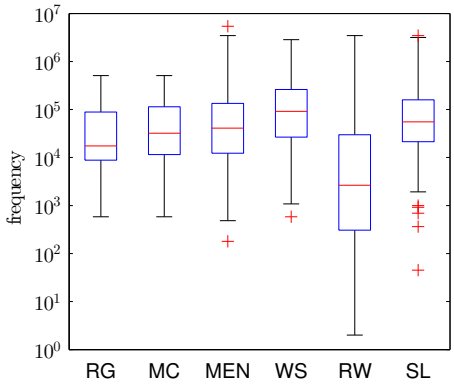


Figure 2.3: Corpus frequencies of the words in the similarity data sets.

	[1, 10]	[1, 20]	[1, 50]	[1, 100]
RG	0	0	0	0
MC	0	0	0	0
MEN	0	0	0	0
WS	0	0	0	0
RW	24	55	189	387
SL	0	0	1	1

Table 2.2: Number of words in frequency intervals in the similarity data sets.

2.4.4 Corpus downsampling

The goal of our work is to investigate the effect of using distributional initialization vs. one-hot initialization on the quality of embeddings of rare words. Therefore we decided to use words from word similarity data sets as a proxy to assess the quality of word embeddings learned for rare words. However, except for RW, the six data sets contain only a single word with frequency $\leq$100, all other words are more frequent (see Figure 2.3 and Table 2.2).

To address this issue, all words in the six data sets are artificially made rare by a procedure we called *corpus downsampling*. We randomly choose θ occurrences of the words from the similarity data sets in the corpus (if a word occurs less than θ times, all of it occurrences were chosen), and replace all other occurrences with a different token (e.g., "fire" is replaced with "*fire*"). This procedure ensures that all words from the six data sets are rare during the training and that our setup directly evaluates the impact of distributional initialization on rare words.

For example, in the training corpus words from the similarity data sets have the following frequencies: stadium: 68,115, quality: 631,619, indirectness: 64. After downsampling, all of those words will have frequency θ.

2.4.5 Use of the parameter θ

Note that we use θ for two different purposes:

(i) θ is the frequency threshold that determines which words in the vocabulary are classified as rare and which as frequent: in Figure 2.2 changing θ corresponds to moving the horizontal dashed line in separate and mixed initialization up and down;

(ii) θ is the parameter that determines how many occurrences of a word are left in the corpus after the corpus downsampling procedure.

We covary these two parameters in the experiments; e.g., we apply distributional initialization with $\theta = 20$ (to the words with frequency ≤ 20) to a corpus constructed to have $\theta = 20$ occurrences of words from similarity data sets. We do this to ensure that all evaluation words are rare words for the purpose of distributional initialization and so we can exploit all pairs in the evaluation data sets for evaluating the efficacy of our method for rare words.

We investigate changes in the quality of the learned word embeddings for four values of the frequency threshold $\theta \in \{10, 20, 50, 100\}$. One of the questions to asks here is whether the value of θ should depend on the size of the training corpus. Our intuition is that it is independent of corpus size. If a certain amount of information – corresponding to a certain number of contexts – is required to learn a meaningful representation of a word, then it should not matter whether that given number of contexts occurs in a small corpus or in a large corpus. However, if the contexts themselves contain many rare words (which is more likely in a small corpus), then a corpus size could be an important variable to take into account.

2.4.6 word2vec modification

We modified word2vec[7] (Mikolov et al., 2013a) to accommodate distributional initialization; to support distributional vectors at the input layer, we changed the implementation of activation functions and back propagation. Constructed distributional vector of words are provided to the model together with the vocabulary before the training starts. For training, we use the Skip-gram model, hierarchical softmax, set the size of the context window to 10 (10 words to the left and 10 to the right), min-count to 1 (train on all tokens), embedding size to 100, sampling rate to 10^{-3} and train models for one epoch (see Table 2.3).

2.4.7 Training regime

For each of the four values of the frequency threshold $\theta \in \{10, 20, 50, 100\}$, we train 5 word2vec models: one with one-hot initialization and one for each of the four combinations of association measurement (BINARY, PPMI) and distributional initialization (mixed, separate). In total, it results in $4 \times (1 + 2 \times 2) = 20$ models trained.

To get a reliable assessment of performance, we perform 5 training runs, starting with a different seed for the corpus downsampling and initialize the parameters of the models randomly. As results, averaged correlation values of the 5 runs are reported. Every model is trained for 1 epoch, that takes $\sim$3 hours to train on 23 CPU cores, 2.30GHz. Other word2vec parameters are reported in Table 2.3.

parameter	value	description
`cbow`	0	use Skip-gram architecture (for CBOW architecture, cbow=1)
`hs`	1	use hierarchical softmax
`size`	100	word embedding size
`window`	10	size of the context window (10 words to the left and 10 words to the right)
`min-count`	1	discard words that appear less than `<int>` times
`sample`	0.001	Set threshold for occurrence of words. Those that appear with higher frequency in the training data will be randomly down-sampled; default is 0.001, useful range is (0, 0.00001)
`negative`	0	do not use negative examples
`iter`	1	number of training epochs
`alpha`	0.025	initial learning rate; default value for Skip-gram model
`threads`	23	use 23 threads to train a model

Table 2.3: Parameter values used for word2vec training.

2.5 Experimental results and discussion

Table 2.4 shows experimental results, averaged over 5 runs. The evaluation measure is Spearman's correlation $\times 100$ between human and machine-generated pair similarity judgments.

2.5.1 Frequency threshold θ

The main result is that for $\theta \in \{10, 20\}$ *distributional initialization is better than one-hot initialization* (see bold numbers): compare lines 1&5 with line 9; and lines 2&6 with line 10. This is true for both mixed and separate initialization, with the exception of WS, for which mixed (column G) is better in only 1 (line 5) of 4 cases.

Looking only at results for $\theta \in \{10, 20\}$, 18 of 24 improvements are significant[8] for mixed initialization and 16 of 24 improvements are significant for separate initialization (lines 1&5 vs 9 and lines 2&6 vs 10).

For $\theta \in \{50, 100\}$, mixed initialization does well for RG, MC and SL, but the gap between mixed and one-hot initializations is generally smaller for these larger values of θ; e.g., the difference is larger than 9 for $\theta = 10$ (A1&A5 vs A/B9, C1&C5 vs C/D9, K1&K5 vs K/L9) and less than 9 for $\theta = 100$ (A4&A8 vs A/B12, C4&C8 vs C/D12, K4&K8 vs K/L12) for these three data sets.

Recall that each value of θ effectively results in a different training corpus – a training corpus in which the number of occurrences of the words in the evaluation data sets has been reduced to $\leq \theta$ (cf. Section 2.4.5).

Our results indicate that distributional initialization is beneficial for very rare words

[7] `code.google.com/p/word2vec`

[8] Two-sample t-test, two-tailed, assuming equal variance, $p < .05$

		A	B	C	D	E	F	G	H	I	J	K	L
		RG		MC		MEN		WS		RW		SL	
	θ	mixed	sep	mixed	sep	mixed	sep	mixed	sep	mixed	sep	mixed	sep
1	10	***56.54**	**47.06**	**35.96**	**32.10**	***43.76**	***45.56**	34.21	***40.93**	***24.81**	**20.85**	***18.30**	***13.76**
2	20	***59.08**	**45.31**	***46.66**	**35.22**	**52.05**	***52.38**	41.44	**47.53**	***29.48**	**26.93**	***20.85**	***16.86**
3	50	***63.20**	51.07	***52.35**	37.45	58.21	53.80	43.14	44.88	31.32	29.16	***24.19**	***22.45**
4	100	**68.33**	52.50	**61.70**	35.94	61.69	55.23	48.25	44.89	33.29	30.22	***26.74**	**24.66**
		BINARY											
5	10	***56.87**	***51.94**	***37.31**	***46.52**	***48.05**	***50.49**	**38.41**	***47.54**	***25.53**	**23.12**	***19.70**	***15.59**
6	20	***59.08**	***50.32**	***47.51**	***45.17**	***54.88**	***56.42**	43.31	***53.19**	***29.78**	***28.51**	***21.84**	***19.23**
7	50	***64.90**	***64.36**	***55.27**	***56.75**	**60.51**	**61.04**	45.76	**55.55**	32.05	30.25	***25.11**	***21.60**
8	100	**71.08**	58.37	**68.14**	52.33	63.05	60.74	48.66	55.49	33.25	30.49	***27.13**	22.60
		PPMI											
9	10	38.93		16.67		40.70		35.17		20.69		8.97	
10	20	42.17		25.21		50.21		43.74		26.58		13.62	
11	50	56.01		42.35		60.22		54.10		32.16		20.01	
12	100	67.47		61.33		65.14		59.87		35.19		24.06	
		one-hot											

Table 2.4: Spearman's correlation coefficients ×100 between human and embedding-based similarity judgments, averaged over 5 runs. Distributional initialization correlations that are higher (resp. significantly higher, $p < .05$) than corresponding one-hot correlations are set in **bold** (resp. marked *).

– those that occur no more than 20 times in the corpus. Our results for medium rare words – those that occur between 50 and 100 times – are less clear: either there are no improvements or improvements are small.

Thus, our recommendation is to use $\theta = 20$.

2.5.2 Scalability

The time complexity of the basic version of word2vec is $O(ECWD \log V)$ (Mikolov et al., 2013a) where E is the number of epochs, C is the corpus size, W is the context window size, D is the number of dimensions of the embedding space, and V is the vocabulary size. Distributional initialization adds a term I, the average number of entries in the distributional vectors, so that time complexity increases to $O(ECWD(\log V + I))$. For rare words, I is small, so that there is no big difference in efficiency between one-hot initialization and distributional initialization of word2vec. However, for frequent words I would be large, so that distributional initialization may not be scalable in that case. So even if our experiments had shown that distributional initialization helps for both rare and frequent words, scalability would be an argument for only using it for rare words.

2.5.3 Binary vs. PPMI

PPMI scheme is almost always better than BINARY, with three exceptions (I8, L7, L8) where the difference between the two is small and not significant. The probable explanation is that as the PPMI values are real values in the $[0, 1]$ interval, that allow them to convey detailed, graded information about the strength of association between two words. In contrast, the values in the BINARY scheme are from $\{0, 1\}$ set: that allow them only to indicate whether there was any instance of co-occurrence at all – without considering frequency of co-occurrence and without normalizing for base frequencies.

2.5.4 Mixed vs. Separate

From the 48 pairs of mixed/separate models, where all other parameters are the same, mixed models outperform separate ones in 34 cases, significantly so in 28. There are also 7 cases where separate models are significantly better than mixed. We attribute the better performance of the mixed models to the weights rare and frequent words share. Information about word distribution (as it is represented in the separate model) is not enough on its own; what seems to be more helpful are the *interconnections between frequent and rare words* that mixed initialization provides.

Moreover, mixed initialization is less variable – as a function of the parameter θ – and more predictable than separate initialization: performance for mixed initialization always goes up as θ increases, e.g., $56.54 \rightarrow 59.08 \rightarrow 63.20 \rightarrow 68.33$ (column A, lines 1–4). In contrast, separate initialization performance often decreases, e.g., from 47.06 to 45.31 (column B, lines 1–2) when θ is increased. Since more information (more occurrences of the words that similarity judgments are computed for) should generally not have a

negative effect on performance, the only explanation is that separate initialization is more variable than mixed and that this variability sometimes results in decreased performance. Figure 2.2 explains this difference between the two initializations: in mixed initialization (right panel), rare words are tied to frequent words, so their representations are smoothed by representations learned for frequent words. In separate initialization (left panel), no such links to frequent words exist, resulting in higher variability.

Because of its lower variability with respect to parameter θ, our experiments suggest that mixed initialization is a better choice than separate initialization.

2.5.5 One-hot vs. Distributional initialization

Our experiments show that distributional representation is helpful for rare words. It is difficult for one-hot initialization to learn good embeddings for such words, based on only a small number of contexts in the corpus. In such cases, distributional initialization makes the learning task easier since in addition to the *contexts of the rare word*, the learner now also has access to the *global distribution of the rare word* and can take advantage of weight sharing with other words that have similar distributional representations to smooth embeddings systematically.

Thus, distributional initialization is a form of smoothing: the embedding of a rare word is tied to the embeddings of other words via the links shown in Figure 2.2: the 1s in the lower "rare words" part of the illustrations for separate and mixed initialization. As is true for smoothing in general, parameter estimates for frequent events benefit less from smoothing or can even deteriorate. In contrast, smoothing is essential for rare events. Where the boundary lies between rare and frequent events depends on the specifics of the problem and the smoothing method used, and is usually an empirical question. Our results indicate that that boundary lies somewhere between 20 and 50 in our setting.

One of the intrinsic problems that our approach faces in its base design is that if a word is rare, its distributional vector will be sparse and less informative, which does not guarantee to be a good starting point for a NN trainer. This is true and suggests that it may not be possible to learn a very high-quality representation for a rare word. But what we want to show is that using the same word2vec training procedure, *better* representations can be learned. Our explanation for obtained positive experimental results is that distributional initialization implements a form of smoothing, that helps in case of rare events.

2.5.6 Variance of results

Table 2.4 reports the averages of five runs. The variance of results was quite high for low-performing models. For higher performing models – those with values ≥ 40 – the ratio of standard deviation divided by mean ranged from .005 to .29 (see Table 2.5). The median was .044. While the variance from run to run is quite high for low-performing models and for a few high-performing models, the significance test takes this into account, so that the relatively high variability does not undermine our results.

			A	B	C	D	E	F	G	H	I	J	K	L
			RG		MC		MEN		WS		RW		SL	
		θ	mixed	sep	mixed	sep	mixed	sep	mixed	sep	mixed	sep	mixed	sep
1	BINARY	10	.054	.228	.283	.407	.034	.023	.129	.082	.027	.080	.103	.146
2		20	.067	.026	.290	.448	.039	.032	.138	.074	.037	.012	.051	.091
3		50	.060	.066	.102	.230	.014	.011	.036	.034	.020	.026	.055	.069
4		100	.042	.065	.097	.212	.013	.005	.014	.052	.016	.039	.036	.057
5	PPMI	10	.041	.141	.279	.095	.027	.032	.094	.047	.033	.090	.076	.131
6		20	.056	.045	.259	.217	.023	.013	.123	.093	.026	.023	.074	.072
7		50	.065	.067	.106	.206	.016	.010	.031	.029	.024	.026	.066	.044
8		100	.035	.056	.093	.247	.012	.008	.030	.035	.006	.023	.021	.103
9	one-hot	10	.166		.968		.036		.091		.054		.241	
10		20	.069		.500		.020		.115		.023		.104	
11		50	.040		.145		.015		.017		.021		.056	
12		100	.043		.048		.011		.020		.021		.066	

Table 2.5: Standard deviations of Spearman's coefficients divided by their mean values for results reported in Table 2.4.

			A	B	C	D	E	F	G	H	I	J	K	L
			RG		MC		MEN		WS		RW		SL	
		θ	mixed	sep	mixed	sep	mixed	sep	mixed	sep	mixed	sep	mixed	sep
1	BINARY	10	***54.61**	**43.34**	**36.29**	**34.60**	***44.43**	***45.92**	33.90	***40.14**	***25.06**	**20.31**	***18.03**	***13.77**
2		20	***60.36**	***46.92**	***48.33**	***35.14**	**51.93**	***52.15**	41.08	***47.66**	***29.15**	***27.07**	***20.94**	***17.21**
3		50	***63.61**	50.78	***53.52**	36.58	58.25	53.83	44.14	45.76	31.33	29.19	***24.20**	***22.10**
4		100	**68.20**	52.58	**60.95**	34.05	61.97	55.28	48.04	45.18	32.87	29.66	***26.27**	**24.25**
5	PPMI	10	***55.11**	***50.33**	***37.36**	***47.29**	***48.68**	***50.58**	**37.78**	***45.88**	***25.72**	***22.99**	***19.50**	***15.75**
6		20	***59.82**	***54.03**	***49.88**	***50.61**	***55.14**	***57.25**	43.41	***52.80**	***29.57**	***27.94**	***21.91**	***18.47**
7		50	***65.69**	***63.37**	***58.15**	***59.14**	**60.57**	***61.34**	46.41	**55.23**	32.03	30.08	***25.01**	***21.59**
8		100	***70.44**	60.23	***66.17**	55.66	63.33	61.00	48.50	56.06	32.89	31.01	***27.02**	21.85
9	one-hot	10	34.72		23.83		40.01		35.61		20.18		8.40	
10		20	42.55		23.29		50.74		43.59		26.10		13.28	
11		50	59.00		46.23		60.42		54.34		32.40		20.04	
12		100	65.87		58.95		65.32		60.38		35.19		23.84	

Table 2.6: Spearman's correlation coefficients $\times 100$ between human and embedding-based similarity judgments, averaged over 10 runs. Correlations of models with distributional initialization that are higher (resp. significantly higher) than corresponding one-hot correlations are set in **bold** (resp. marked *).

To justify the performance stability, we run our experiment for 5 more times, resulting in 10 runs in total. Averaged results can be found in Table 2.6. The performance pattern does not change much: models with distributional initialization that performed better on 5 runs maintain their reputations, with some correlation values becoming significantly higher with comparison to Table 2.4 (line 2, columns B, D, H, J; columns A and C for line 8; column F, line 7; and column J, line 5). The gain of performance for these models can be explained with increase of differences between mean values of correlations of models with distributional and one-hot initializations.

2.5.7 Summary

Summing up, we have shown that distributional initialization improves the quality of word embeddings for rare words. Our recommendation is to use mixed initialization with PPMI values and the value $\theta = 20$ of the frequency threshold.

2.6 Related work

We would like to mention several lines in research attempts to improve quality of word representations. Such works can be assigned to several broad categories.

2.6.1 Incorporation of external knowledge into the input layer

One way to make use of existing information about a word is to represent it as a vector and concatenate this vector with an input vector of a word.

Alexandrescu and Kirchhoff (2006) employ *factors* – explicit word features – as a part of an initial word representation. Proposed factors consist of word shape features (affixes, capitalization, hyphenation, etc.) or other annotations (part-of-speech classes). Bian et al. (2014) construct such a vector with usage of part-of-speech tags, morphemes, word's entity vector. Qiu et al. (2014) propose to segment words into morphemes and insert them into the training vocabulary; then an embedding for a word is represented as a weighted sum of learned embeddings of the word itself and its morphemes.

2.6.2 Incorporation of external knowledge as an objective function

External knowledge about words can be also provided to a trainer in a form of an objective function or regularization term(s).

Relational and categorical knowledge, extracted from a knowledge graph, can be integrated into a training objective as two separate regularization functions as is shown by Xu et al. (2014). Yu and Dredze (2014) propose to add an objective function that learns word's entity type and its relations to a word2vec training objective. As a continuation of this work, Celikyilmaz et al. (2015) use an additional objective that takes care about

word's relations in a knowledge graph. Joint optimization of distributional (predict word given its context) and relational (distance in a WordNet(Miller, 1995) graph) objectives was proposed in a work (Fried and Duh, 2015). Huang et al. (2012) introduce local and global context via a joint training objective of a neural network language model. Global context here is represented as a sum of word embeddings of words from the document. Same spirit is shared by a work (Sun et al., 2015) that pays attention to a syntagmatic ("relate words that co-occur in the same text region") and paradigmatic ("relate words that occur in the similar contexts but may not co-occur in the text") relations of a word: they try to predict a word given its context and a vector representation of the whole phrase itself. In the work (Cotterell and Schütze, 2015) authors try to predict a feature vector of morphological tags of a target word as an extra objective incorporated in an LBL training.

2.6.3 Words representation as a composition of other words

Several works explore a word representation acquisition with more trust given to the distributional hypothesis: they make word representation heavily depend or been constructed from learned representations of other, similar to a target, words.

Niwa and Nitta (1994b) compare 2 different semantic representations for rare words: co-occurrence vectors gathered from a corpus and distance vectors built from dictionaries. Yogatama et al. (2015) use context to represent words and then learn word representations applying matrix factorization. Le et al. (2011) cluster rare words together, making representation of rare words close to each other. Celikyilmaz et al. (2015) propose to constrain the context of a word with the corresponding entity. Levy and Goldberg (2014a) include dependency-based context (surrounding words with their dependency labels) into embeddings learning with word2vec, in contrast to the traditional linear context. Cui et al. (2015) incorporate morphological knowledge into word2vec. Word embeddings of words that are morphologically similar to a target are combined together (their sum is weighted by a similarity score) in a parallel branch of a Skip-gram trainer.

2.6.4 Words representation as a composition of words parts

Another expected way to improve words embeddings that we would like to mention is to decompose word into subparts, e.g., morphemes, letters, obtain representations of these subparts and then build representation of the word as a composition of learned embeddings.

Botha and Blunsom (2014) propose to represent a word as a sum of embedding vectors of its surface morphemes. Use of RNN on word's morphemes to obtain word representation on the last output layer was proposed by (Luong et al., 2013). Salimbajevs and Strigins (2015) explore the sub-word approach, i.e., prefixes and endings, which are mostly common for all words, are split and treated as separate words. Ling et al. (2015b) represent words using a single embedding vector for each character. Word's representation is then obtained as an output of an LSTM model.

2.6.5 Improvement of the training regime

2.6.5.1 Initialization

Le et al. (2010) propose three schemes to address word embedding initialization. *Reinitialization* and *iterative reinitialization* use vectors from the prediction space of a model to initialize the context space during training: after the training converges or after each epoch respectively. *One-vector initialization* initializes all word embeddings with the same random vector to keep rare words close to each other.

2.6.5.2 Multitask learning

Ideas of training one network architecture for several different tasks (*multitask learning*) or reuse one architecture for another task (*transfer learning*) also found their places in DL research as ones that can help generalization. Collobert et al. (2011) show in their work that a single unified architecture can perform well for several tasks, only marginal improvements are obtained in comparison to separate architectures per task. Later Liu et al. (2015b) propose multi-task DNN for learning representations across query classification and ranking for web search tasks, showing strong positive results.

2.6.5.3 Embeddings pre-training

Greedy layer-wise pre-training or supervised fine-tuning of the unsupervised pre-trained system can also be seen as an attempt to improve quality of learned representations. This research has a long history started from the well-known works of (Hinton et al., 2006), (Bengio et al., 2007) for NLP.

2.6.6 Discussion

All introduced related approaches share the same goal: to learn better word representations. Despite of some resemblance with our approach, there are several very crucial differences that make our work interesting and more appealing to use. **First**, our approach does not require external knowledge as described in Section 2.6.1 and Section 2.6.2: neither linguistic (morphemes, part-of-speech tags, dependency parses) nor lexical (concept hierarchies, dictionaries) nor structural (knowledge bases). **Second**, our approach does not need computationally advanced theoretical solutions like a composition of several objective functions as mentioned in Section 2.6.2. **Third**, our approach does not need to employ any advanced neural network architectures like RNNs or LSTMs as mentioned in Section 2.6.4, avoiding training complications of long-dependency architectures. **Fourth**, our approach is directly concentrated on the quality of rare words representations, while in most works these words are excluded either from training or from evaluation, even very congenial to ours described in Section 2.6.3. **Fifth**, our training goal is to obtain *general* word embeddings; this is different to learning task-specific words embeddings, though can be seen as a

pre-processing step (as in multitasking or pre-training from Section 2.6.5). **Sixth**, initialization regimes from Section 2.6.5 are all more complex and less efficient than ours since the initial embedding is much denser than in our approach.

Though our approach is different from the discussed above, it can be easily enriched by proposed ideas: our distributional vectors can be extended with external knowledge; our result embeddings can serve as pre-trained for further tuning. Distributional information on the one hand and syntactic/semantic information on the other hand are likely to be complementary, so that its combination with our approach can lead to better representations for words of all frequencies – the main goal of the course of these works.

2.7 Conclusion

We have introduced distributional initialization of neural network architectures for learning better embeddings for rare words. The proposed approach consists of a combination of two types of association weights – BINARY and PPMI – with direct and indirect composition of distributional vectors with one-hot vectors – mixed and separate – to enhance initial representation for rare words.

Experimental results on a word similarity judgment task demonstrate that embeddings of rare words learned with distributional initialization perform better than embeddings learned with traditional one-hot initialization, especially for words with low frequency.

2.8 Future work

Our work is the first exploration of the utility of distributional representations as initialization for embedding learning algorithms like word2vec. There are a number of research questions we would like to investigate in the future.

First, we showed that distributional representation is beneficial for words with very low frequency ($\theta \in \{10, 20\}$). It was not beneficial in our experiments for more frequent words ($\theta \in \{50, 100\}$). A more extensive analysis of the factors that are responsible for the positive effect of distributional representation is in order.

Second, to simplify our experimental setup and make the number of runs manageable, we used the parameter θ both for corpus downsampling (only θ occurrences of a particular word were left in the corpus) and as the separator between rare words that are distributionally initialized and frequent words that are not. It remains to be investigated whether there are interactions between these two properties of our model, e.g., a high value of rare-frequent separator may work well for words whose corpus frequency is much smaller than the separator.

Third, while we have shown that distributional initialization improves the quality of representations of rare words, we did not investigate whether distributional initialization for rare words has any adverse effect on the quality of representations of frequent words for which one-hot initialization is applied. Since rare and frequent words are linked in the

mixed model, this possibility cannot be dismissed and we plan to investigate it in future work.

Chapter 3

Language Modeling

3.1 Summary

We conduct another set of experiments to test the proposed approach – initialization of neural networks with distributional vectors – with more practical goals. As a task, we choose language modeling, and the results are measured in perplexity points. Different distributional representations of the input vocabulary were explored, and minor improvements over models with traditional one-hot initialization were achieved.

3.2 Introduction

Language modeling is a task of predicting probabilities of given sequences of words. This makes language models (LMs) suitable for independent Natural Language Processing (NLP) tasks such as morphological analysis (Morita et al., 2015), word segmentation (Mansur et al., 2013), sentence completion [Gubbins and Vlachos (2013), Mirowski and Vlachos (2015)], image captioning (Devlin et al., 2015a), spell checking (Chen et al., 2013), and also as parts of real-world NLP systems such as automatic speech recognition [Adel et al. (2013), Li and Fung (2014), Vu and Schultz (2014), Masumura et al. (2015)] and machine translation [Sperr et al. (2013), Auli and Gao (2014), Parikh et al. (2014), Murray and Chiang (2015), Luong et al. (2015), Baltescu and Blunsom (2015)].

For a long time, the most popular LMs were n-gram models that accumulate statistics of word sequences of length n from a given training corpus and then make predictions based on these statistics. In order to improve n-gram models performance in cases of rare and unseen events, different techniques were proposed, such as smoothing [Katz smoothing (Katz, 1987), Jelinek-Mercer smoothing (Jelinek and Mercer, 1980), Kneser-Ney smoothing (Ney et al., 1994)], skipping models (Rosenfeld, 1994), clustering (Brown et al., 1992), caching (Kuhn, 1988), sentence mixture (Iyer et al., 1994) – together with combinations of different techniques, e.g., backing-off to the lower order n-grams, model combinations and interpolations (for more details see Goodman (2001)).

Another class of LMs that becomes very popular is Neural Network Language Models

(NNLMs). One of the most influential works is (Bengio et al., 2003), where they proposed to use a feed-forward neural network architecture that at the input layer employs continuous representations of context words – word embeddings, and outputs a probability distribution over the prediction vocabulary.

Recently, the class of NNLMs was extended with Neural Networks (NNs) of different nature: feed-forward NNs and continuous-space LMs [Schwenk (2007), Mansur et al. (2013), Vaswani et al. (2013), Wang et al. (2014), Devlin et al. (2015b), Murray and Chiang (2015)], Recurrent Neural Networks (RNNs) [Adel et al. (2013), Rei (2015), Morita et al. (2015)] and LSTM (Kim et al., 2016), log-bilinear NNs (Botha and Blunsom, 2014), convolutional NNs (Devlin et al., 2015a). In these works, different enhancements of original architectures were investigated, such as number and combination of hidden layers and hidden units [Devlin et al. (2015b), Murray and Chiang (2015), Luong et al. (2015)], integration of POS tags at the input of NNs (Adel et al., 2013), models combinations (Devlin et al., 2015a), etc.

Our work is in the line of this research: we propose to enrich input layer of NNLMs with distributional information in order to improve the quality of the LM predictions.

3.2.1 Factored Language Models

Bilmes and Kirchhoff (2003) propose to represent words as a vector of k factors, where factors are representatives of characteristics of words, e.g., parts-of-speech, morphological classes, stems – Factored Language Model (FLM). Its extension with NNs – Factored Neural Language Models – was introduced by Alexandrescu and Kirchhoff (2006). The main challenge of FLMs is to find the suitable set of factors for word representations

In our work, we represent each word with a vector that emphasizes words with similar contexts. Thus, our approach can be seen as a FLM where factors are context words or contextually similar words.

3.2.2 Letter n-grams and Character-based language models

The idea of using not only word surroundings but also words themselves appeared to be fruitful in the research field.

Use of letter n-grams for words representation – *word hashing* – was introduced by Huang et al. (2013) as a technique that reduces dimensionality of words representation vectors and helps dealing with out-of-vocabulary words and spelling variations. Gao et al. (2014) propose to concatenate letter n-gram vectors to one-hot vectors of vocabulary words, and then use the result vector as an input representation of words for NN training. Sperr et al. (2013) suggest to use letter n-grams (n from 1 to 4) as an input to restricted Bolzmann machine for statistical machine translation.

Another way to employ the fine-grained information about words is to look on them in character-based manner: by means of character-based NNs. dos Santos and Zadrozny (2014) and dos Santos and Guimarães (2015) explore word embeddings together with embeddings for each character of the input word in order to solve part-of-speech tagging

and named entity recognition tasks. Kim et al. (2016) propose to create word representation combining embeddings for each character by means of convolutional NN, and then use this representation in LSTM.

In our work, we employ letter n-grams of words themselves and surrounding words in two ways: as one of suggested distributional representations and also as an intermediate representation for construction of words distributional representations.

3.2.3 Problem statement

As it was shown in Chapter 2, distributional initialization of NNs is a convenient way to improve quality of learned word embeddings for rare words. The question we would like to ask here is whether distributional initialization can help to improve the ability of language models to correctly predict the next word, especially when we change the representation of rare words only.

We train and explore several LMs with distributional initializations of words with a specific frequency or within a specified frequency range. To build distributional representations, we use positioned vectors with PPMI values and positioned vectors with PPMI counts of letter 3-grams of target and context words. We also study several input normalization schemes that help to keep in balance activation on the input layer of NN.

3.3 Methods

The framework of our experiments includes training of n-gram and neural LMs, their interpolation and evaluation of the obtained model. In the rest of this section, we are going to describe in detail the theoretical background of the new and existing components of the framework:

1. n-gram LM:
    - 1.1 n-gram LMs,
    - 1.2 Kneser-Ney smoothing for LMs,
    - 1.3 Modified Kneser-Ney smoothing for LMs;
2. distributional representation of words;
3. NNLM:
    - 3.1 LBL and vLBL architectures,
    - 3.2 integration of the distributional representations at the input of NN,
    - 3.3 NN training;
4. interpolation of n-gram and neural LM;

5. evaluation of LMs.

Our contributions to this framework (points 2 and 3.2) is discussed in Section 3.3.2 and Section 3.3.3.3 : the proposed distributional representations and their integration into NNLM training.

3.3.1 n-gram language models

3.3.1.1 n-gram language models

The question that frequently arises in some NLP problems is how to estimate the probability of a given sequence of words. In machine translation, it is helpful to know which generated translation is more probable from the point of target language. In speech recognition, we would like to resolve the ambiguity of the next word given its acoustic representation and previous words. Basically, we would like to have access to a probability $p(w_1...w_n)$ or to a distribution over the next word $p(w_n|w_1...w_{n-1})$. The straightforward way to estimate this probability is to employ the chain rule:

$$p(w_1...w_n) = p(w_1)\ p(w_2|w_1)\ p(w_3|w_1w_2)\ ...\ p(w_n|w_1...w_{n-1}). \tag{3.1}$$

The probabilities in the right part of the equation can be estimated directly from a given text corpus; though, there are some complications. One is that given a huge corpus, the size of a model that stores all probabilities for all possible sequences of words observed in this corpus will be extremely huge. Another one is that, despite of the size of a corpus, there always going to be sequences of words that were never observed in it. This will lead to a zero value of some multipliers in Eq. 3.1, that will lead to a zero value as a result. This problem is caused by *data sparsity*, and there have been many research attempts to fight it.

To make probability estimations of sequences feasible, utilization of Markov chains was suggested (Shannon, 1948). The main idea of Markov chains of order n is to ignore longer sequences. Thus, for $n = 3$, Eq. 3.1 takes the following form:

$$p(w_1...w_n) = p(w_1)\ p(w_2|w_1)\ p(w_3|w_1w_2)\ p(w_4|w_2w_3)\ ...\ p(w_n|w_{n-2}w_{n-1}).$$

When this approach is applied to a sequences of words, the resulting probabilistic model is called *n-gram language model*.

Though such an n-gram model has a smaller size since only sequences up to length n are stored and rate of unseen sequences reduces with the size limitation, none of the aforementioned issues are solved completely.

In order to avoid zero probabilities of unseen or rare events, various smoothing techniques were proposed, e.g., backing off to lower-order models, skip gram models, interpolation and mixture of different models (see for example Goodman (2001) for more detailed description).

We would like to illustrate smoothing techniques with an example of a simple back-off 3-gram model that uses probability of 2-gram model in case the 3-gram has not been

seen in the corpus (α and β here are determined probabilities of trigrams and bigrams respectively, normalized in order to form a probability distribution):

$$p(w_k|w_{k-2}w_{k-1}) = \begin{cases} \alpha(w_k|w_{k-2}w_{k-1}), & \text{if } count(w_{k-2}w_{k-1}w_k) > 0, \\ \beta(w_k|w_{k-1}), & \text{if } count(w_{k-2}w_{k-1}w_k) = 0, \end{cases}$$

In the next sections we would like to cast a light upon the most popular and successful smoothing technique: Kneser-Ney smoothing and its modified form.

3.3.1.2 Kneser-Ney smoothing for language models

The problem with traditional back off smoothing technique is that while backing off to the lower-order models, important information about co-occurring events is lost. This leads to a bias towards words that have a limited number of preceding words. Kneser and Ney (1995) provide an example of word *dollars* that occurs frequently after numbers and country names, but is rarely seen in other contexts; so in case of backing off to a regular unigram probability, *p(dollars)* will be relatively high, while the actual probability of $(x, dollar)$ is 0 for most of the bigrams (as they will not be seen in the corpus).

In the same paper, authors also proposed an improved version of back-off smoothing, where backing-off probability distribution is different from normal probability distribution and relies on a number of distinct histories of a particular word. Their final suggestion has form:

$$p(w|h) = \begin{cases} \alpha(w|h), & \text{if } N(h,w) > 0, \\ \gamma(h)\beta(w|\hat{h}), & \text{if } N(h,w) = 0, \end{cases}$$

where w is a word, h is a history, $\hat{h}$ is a back-off history (e.g., in case of n-gram model, $\hat{h}$ is $(n-2)$-gram); α is a reliable estimate of next word probability, γ is determined completely by α and β, and distribution β is defined as:

$$\begin{aligned} &\beta(w|\hat{h}) = \tfrac{N_+(\bullet,\hat{h},w)}{N_+(\bullet,\hat{h},\bullet)}, \\ &where \quad N_+(\bullet w) = |\{w_i : N(w_i,w) > 0\}|, \\ &\qquad\qquad N_+(\bullet\bullet) = \sum_w N_+(\bullet w). \end{aligned}$$

Here $N(.)$ is a counted number of n-grams.

3.3.1.3 Modified Kneser-Ney smoothing of n-gram model

The most popular n-gram model nowadays is modified Kneser-Ney smoothing LM, proposed by Chen and Goodman (1999). They suggest to use 3 different discounting parameters in Kneser-Ney smoothing: D_1, D_2, and D_{3+} for n-grams with one, two and three or more occurrences respectively. Formally, the probabilities are estimated with an equation:

$$p_{MKN}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i}))}{\sum_{w_i} c(w_{i-n+1}^{i})} + \gamma(w_{i-n+1}^{i-1}) p_{MKN}(w_i|w_{i-n+2}^{i-1}).$$

The main difference with Kneser-Ney smoothing is employment of different discounts D depending on the count of context n-gram:

$$D(c) = \begin{cases} 0, & \text{if } c = 0, \\ D_1, & \text{if } c = 1, \\ D_2, & \text{if } c = 2, \\ D_{3+}, & \text{if } c \geq 3. \end{cases}$$

The function of γ is to make the probabilities sum to 1, i.e.

$$\gamma(w_{i-n+1}^{i-1}) = \frac{D_1 N_1(w_{i-n+1}^{i-1}\bullet) + D_2 N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+} N_{3+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i-1})},$$

where $N_1(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1} w_i) = 1\}|$ – the number of words that appear exactly once after w_{i-n+1}^{i-1}. N_2 and N_{3+} are defined analogously.

Chen and Goodman (1999) also propose the estimations of $D(c)$:

$$\begin{aligned} D_0 &= 0, \\ D_1 &= 1 - 2Y\tfrac{n_2}{n_1}, \\ D_2 &= 2 - 3Y\tfrac{n_3}{n_2}, \\ D_{3+} &= 3 - 4Y\tfrac{n_4}{n_3}, \end{aligned}$$

where $Y = \frac{n_2}{n_1+2n_2}$; n_i is the total number of n-grams with exactly i counts.

In our experiments, we use n-gram LM with modified Kneser-Ney smoothing.

3.3.2 Distributional representations of words

In this section, we describe how we build distributional representations of words from a given corpus.

In our experiments, we assign every vocabulary word a fixed vector that is built based on the information from a training corpus. Frequent words are always represented with one-hot vectors. For the subset of rare words, several different distributional representations are proposed: with different combination schemes, association functions, and normalization schemes. We decide which words receive one-hot representation (OHR words) and which receive distributional representation (DR words) based on their frequency in the training corpus. The choice of the parameter θ that is responsible for frequency ranges is discussed in Section 3.4.4.

For short, further we refer to an LM initialized with particular distributional representations as "[scheme name] model", e.g. *separate model*, *CN model*, etc.

3.3.2.1 Combination schemes

A combination scheme decides how to combine distributional and one-hot representations for different words. Recall that n is the dimensionality of the distributional vectors. Let k be the number of words with frequency $\notin \theta$ (if θ is an interval) or $\neq \theta$ (if θ is a constant), where the frequency range θ is a parameter.

ONEHOT. This is the baseline initialization that assigns all-but-one zeros to a distributional representation of every target word w_i, putting 1 to a word's index position i:

$$\begin{array}{rl} w_i = & [0 \ldots \quad 0 \quad 1 \quad 0 \quad \ldots 0] \\ & 1 \ \ldots \ (i-1) \ i \ (i+1) \ \ldots \ k \end{array}$$

The matrix that contains one-hot representations of the vocabulary words is diagonal. This representation is equivalent of usage of the word's index to retrieve embedding vector from a look-up table.

ONLY. In ONLY models, distributional representation is applied *only* to words with a particular frequency (θ is a constant). All other words receive one-hot representations. Hereafter we will refer to models with this scheme as "θ ONLY" or "θonly", where θ is specified in the respective experiment.

SEPARATE. In separate scheme, the input representation for a word is the concatenation of a k-dimensional vector and an n-dimensional vector. For a word with frequency $\notin \theta / \neq \theta$, the k-dimensional vector is a one-hot vector and the n-dimensional vector is zero vector. For a word with frequency $\in \theta / = \theta$, the k-dimensional vector is a zero vector and the n-dimensional vector is its distributional vector.

MIXED. In mixed scheme, the input representation for a word is an n-dimensional vector: a one-hot vector for a word with frequency $\notin \theta / \neq \theta$ and a distributional vector for a word with frequency $\in \theta / = \theta$.

In summary, separate scheme uses separate representation spaces for words with one-hot and words with distributional representations. Mixed initialization scheme uses the same representation space for all words; and DR words share weights with OHR words that they co-occur with. Schematically, mixed and separate representations are shown in Chapter 2, Figure 2.2.

3.3.2.2 Association measurement schemes

Association measurement schemes decide which association function is used in creation of distributional representations.

There are several association functions that we explore: co-occurrence in a window of a fixed size, cosine similarity of vectors that are built from context words: on positions {-2,

-1, 1, 2} with respect to a target word with PPMI values and on positions {-2, -1, 0, 1, 2} (target word included) with counts of letter 3-grams.

Co-occurrence in a window

The simplest association scheme is similar to a BINARY scheme described in Chapter 2. For a target word w, every entry $1 \leq i \leq n$ in the distributional vector is set to 1 iff w and v_i co-occur at a distance of at most C words in the corpus, and to 0 otherwise. $\{v_1, \ldots, v_n\}$ is the vocabulary of context words. In experiments, we set C = 10.

Further we refer to LMs trained with such distributional representations as "binary models" since the distributional vectors contain values equal either 0 or 1.

Positioned PMI/PPMI

For every target word, we consider words at a distance of at most 2 (two words to the left and two words to the right, excluding target word). We build count vectors for each of these positions; replace raw counts in the vectors with PPMI values; and concatenate positioned vectors. The definition of positive pointwise mutual information (PPMI) was introduced in Section 2.3.2, Chapter 2, here we repeat it for the reader's convenience.

PPMI values are calculated for a target word w and a context word v_i with respect of its relative position to a target word $pos \in \{-2, -1, 1, 2\}$:

$$PPMI(w, v_{i_{pos}}) = \max\{\log \frac{P(w, v_{i_{pos}})}{P(w)P(v_{i_{pos}})}, 0\}$$

where $P(w, v_{i_{pos}})$ is the probability to observe a target word w with a context word v_i on a relative position pos, and $P(w)$ and $P(v_{i_{pos}})$ are the probabilities to observe a word w and a context word v_i on a position pos in the corpus respectively.

For separate representations. For all words we would like to give a distributional initialization to, we concatenate 4 derived positioned PPMI vectors. We also explore normalization schemes where PPMI values are divided by a constant value or distributional vectors are normalized (normalization schemes are described in Section 3.3.2.3). In the first experiment with separate models (Section 3.5.4), we use only these concatenated vectors as representations of words. In other experiments, we enrich positioned vectors with diagonal matrix that precedes the distributional vector (see Figure 3.2, last row and Figure 3.3, last two columns). All numbers on the diagonal of the derived matrix are set to 1.

For mixed representations. For each vocabulary word, we concatenate 4 derived positioned PPMI vectors. We then normalize the obtained vector dividing its every element by the sum of all elements in the vector. To build distributional vector, we employ cosine similarity of this PPMI vector for each DR word and the PPMI vectors of other words as described below.

We would like to determine similarity threshold α in a way that average number of similar words for each DR word would be equal to a given parameter NUMBER_OF_ONES

(NOO/noo). To determine the similarity threshold α, we apply the steps described in Algorithm 1. We pick randomly N DR words, measure similarity scores between these words and all context words, sort context words according to the similarity scores, then set α to the similarity score of the NOO*N element. Afterward, we create distributional representation for each DR word, where a value in the distributional vector is set to SIMILARITY_WEIGHT if the similarity of the DR word and a context word is not lower than α, and to 0 otherwise.

Data:
pre-computed similarity matrix of $|DR\_words| \times |context\_words|$;
average number of non-zero values in the distributional vectors NUMBER_OF_ONES;
sample size N;
weight of non-diagonal elements SIMILARITY_WEIGHT;

sampled_words ← sample N random DR words;
merged_list ← {} ; `// list of word-similarity score pairs`
foreach *word* **in** *sampled_words* **do**
  foreach *context_word* **in** *context_words* **do**
    populate *merged_list* with a pair (*context_word*, similarity(*word*, *context_word*));
  end
end
sorted_merged_list ← sort *merged_list* based on similarity values in descending order;

`// set similarity threshold` *alpha* `to the similarity of the pair with index NUMBER_OF_ONES·N:`
alpha ← *sorted_merged_list*[NUMBER_OF_ONES · N].get_similarity_score();

`// create distributional representations`
foreach *DR_word* **and** *context_word* **do** create distributional representation:
  if similarity*(DR_word, context_word)* $\geq$ *alpha* **then**
    distributional_representations[*DR_word*][*context_word*] =
     SIMILARITY_WEIGHT;
  else
    distributional_representations[*DR_word*][*context_word*] = 0;
  end
end
Result: matrix *distributional_representations* with distributional representations of DR words.

Algorithm 1: Deriving distributional representation for words through the similarity of their positioned PPMI vectors.

In our experiments, we set NUMBER_OF_ONES to 2, 10 or 30, and set N=100. SIMILARITY_WEIGHT decides on which values are used in distributional representations; this parameter corresponds to W normalization scheme (see Section 3.3.2.3). On diagonal, 1 is put (words are similar to themselves); for non-diagonal elements, we usually use 0.1. Other values of W in range {1, 0.5, 0.01, 0.001} are also investigated.

Further we refer to LMs trained with such distributional representations as "positioned PPMI models".

Letter 3-grams
Letter 3-gram scheme is very similar to positioned PMI/PPMI, described in the Section 3.3.2.2. The only differences are:

- for similarity comparison, as count vectors we use vectors of co-occurrences of target words with letter 3-grams of context words;
- we also use letter 3-gram representation of a target word itself in a positioned vector.

Every time target and context word are at a distance ≤ 2, counts of the target word and all letter 3-grams of context words are increased by 1. E.g., consider as a target word *money* in a phrase "*the top money funds are*". Then in the co-occurrence matrix values that lay in a row that correspond to the word *money* and columns that correspond to letter 3-grams $\#th_{-2}$, the_{-2}, $he\#_{-2}$, $\#to_{-1}$, top_{-1}, $op\#_{-1}$, $\#mo_0$, mon_0, one_0, ney_0, $ey\#_0$, $\#fu_1$, fun_1, und_1, nds_1, $ds\#_1$, $\#ar_2$, are_2, $re\#_2$ will be increased by one.

Formally, for every target word, we consider words at a distance ≤ 2 (two words to the left and two words to the right, including target word itself). Then we build count vectors for letter 3-grams of context words for every of these positions. After this we replace raw counts with PPMI values and concatenate count vectors. Definition of PPMI was introduced in Section 2.3.2, Chapter 2; here we specify the modified PPMI equation for the reader's convenience.

PPMI values are calculated for a target word w and a letter 3-gram t_i with respect to the relative position of the context word to the target word $pos \in \{-2, -1, 0, 1, 2\}$:

$$PPMI(w, t_{i_{pos}}) = \max\{\log \frac{P(w, t_{i_{pos}})}{P(w)P(t_{i_{pos}})}, 0\}$$

where $P(w, t_{i_{pos}})$ is the probability to observe a target word w with a context word on a relative position pos that contains letter 3-gram t_i; $P(w)$ and $P(t_{i_{pos}})$ are the probabilities to observe a word w and a letter 3-gram $t_{i_{pos}}$ in the corpus respectively.

For separate models. For all DR words, we concatenate 5 derived positioned letter 3-gram PPMI vectors. We also explore normalization schemes where PPMI values are divided by a constant value or distributional vectors are normalized (normalization schemes are described in Section 3.3.2.3). We enrich positioned vectors with diagonal matrix that precedes the distributional vector. All numbers on diagonal of the derived matrix are set to 1.

For mixed models. For each target word, we concatenate 5 derived positioned PPMI vectors. We normalize the obtained vector, and build distributional vector for each DR word based on whether its PPMI vector is similar enough to PPMI vectors of other words. To determine the similarity threshold α, we apply steps described in Algorithm 1. In experiments with

letter 3-gram, we set NUMBER_OF_ONES = 10, N = 100, SIMILARITY_WEIGHT = {1 on diagonal, 0.1 otherwise}.

Further we refer to LMs trained with such distributional representations as "letter 3-gram models".

3.3.2.3 Normalization schemes

The main intuition in use of normalization schemes is that word representation should provide strong word's identity signal, that is not overridden by activation signals from other words; to put this another way, values on diagonal of representation matrix should be the highest (equal 1), and non-diagonal values should by limited to the subinterval of [0, 1].

There are 4 normalization approaches we apply to our models:

1. constant (**W**): all non-diagonal values are set to a fixed constant, like 1 or 0.1;

2. scale (**S**): all non-diagonal values are divided by a fixed constant S_coeff $\in$ {6, 12, 60}. As the maximum value in the our distributional matrices is around 6, division by 6 scales values to [0, 1], division by 12 scales values to [0, 0.5], and division by 60 scales values to [0, 0.1];

3. row normalization (**RN**): every non-diagonal value is divided by a sum of all values (except of the diagonal "1") in a *row* of the PPMI matrix, and then multiplied by a constant coefficient. After division, all values will be in range [0, 1]; multiplication with a constant will scale values to the range [0, RN_coeff]. RN_coeff $\in \{1.0, 0.5, 0.1\}$.

4. column normalization (**CN**): every non-diagonal value is divided by a sum of all values (except of the diagonal "1") in a *column* of the PPMI matrix, and then multiplied by a constant coefficient. After division, all values will be in range [0, 1]; multiplication with a constant will scale values to the range [0, CN_coeff]. CN_coeff $\in \{1.0, 0.5, 0.1\}$.

    - In CN normalization without coefficient ("CN-none"), we divide every PPMI value by its context frequency. The idea here is to suppress frequent features that appear too often, and support infrequent features in the role they play in word representation.

We apply these strategies to restrict activation on the input of an NN. RN normalization is conceptually different from CN normalization: in RN normalization, values that correspond to *every context word* are scaled depending on each other; in CN normalization, values that correspond to *every input unit* of NN are scaled depending on each other.

Normalization strategies are different for mixed and separate models: W normalization is applied to mixed models only, and S, RN, and CN normalizations are applied to separate ones. The reason we use different normalization schemes is that the construction of the distributional matrices is different:

- in case of mixed models, distributional matrix is basically a similarity matrix of target and context words, and it is up to us to decide which values to assign to non-diagonal elements;
- in case of separate models, distributional representations of DR words are concatenated positioned vectors that contain real values from [0, 6], and we would like to preserve the association degree between words by scaling these values.

3.3.2.4 Unknown words treatment in distributional representations

Since unknown words can appear in the training corpus (when reduced vocabulary is used), we need to have a method to construct distributional representations for <UNK> token. We propose 3 strategies:

- **onehot**: represent unknown token with a one-hot vector;
- **averaged**: representation vector of unknown token is set to the vector of a word with frequency 1 that is the closest to an average of vectors of words with frequency 1;
- **random**: representation vector of unknown token is set to a representation of a random word with frequency 1.

3.3.3 Neural Language Models

We decide to use Log-Bilinear language model (LBL) language model for its simplicity and fast training. In the rest of this section, we describe LBL model and its modification – vector LBL (vLBL), which is the NNLM we use.

3.3.3.1 Log-bilinear language model

LBL was introduced by Mnih and Hinton (2007) as a simple LM that predicts a vector representation for the next word as a linear function on representations of the context words. The next word is then determined by the probability distribution over the words from the output vocabulary. These probabilities are computed based on how similar words representations are to the predicted representation. Following Mnih and Hinton (2008), these computations can be formalized as:

$$\hat{q} = \sum_{i=1}^{n-1} C_i r_{w_i},$$

$$P(w_n = w | w_{1:n-1}) = \frac{exp(\hat{q}^T q_w + b_w)}{\sum_j exp(\hat{q}^T q_j + b_j)}.$$

Here r_{w_i} stands for vector representations of context words, C_i are weight matrices for each context position i, q_w are representations of words in the prediction space, $\hat{q}$

is a predicted representation, and equation for $P(w_n = w|w_{1:n-1})$ computes probability distribution over vocabulary words. b_w is a bias for each word w, and it is used "to capture the context-independent word frequency" (Mnih and Hinton, 2008).

There are several proposed modified versions of LBL: hierarchical LBL (Mnih and Hinton, 2008), vLBL and inverse vLBL (ivLBL) (Mnih and Kavukcuoglu, 2013).

3.3.3.2 Vector Log-Bilinear Language model

vLBL was proposed by Mnih and Kavukcuoglu (2013) as a simplification of LBL model. Main goal of such simplification is to make model more scalable by reducing the number of parameters to train. This modification, as an original LBL model, operates on target and context representations of word w: q_w and r_w respectively. Predicted vector is computed similar to LBL model:

$$\hat{q}(w_1^{n-1}) = \sum_{i=1}^{n-1} c_i \odot r_{w_i},$$

where r_{w_i} stands for vector representations of context words, c_i are weight **vectors** for each context position i (the main difference with LBL model, where C_i are weight matrices), $\hat{q}$ is a predicted representation given history of $(n-1)$ words w_1^{n-1}. The composition in this case is a point-wise multiplication.

In our experiments, we used even more simplified version of vLBL from (Mnih and Kavukcuoglu, 2013) where the combination of context vectors is just an average of context words' vectors: $\hat{q}(w_1^{n-1}) = \frac{1}{n-1}\sum_{i=1}^{n-1} r_{w_i}$.

3.3.3.3 Initialization of LBL with distributional vectors

In order to employ during the training the created distributional representations for words (see Section 3.3.2), we change the computations of context representations r_{w_i}. Now r_{w_i} is computed as a composition of distributional vector d_{w_i} of a word and randomly initialized low-dimensional matrix R_w:

$$r_{w_i} = d_{w_i} R_w.$$

The d_{w_i} vectors stay fixed during the training, while values of R_w are adapted.

3.3.4 Neural network language model training

We train our NNLM with use of AdaGrad for adapting learning rate. In order to speed up the training, we employ noise-contrastive estimation in our learning objective. These methods are described below.

3.3.4.1 AdaGrad

To train vLBL model, we use AdaGrad. AdaGrad was proposed in (Duchi et al., 2011) as an adaptive gradient algorithm that "dynamically incorporates knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning". It is widely used in the NLP research field to improve NNs training, for examples see (Mikolov et al., 2013a), (Botha and Blunsom, 2014), (Socher et al., 2013).

The main idea of AdaGrad is to use the history of computed gradients on steps $1..t$ together with the gradient of step $(t+1)$ in the derivation of the parameter updates for the $(t+1)^{th}$ step. In a gradient descent algorithm, the next values of parameter vector x_{t+1} are computed according to previous values x_t, learning rate η and gradient g_t:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \mathbf{g}_t.$$

Duchi et al. (2011) propose to compute the update of each parameter $x_{t+1,i}$ as follows:

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} g_{t',i}^2}} g_{t,i}.$$

This update can be applied to the whole parameter vector $\mathbf{x}_{t+1}$ at once:

$$\mathbf{x}_{t+1} = \mathbf{x_t} - \eta \ \mathrm{diag}(\mathbf{G}_t)^{-1/2} \odot \mathbf{g}_t.$$

Here matrix $\mathbf{G}_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T$; $\odot$ stands for a point-wise multiplication.

3.3.4.2 Noise-contrastive estimation

Noise-contrastive estimation (NCE) was proposed by Gutmann and Hyvärinen (2012) as a method to speed up the training of NNs.

The main computational bottleneck of NNs is the estimation of the partition function Z that is needed to turn scores predicted by NN for each word into a probability distribution over the vocabulary:

$$P_\phi^h(w) = \frac{1}{Z} \exp(s_\phi(w,h)),$$
$$Z = \sum_{w'} \exp(s_\phi(w',h)),$$

where $s_\phi(w,h)$ is a scoring function with parameters ϕ defined by the NN.

The value of Z is computed for every pass of training example through the NN, that makes training time depend on the vocabulary size – usually an extremely huge number. The main idea of NCE is to avoid complex computation of a partition function by replacing estimation of the probability density with a binary classification problem. In other words, we would like the model to be able to differentiate between samples derived from the real data distribution and the samples derived from a known noise distribution.

Following (Mnih and Kavukcuoglu, 2013)[1], we denote with $P_d^h(w)$ the distribution we would like to learn (d for "data", h for "history"). By creating a noise distribution $P_n(w)$ (n for "noise") and making the model to classify which distribution the sample comes from, we reduce the problem to a binary classification. Mnih and Kavukcuoglu (2013) suggest to use the global unigram distribution of the training data as the noise distribution. Assume that on each training pass the model needs to differentiate one data sample from k noise samples. Then the posterior probability has the form:

$$P^h(D = 1|w) = \frac{P_d^h(w)}{P_d^h(w) + kP_n(w)}. \tag{3.2}$$

As we do not have access to the data distribution but only to its estimate, the posterior probability can be re-formulated taking into account the model parameters ϕ:

$$P^h(D = 1|w, \phi) = \frac{P_\phi^h(w)}{P_\phi^h(w) + kP_n(w)} \tag{3.3}$$

The following objective function suits to learn how to distinguish samples from data and from noise distribution:

$$\begin{aligned} J^h(\phi) &= E_{P_d^h}\left[\log P^h(D = 1|w, \phi)\right] + kE_{P_n}\left[\log P^h(D = 0|w, \phi)\right] = \\ &= E_{P_d^h}\left[\log \frac{P_\phi^h(w)}{P_\phi^h(w) + kP_n(w)}\right] + kE_{P_n}\left[\log \frac{kP_n(w)}{P_\phi^h(w) + kP_n(w)}\right] \end{aligned} \tag{3.4}$$

The gradient can then be computed as follows:

$$\frac{\partial}{\partial\phi} J^{h,w}(\phi) = \frac{kP_n(w)}{P_\phi^h(w) + kP_n(w)} \frac{\partial}{\partial\phi} \log P_\phi^h(w) - \sum_{i=1}^{k}\left[\frac{P_\phi^h(x_i)}{P_\phi^h(x_i) + kP_n(x_i)} \frac{\partial}{\partial\phi} \log P_\phi^h(x_i)\right] \tag{3.5}$$

In contrast to a partition function, in this case the gradient computations contain only a sum over k noise samples. This makes training time independent of the vocabulary size, but dependent only on the number of noise samples k. Another key point of NCE is that weights $\frac{P_\phi^h(x_i)}{P_\phi^h(x_i)+kP_n(x_i)}$ are always between 0 and 1, that makes training with NCE very stable (Gutmann and Hyvärinen, 2010).

3.3.5 Models interpolation

A combination of models usually performs better than independent models [Chen and Goodman (1999), Schwenk (2007), Masumura et al. (2015)], since it allows the resulting

[1]Eq. 3.2, Eq. 3.3, Eq. 3.4, Eq. 3.5 are based on (Mnih and Kavukcuoglu, 2013).

model to benefit from strength of both interpolated models. That is why we interpolate every learned LBL model with a modified Kneser-Ney n-gram model.

Applied interpolation is a linear combination with parameter $\lambda \in [0, 1]$:

$$P_{INT}(w_n|w_1..w_{n-1}) = \lambda P_{LBL}(w_n|w_1..w_{n-1}) + (1-\lambda)P_{MKN}(w_n|w_1..w_{n-1}).$$

3.3.6 Models evaluation

After the models are trained, we use them to predict test data and report achieved perplexity. Special strategies to handle cases where the model needs to predict a word that it has not seen during training are discussed in Section 3.3.6.2.

3.3.6.1 Perplexity

There are 2 main approaches to evaluate the performance of LMs: extrinsic evaluation and intrinsic evaluation.

In extrinsic evaluation, LM is used as a part of a bigger system (e.g., maching translation, speech recognition) and evaluate quality of LM based on the performance of this system.

Intrinsic evaluation for LM is done with *perplexity* – the measurement of how good a probability model can predict a given sample. Maximizing probability of a word sequence is the same as minimizing perplexity. Perplexity is computed as follows:

$$PPL(w_1...w_N) = \sqrt[N]{\frac{1}{P(w_1...w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1...w_{i-1})}} = b^{-\frac{1}{N}\sum_{i=1}^{N} \log_b P(w_i|w_1...w_{i-1})},$$

where b usually is set to 2 or e. In our experiments, we set $b = 10$.

3.3.6.2 Unknown words treatment in language model predictions

Probability evaluation of unknown words – those that were not seen during the training – require a special treatment. Usually a development set is used to optimize the hyperparameters, and the final performance is measured on the test set. But in our work we use development set (here we refer to it as a "test set" in the experiments, see Section 2.4.1) both to estimate parameters and to report the model performance. The reported results are preliminary, and should be used as suggestions for the directions of future research with evaluation on the test set.

The two cases that need special treatment consider full and partial vocabulary.

Full vocabulary
In case our training vocabulary contains all words from the training set, we

- add an <UNK> token to the vocabulary,

- set $P_{new}(\text{<UNK>}) = \frac{unknown\_dev}{|dev|}$,
- to compensate, the probabilities of known words w_i get squeezed: $P_{new}(w_i) = (1 - P_{new}(\text{<UNK>}))P_{LBL}(w_i)$.

where $|train|$ and $|dev|$ are sizes of train and development set respectively, $unknown\_dev$ is number of unknown words in the development set. P_{new} are the resulting probabilities used for the model evaluation.

Partial vocabulary

In case we experiment on reduced vocabulary where we replace some words in the training set with <UNK> token (experiments in Section 3.5.3), we employ a slightly different strategy: number of unknown words is estimated as a composition of a number of all unknown words in the training corpus and unknown words in the development set. Then the probabilities of *seen unknown words* are set proportionally to the number of their occurrences in the training corpus; the probabilities of *unseen unknown words* are replaced with estimated unknown words count. The probabilities of known words are left unchanged. To be exact:

First, we estimate a total number of unknown words – through a ratio of unknown words in the development set and a frequency of <UNK> token:

$unknown\_train\_estimate = |train|\frac{unknown\_dev}{|dev|}$,
$unknown\_total\_estimate = freq_{train}(\text{<UNK>}) + unknown\_train\_estimate$.

Here $|train|$ and $|dev|$ are sizes of train and development set respectively, $unknown\_dev$ is number of unknown words in the development set, $freq_{train}(\text{<UNK>})$ is a frequency of <UNK> token in the training set.

Second, though we replace words from the training corpus with <UNK> token, we still know what words are there. We call these words *seen unknown* words, and put their frequencies in a map $\{(seen\_unknown_i \rightarrow freq_{train}(seen\_unknown_i))\}$.

Third, we replace predicted probabilities with P_{new}:

- for unknown words:
  - for seen unknown words: $P_{new}(seen\_unknown_i) = \frac{freq_{train}(seen\_unknown_i)}{unknown\_total\_estimate} P_{LBL}(\text{<UNK>})$;
  - for never seen unknown words: $P_{new}(unseen\_unknown) = \frac{unknown\_train\_estimate}{unknown\_total\_estimate} P_{LBL}(\text{<UNK>})$;

    Here $freq_{train}$ values are taken from the map created on the second step.
- for known words: leave the predicted values without changes.

3.4 Experimental setup

3.4.1 Training corpus

For training, we use the Wall Street Journal corpus (WSJ) created by Marcus et al. (1999) from Wall Street Journal articles of 1989. We use the traditional split of the WSJ corpus into 3 parts: parts 00–20, parts 21–22 and parts 23–24.

We use parts 00–20 as a train set; it contains 1,004,073 tokens in 42,075 sentences with 45,565 word types.

Parts 21–22 were used as a test set; it contains 80,156 tokens in 3371 sentences with 10,804 word types.

The rest parts 23–24 contains 89,537 tokens in 3762 sentences with 11,291 word types, and were not used in our experiments.

We use train set for building distributional representations and for models training, and report results on the test set.

Usually a development set is used to optimize the hyper-parameters, and the final performance is measured on the test set. In our work here, we use "traditional" development split of the WSJ – parts 21–22 – both to estimate hyper-parameters and to report model performances. The reported results are preliminary, and should be used as suggestions for the directions of future research with evaluation on the test split of the WSJ (parts 23–24).

3.4.2 Vocabularies

To make the corpus suitable for language modeling task, we add to the vocabulary special tokens for an unknown word <UNK>, beginning of a sentence <S>, end of a sentence <\S>, and padding <PAD>.

We explore several different ways to construct target and context vocabularies:
45K vocabulary includes all 45,569 words from the WSJ training set.
2vocab vocabulary includes all words from the WSJ training set that have frequency ≥ 2, contains exactly 24,532 words.
10K vocabulary includes top frequent 10K words from the WSJ training set, contains exactly 10,004 words (10K + 4 special tokens). The smallest frequency observed in this vocabulary is 6.

There are also 2 special vocabularies that are built from preprocessed corpus:
40K vocabulary is created from the training corpus where all digits are replaced with one token; this leads to a vocabulary of size 40,370.
35K vocabulary is created from the training corpus where all digits are replaced with one token and all words are lower cased; this leads to a vocabulary of size 35,385.

3.4.3 Evaluation task

We train a modified Kneser-Ney n-gram LM and several LBL models on the WSJ train set, interpolate each LBL model with the modified Kneser-Ney model and report the perplexity

of the interpolated model on the WSJ test set. The lower the perplexity, the higher is the probability that our model assigns to the test set and the better the model is considered.

Unlike Wang and Cho (2015) that evaluate and report perplexity for words with particular part-of-speech tag only, we are interested not only in the increase of performance for rare words, but in overall LM performance. That is why we report perplexity points obtained for all words in the test corpus as results of our experiments.

3.4.4 Frequency ranges for distributional initialization: hyperparameter θ

θ	# of words
1	21037
2	6648
3	3335
4	2147
5	1501
6	1198
7	827
8	694
9	598
10	489
20	158

θ	# of words
$[1, 1]$	21037
$[1, 2]$	27685
$[1, 5]$	34668
$[1, 10]$	38474
$[2, 2]$	6648
$[2, 5]$	13631
$[2, 10]$	17437
$[6, 6]$	301
$[6, 10]$	2909

Table 3.1: Number of words in the 45K vocabulary for particular values of the frequency range θ: constant θ on the *left* and interval θ on the *right*.

Parameter θ is a frequency range that determines which words from the vocabulary receive distributional initialization and which receive one-hot: words with frequency $\in \theta$ are initialized with distributional vectors; others are initialized with one-hot[2].

We would like to investigate the performance of LMs when rare words receive distributional initialization, that is why for this we pick words that have low frequencies. In experiments, θ is set to one of the intervals {[1, 1], [1, 2], [1, 5], [1, 10], [2, 2], [2, 5], [2, 10], [6, 6], [6, 10]}, or to a constant value from $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20\}$ for ONLY models: further "frequency is in θ" means that frequency either belongs to the interval or frequency is equal θ. We specify the utilized values of θ for each experiment. Table 3.1 shows how many words in the 45K vocabulary have particular frequency for different values of θ.

[2]In contrast to Chapter 2, it is not correct to call words with frequency $\notin \theta$ "frequent" anymore since for ONLY models there are rare words that also receive one-hot initialization; though all words that receive distributional initialization in following experiments *are* rare.

Same as in Section 2.4.5 of Chapter 2, the argument about the ranges of values of θ is valid in the experiments of this chapter: the values of θ do not depend on the training corpus size because if a certain amount of information is needed to learn meaningful representations for words, than it should not matter whether that given number of contexts occurs in a small corpus or in a large corpus. However, if the contexts themselves contain many rare words (which is more likely in a small corpus), then a corpus size should probably be taken into account.

3.4.5 Training regime

vLBL architecture and training method are implemented in Python 2.7 by means of the Theano library [Bergstra et al. (2010), Bastien et al. (2012)].

Number of training epochs. We usually train vLBL models for 10 or 20 epochs, since the convergence appears much earlier. In case the model does not converge, we continue training up to 50 epochs.

Batch size is set to 100.

Learning rate. We first set learning rate to 0.5, but change is later to 0.1 after evaluation of different values of learning rate in Section 3.5.7. During the training, the learning rate is adapted with AdaGrad.

Number of noise samples for NCE is set to 25. Same as (Mnih and Kavukcuoglu, 2013), we use global unigram distribution of the training data as the noise distribution.

Word embeddings size is set to 100.

Context window for training vLBL model is 2 words to the left (3-gram model). We pad sentences to the left with <PAD> token and allow learning of end-of-sentence token <\S>.

Interpolation weight. For experiments in Section 3.5.3-Section 3.5.5, we change interpolation weight λ in [0, 1] with step 0.1. Starting from the experiment in Section 3.5.6, we explore λ in [0, 0.2] with step 0.02. λ is used to weight output of LBL model, $(1 - \lambda)$ weights output of modified Kneser-Ney LM.

Training time of one epoch of LM is ∼30 min, predicting time for one epoch is ∼20 min on a single 2.80GHz or 3.00GHz CPU core.

3.4.6 Reporting results

In experiments, we store every model after each training epoch. Then we interpolate every stored model with modified Kneser-Ney n-gram model, changing LBL interpolation weight λ in interval [0, 1] (as mentioned above). We report, for every model initialization setting, the best value that was achieved during interpolation and indicate epoch number and LBL interpolation weight for this value.

3.5 Experiments

3.5.1 Modified Kneser-Ney n-gram models

We train 3-gram and 5-gram modified Kneser-Ney LM with SRILM toolkit [3] [4]. Perplexity of these n-gram models is 173.07 for 3-gram model (KN3) and 186.17 for 5-gram model. All words from the training corpus were included in the vocabulary. We use 3-gram model for interpolation with our trained LBL models.

3.5.2 Baseline models with one-hot initialization

	vocabulary		
	45K	2vocab	10K
perplexity	392.78(2e)	201.46(2e)	194.75(2e)
interpolated perplexity	171.90(2e, .1)	148.58(2e, .4)	146.81(2e, .5)

Table 3.2: Perplexity of vLBL models with one-hot initialization for different vocabularies (see Section 3.4.2). Interpolation is performed with modified Kneser-Ney 3-gram model (KN3). In parentheses: for the best value, number of the epoch and LBL interpolation weight λ [$\lambda P_{LBL} + (1 - \lambda)P_{MKN}$].

We train three vLBL models with one-hot initialization, changing target and context vocabulary sizes: 45K, 2vocab, 10K (see Section 3.4.2). These models are used as baselines for comparison with vLBL models with distributional initialization.

Table 3.2 shows the best perplexity results of one-hot initialized models, together with number of epochs and LBL interpolation weight $\lambda \in [0, 1]$ with step 0.1.

Initial learning rate for vLBL training is set to 0.5. Number of epochs is set to 30 for model with 45K vocabulary, and to 20 for models with 2vocab and 10K vocabularies.

3.5.3 Binary mixed models with different target/context vocabularies

Description

In this experiment, we would like to investigate the performance of LMs with different target and context vocabularies for different values of interval θ. We compare performance of one-hot initialized models with binary mixed distributional models, setting target/context vocabularies to 45K, 2vocab, and 10K. Parameter θ takes values from intervals {[1, 1], [1, 2], [1, 5], [1, 10]}; in case of 10K/10K, intervals are {[6, 6], [6, 10]} since the 10K vocabulary does not contain words with frequency less than 6. Co-occurrence window size

[3] `www.speech.sri.com/projects/srilm`

[4] For the training scripts, see Appendix B

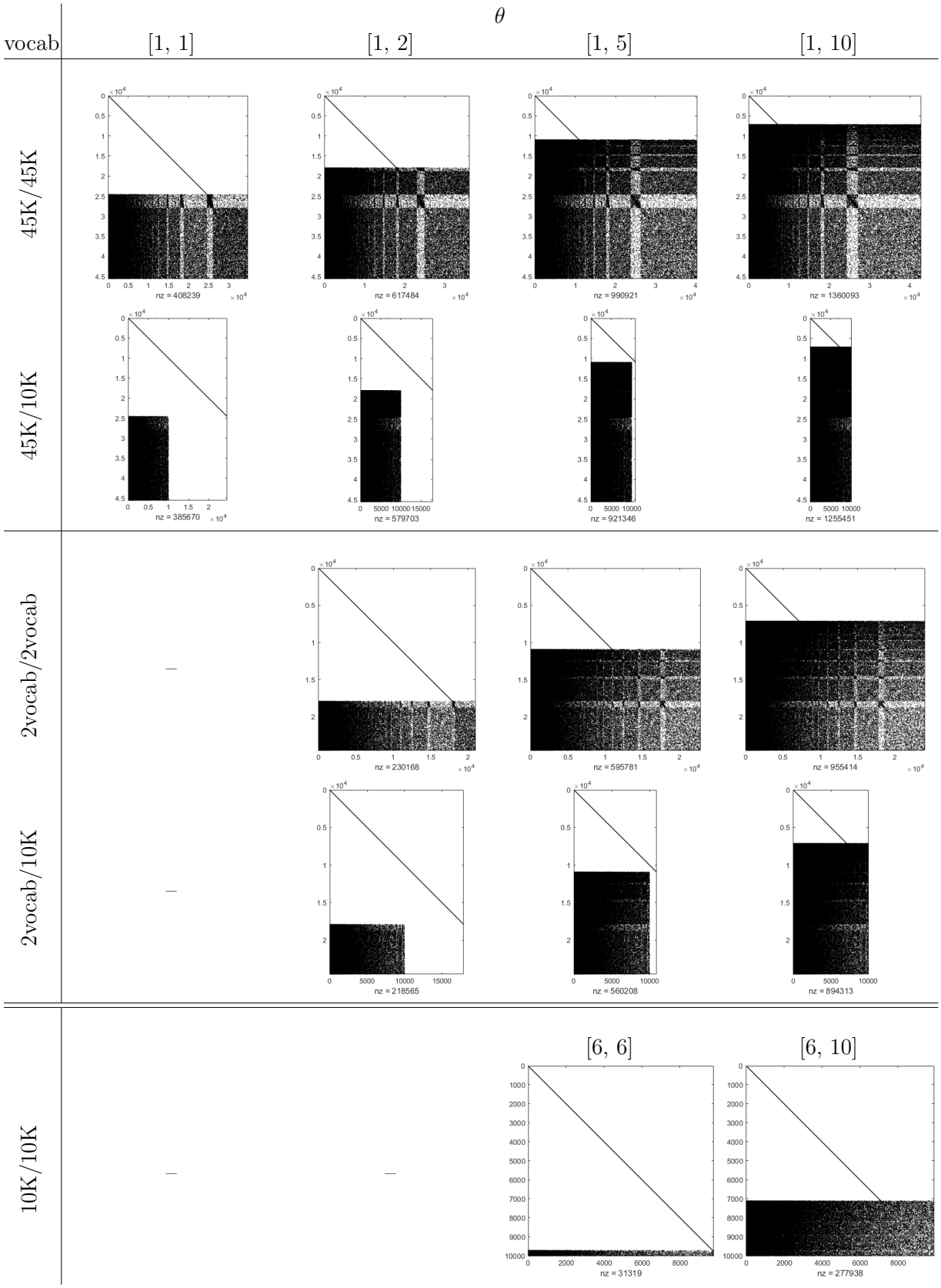


Figure 3.1: Binary mixed distributional representations for different target/context vocabularies as in Table 3.3. 10K/10K models have different interval θ.

target/context vocabulary	one-hot	θ			
		[1, 1]	[1, 2]	[1, 5]	[1, 10]
45K/45K	171.90(2e, .1)	173.84(2e, .1)	175.18(4e, .1)	175.59(16e, .1)	175.94(20e, .1)
45K/10K		174.39(3e, .1)	175.38(2e, .1)	176.75(11e, .1)	177.41(20e, .1)
2vocab/2vocab	148.58(2e, .4)	–	153.56(3e, .4)	157.49(9e, .3)	160.34(19e, .3)
2vocab/10K		–	154.06(2e, .4)	159.43(14e, .3)	162.24(16e, .2)

				[6, 6]	[6, 10]
10K/10K	146.81(2e, .5)	–	–	148.25(3e, .4)	152.70(8e, .4)

Table 3.3: Perplexity of vLBL models with one-hot and binary mixed initialization, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. For all vocabularies and θ values, vLBL with one-hot initialization performs better than models with binary mixed initialization. We continue to use setting 45K/45K in the following experiments.

for building distributional representations is 21: 10 context words to the left and 10 to the right of a target word were taken into account.

Distributional representations

Distributional representations are build using words co-occurrences according to Section 3.3.2.2, and their matrices are shown in Figure 3.1. Number of words that receive distributional representations for different values of θ can be found in Table 3.1.

For unknown token, for 45K/45K and 45K/10K settings, averaged representation strategy is used. For 10K/10K, 2vocab/10K, 2vocab/2vocab, one-hot representation is used.

LBL training

vLBL models are trained for 20 epochs with initial learning rate of 0.5.

Results

Results are presented in Table 3.3. One-hot initialized LMs outperform binary mixed models for all tested vocabularies and all frequency intervals for distributional initialization. Reduction of the target vocabulary (from 45K to 2vocab and then to 10K) leads to a decrease of perplexity, while reduction of the context vocabulary (from 45K to 10K for 45K target vocabulary, and from 2vocab to 10K for 2vocab target vocabulary) leads to an increase of perplexity. Performance of models for all target/context vocabularies combinations goes down with the increase of the interval size: best results are observed for θ=[1, 1] and worst are observed for θ=[1, 10].

Discussion

Target vocabulary. The behavior we observe for different target vocabulary sizes is expected: the smaller the target vocabulary the smaller is perplexity. This happens due to the fact that reduction of the target vocabulary draws a reduction of the number of possible predictions and an increase in the probability of the unknown token.

Context vocabulary. The reduction of the context vocabulary decreases the amount of available information at the input level of the NN, that leads to a decrease of the model's capacity and worsen the performance.

Different frequency intervals. The probable reason for the performance decay with expansion of intervals of θ is that, due to the fact that every value in distributional vectors is equal to 1, LBL model with such distributional initialization receives too ambiguous and noisy signals at its input layer. Numbers of values in the distributional matrices could be seen in Figure 3.1. The possible solution to the low performance and its decay is to use normalization schemes, introduced in Section 3.3.2.3, to control the strength of the signals at the input layer of the NN.

Conclusions

In this experiment, for different target/context vocabularies the performance of distributional models for interval θs do not exceed one-hot models. We would like to investigate models performance incorporating knowledge about words positions in the context window. In the following experiments, we employ 45K/45K setting of target/context vocabulary for distributional representations creation.

3.5.4 Positioned PMI/PPMI models with interval θ

		θ			
	values	[1, 1]	[1, 2]	[1, 5]	[1, 10]
W1noo10	PMI	172.86(2e, .1)	172.93(3e, .1)	172.67(2e, .1)	172.84(3e, .1)
mixed W1noo2	PPMI	172.61(2e, .1)	172.76(2e, .1)	172.65(2e, .1)	172.58(2e, .1)
W0.1noo10	PPMI	172.40(2e, .1)	172.49(2e, .1)	172.26(2e, .1)	172.45(2e, .1)
separate	PPMI	173.10(4e, .1)	173.36(4e, .1)	173.83(5e, .1)	174.38(9e, .1)

Table 3.4: Perplexity of vLBL models with positioned PMI/PPMI mixed and separate initialization, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. All models with distributional initialization perform worse than one-hot baseline (171.90); fine-grained analysis with ONLY models is needed. "noo" for NUMBER_OF_ONES, "W" for non-diagonal values.

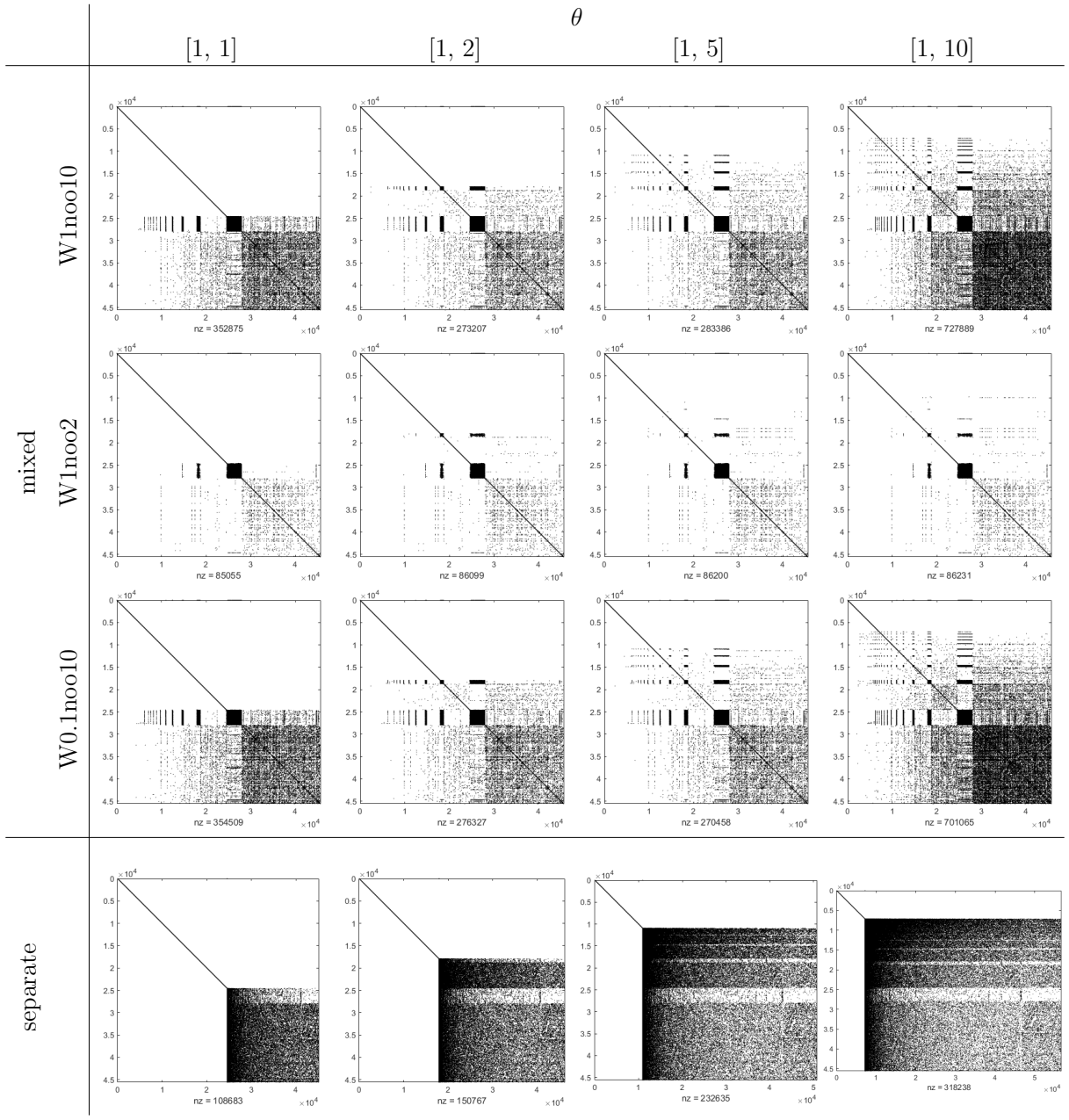


Figure 3.2: Positioned PMI/PPMI mixed and separate distributional representations as in Table 3.4. "noo" for NUMBER_OF_ONES, "W" for non-diagonal values.

Description

Since plain binary co-occurrence vectors did not work well, we decided to follow the approach of Levy and Goldberg (2014b) to employ positioned PMI/PPMI vectors in order to build distributional word representations.

Distributional representations: Mixed

Mixed distributional representations for words with frequency $\in \theta$ are build as described in Section 3.3.2.2. Parameter θ receives values from {[1, 1], [1, 2], [1, 5], [1, 10]}.

Specific parameters for mixed distributional representations are set as follows:

- W1noo10:
  - NUMBER_OF_ONES = 10;
  - SIMILARITY_WEIGHT = 1 for all words;
  - pointwise mutual information (PMI) values are used in positioned vectors;
- W1noo2:
  - NUMBER_OF_ONES = 2;
  - SIMILARITY_WEIGHT = 1 for all words;
  - PPMI values are used in positioned vectors;
- W0.1noo10:
  - NUMBER_OF_ONES = 10;
  - SIMILARITY_WEIGHT = 1 on diagonal and 0.1 for nearest neighbors;
  - PPMI values are used in positioned vectors.

Distributional representations: Separate

Separate distributional representations are build as described in Section 3.3.2.2. Parameter θ receives values from {[1, 1], [1, 2], [1, 5], [1, 10]}.

<UNK> representation is averaged. Sparse distributional matrices are shown in Figure 3.2.

LBL training

For vLBL training, learning rate is set to 0.5, number of epochs is 30.

Results

Results are presented in Table 3.4. One-hot model performs better than positioned PMI/PPMI mixed and separate distributional models with interval θ.

Discussion

Different values of θ. The performance of mixed and separate models does not follow a stable pattern: it goes up and down for different analyzed intervals of words frequencies. This means that fine-grained analysis of performance on different frequencies is needed in order to explain these alternations.

SIMILARITY_WEIGHT. Model with W=0.1 outperform other two mixed models, with NOO=2 and W=1. The reason of the better performance of model with W=0.1 than model with W=1 can be that in case when all values in the distributional matrix are the same (W=1), it is hard for the NN to identify words due to the fact that some different words have the same vectors at the input layer.

NUMBER_OF_ONES. The performance of mixed model with NOO=2 is better for all intervals than of model with NOO=10. We would expect that combination of embeddings of several context words with target word will introduce more noise at the input layer of NN in case of NOO=10 than in case NOO=2, that will influence negatively on the model's performance.

Conclusions

To investigate reasons of poor performance of the models with distributional initialization, we need to conduct fine-grained analysis. For this, we investigate distributional initialization only for words with a particular frequency in the following experiments.

We also found out that model with W=0.1 outperforms models with equal weights (W=1), so we are going to set W=0.1 in the future experiments.

All following models are trained with use of positioned PPMI vectors to build distributional representations.

3.5.5 ONLY distributional models

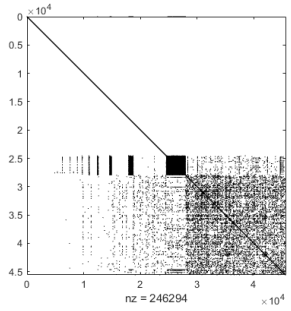
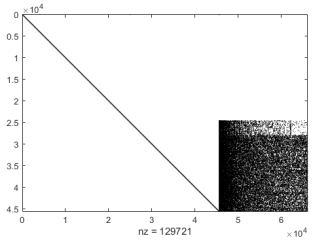
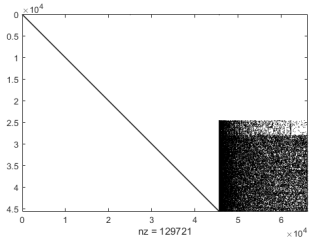
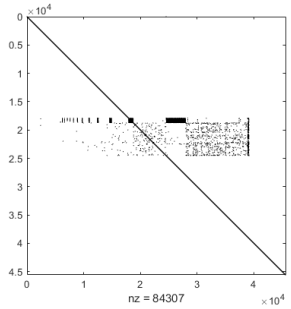
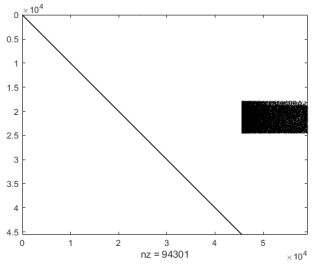
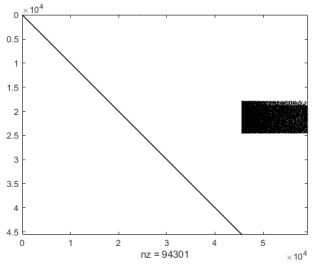
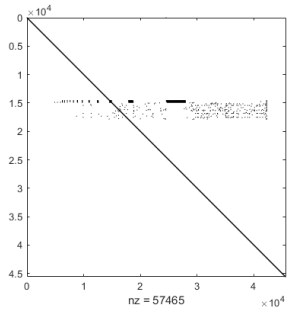
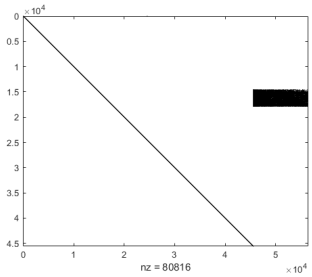
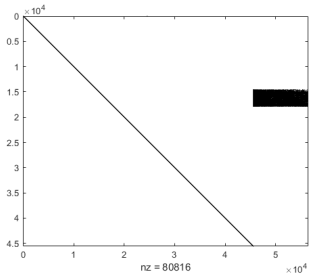
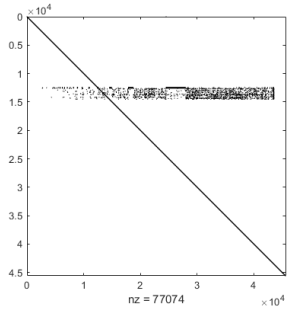
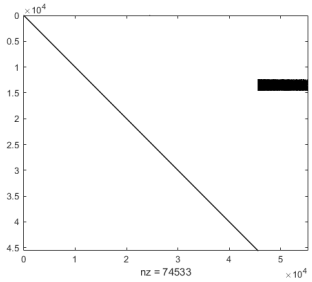
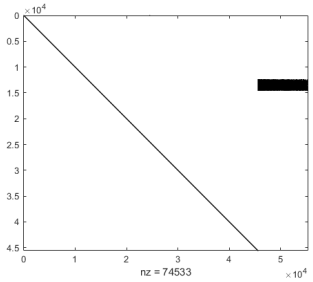
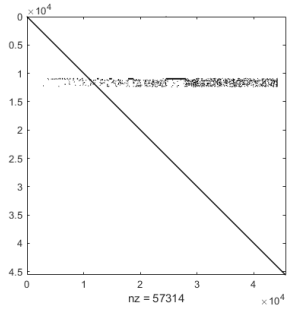
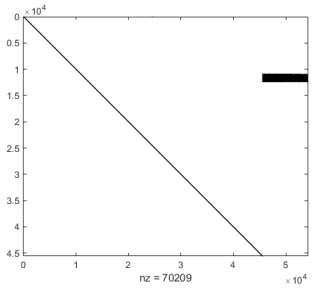
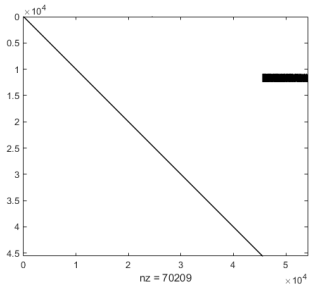
Description

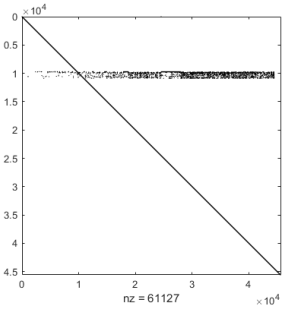
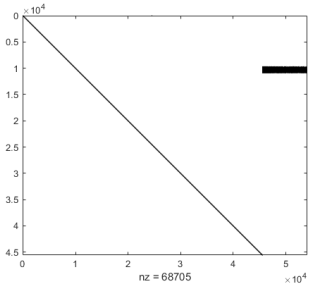
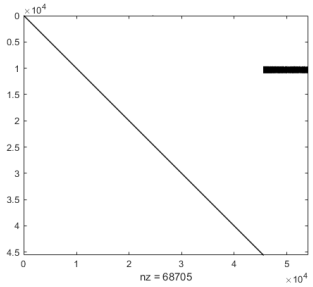
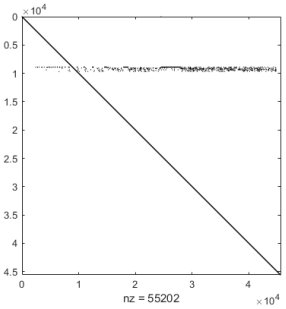
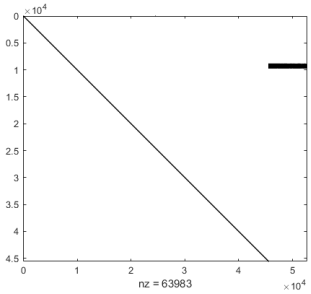
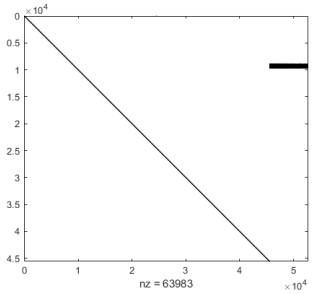
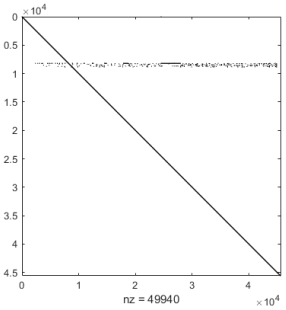
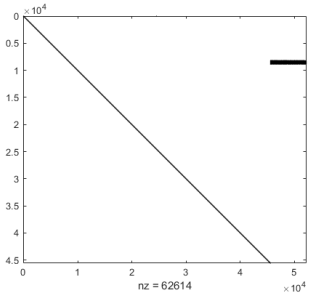
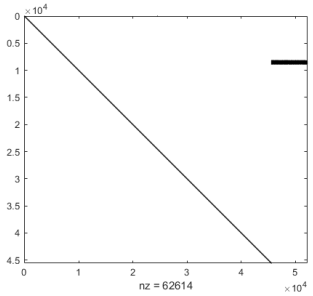
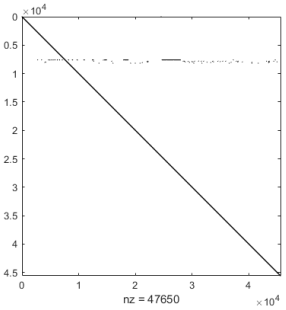
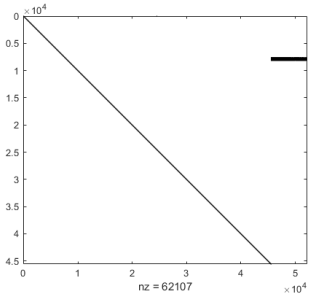
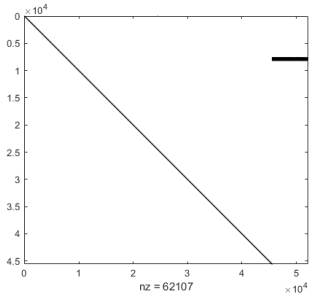
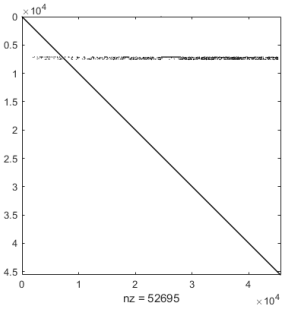
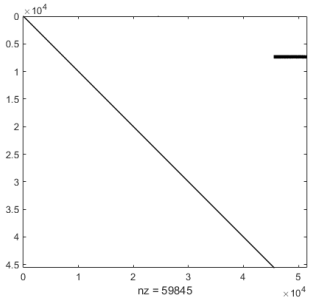
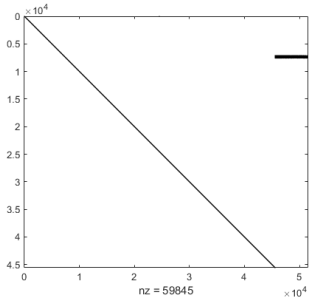
Next we decide to apply distributional representations to words with a particular frequency – ONLY models as described in Section 3.3.2.1. The frequencies we are interested in are {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20}. Sparse distributional matrices are shown in Figure 3.3.

Distributional representations

In current experiment, for mixed models we set NUMBER_OF_ONES = 10 and SIMILARITY_WEIGHT = {1 on diagonal; 0.1 for non-diagonal elements}. We explore 2 additional models:

- *mixed 10only* model without weighting: weights for all values are set equal to 1 in the distributional vector (W=1),

θ	mixedW0.1noo10	separateS12	separateS60
1			
2			
3			
4			
5			

θ	mixedW0.1noo10	separateS12	separateS60
6			
7			
8			
9			
10			

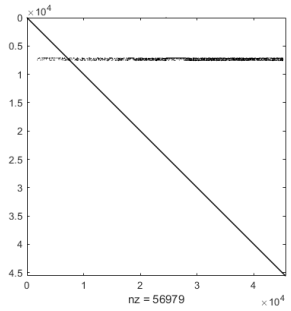
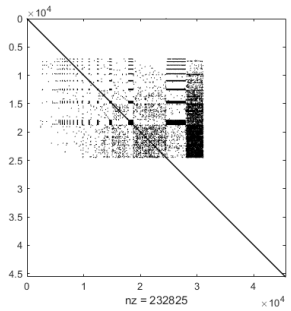
θ	mixedW0.1noo10	separateS12	separateS60
10 W=1		–	–
2-10		–	–

Figure 3.3: Initialization matrices for mixed and separate θ ONLY models, where words with frequency = θ receive distributional representations. "noo" for NUMBER_OF_ONES, "W" for non-diagonal values, "Sx" for S_coeff=x. Matrices correspond to Table 3.5.

- *mixed 2-10* model, where distributional representations are used for words with frequencies in the interval [2, 10].

For separate models, we apply S normalization scheme: divide all non-diagonal values by a constant 12 or 60 to scale values to [0, 0.5] and [0, 0.1] respectively.

<UNK> representation strategy is averaged.

LBL training

For vLBL training, initial learning rate is set to 0.5. Number of epochs is set to: for mixed 1 to 9 ONLY – 20, mixed 10 ONLY and 10 ONLY without weighting – 30; mixed2-10 – 20; separate 1 to 9 ONLY – 10, separate 10only models – 20. The number of training epochs does not influence on the resulting performance since models show best results with ≤ 3 epochs.

Results

Results are presented in Table 3.5. ONLY models show improvement over one-hot baseline from .06 to .46 for $\theta \in \{2, 3, 4, 5, 6, 7, 8, 10\}$ for mixed models, from .06 to .29 perplexity points for $\theta \in \{3, 4, 5, 6, 7, 8, 10\}$ for separate models with S normalization with S_coeff = 12, and from .05 to .32 perplexity points for $\theta \in \{4, 5, 6, 7, 9, 10\}$ for separate models with

θ	mixedW0.1noo10	separateS12	separateS60
1	172.18(2e, .1)	173.03(2e, .1)	172.69(1e, .1)
2	**171.80(2e, .1)**	171.98(2e, .1)	171.96(2e, .1)
3	**171.44(2e, .1)**	**171.71(2e, .1)**	171.98(2e, .1)
4	**171.66(2e, .1)**	**171.61(2e, .1)**	**171.58(2e, .1)**
5	**171.81(2e, .1)**	**171.69(2e, .1)**	**171.85(2e, .1)**
6	**171.79(2e, .1)**	**171.81(2e, .1)**	**171.80(2e, .1)**
7	**171.84(2e, .1)**	**171.84(2e, .1)**	**171.80(2e, .1)**
8	**171.76(2e, .1)**	**171.75(2e, .1)**	171.92(2e, .1)
9	171.90(2e, .1)	172.09(2e, .1)	**171.57(2e, .1)**
10	**171.71(2e, .1)**	**171.75(2e, .1)**	**171.81(2e, .1)**
10 W=1	172.44(3e, .1)	–	–
2-10	**171.81(2e, .1)**	–	–

Table 3.5: Perplexity of vLBL models with θ ONLY mixed and separate initialization, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Models that perform better than one-hot baseline (171.90) are **highlighted**. "noo" for NUMBER_OF_ONES, "W" for non-diagonal values, "Sx" for S_coeff=x. For almost all values of θ, mixed and separate ONLY models outperform one-hot baseline.

S normalization with S_coeff = 60. In particular, mixed3only and separate4only models have the best results. Also, mixed2-10 model perform better than one-hot model.

Discussion

For almost all values of θ, mixed and separate ONLY models outperform the one-hot baseline; though in cases of separate models, the performance does not differ much from the baseline: the models with fair difference are separate with S_coeff = 12 and S_coeff = 60 with $\theta = 4$ (.29 and .32 perplexity points), and separate with S_coeff = 60 with $\theta = 9$ (.33 perplexity points).

Performance for small values of θ. ONLY models show different performance for different values of θ. This suggests potential for further research and improvement: probably, words with different frequency should be treated differently.

SIMILARITY_WEIGHT. Mixed10only model with W=0.1 outperform the mixed10only model without weighting (W=1). The only difference in these models is the value amplitude of non-diagonal elements of the distributional matrix and the fact that non-diagonal and diagonal elements have the same values. There are several consequences of having the same weight for diagonal and non-diagonal elements:

- It pollutes the NN input signal from a target word: combination of context words embeddings with the same weight as a target word embedding takes into account not only useful information about relationships between words, but also introduce unnecessary noise that harms performance.

- It prevents the desired effect of error propagation; when weights are the same, target and context words embeddings are changed equally while we would like the most of the effect to go to the target word embedding.
- It makes DR words indistinguishable since words with same context will have exactly the same distributional vectors.

We assume that these are reasons why when all values in the distributional vectors are the same, it is hard for the NN to correctly identify and train embeddings for each DR word because word identity signal (diagonal value) is the same as subsidiary signals (non-diagonal values). We are going to investigate the effect of different values of W in Section 3.5.6.1.

Mixed1only and mixed with θ=[1,1]. The difference in performance for mixed1only and mixed with $\theta = [1, 1]$ happens due to randomization procedure for picking similarity threshold α in Algorithm 1.

Conclusions

For small values of θ, where the number of times word was seen in the corpus is too small for a meaningful representation, normalization methods could help. We hope they will intensify the signal of word's identity and diminish and balance the signals of connections between words. We are going to investigate normalization techniques in the following experiments.

Since the best values of λ are always around 0.1, from now on we will interpolate with $\lambda \in [0, 0.2]$ with step 0.02. For such LBL interpolation weights, the best value of one-hot model is 171.49(2e, .06).

Use mixed3only and separate4only models for comparison as their performance is the best among the ONLY models.

3.5.6 Exploring normalization schemes

In this experiment, we would like to investigate different ways of normalization of vectors that are used to build distributional representations. We conjecture that normalization can help to balance the signal at the input layer of NN, that can lead to better word representations inside the NN, and results in a better performance of LM.

W normalization is applied to mixed models, and S, RN, and CN normalizations are applied to separate models. Normalization schemes are described in detail in Section 3.3.2.3.

3.5.6.1 Constant normalization for mixed3only model

Description

For mixed models, "to normalize" means to define a value of SIMILARITY_WEIGHT that is used for a non-diagonal elements in distributional representations. We set this value to

SIMILARITY_WEIGHT

0.5	0.1	0.01	0.001

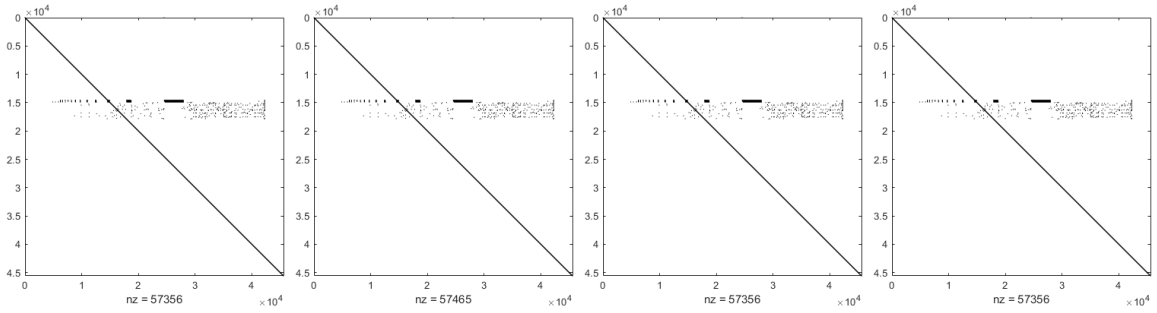


Figure 3.4: Weighted mixed3only distributional representations as in Table 3.6.

SIMILARITY_WEIGHT

0.5	0.1	0.01	0.001
171.47(2e, .06)	**171.19(2e, .06)**	**171.35(2e, .06)**	**171.26(2e, .06)**

Table 3.6: Perplexity of vLBL models with weighted mixed3only initialization, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. All mixed3only models perform better than one-hot baseline (171.49); $\lambda \in [0, 0.2]$ with step 0.02. Because of the best performance, parameter SIMILARITY_WEIGHT is set to 0.1 in the following experiments.

one from {0.5, 0.1, 0.01, 0.001}. Weights on diagonal are always kept equal 1. Distributional matrices have the same structure for different values of SIMILARITY_WEIGHT and are shown in Figure 3.4.

Distributional representations

In the experiment, we use mixed3only model: all words receive one-hot representation, except words with frequency equals 3, that receive positioned PPMI distributional representation (see Section 3.3.2.2). Other parameters are set as follows: SIMILARITY_WEIGHT = 1 for diagonal; non-diagonal values are set to one from {0.5, 0.1, 0.01, 0.001}; NUMBER_OF_ONES = 10. <UNK> representation is averaged.

LBL training

For vLBL training, initial learning rate is set to 0.5. For W $\in$ {0.1, 0.01, 0.001}, number of epochs is 20, for W = 0.5 number of epochs is 10 (that is not crucial since the model converges much earlier).

Results

Results are presented in Table 3.6. Mixed3only models with W $\in \{0.1, 0.01, 0.001\}$ outperform one-hot baseline. Value of W that leads to the best performance is 0.1.

Discussion

Obtained results confirm our hypothesis about balanced activation at the input layer of NNs: the optimal value of the subsidiary signals should not be very high or very low. In case when it is very high (1 or 0.5), the activation at the input layer becomes very noisy and makes model unable to distinguish different words. In case of very low (0.001) values, the signal of the additional information (non-diagonal elements) is not strong enough to improve the performance.

Conclusions

The best normalization value obtained is W=0.1, so we will use this value in our further experiments.

3.5.6.2 CN and RN normalizations of separate10only model

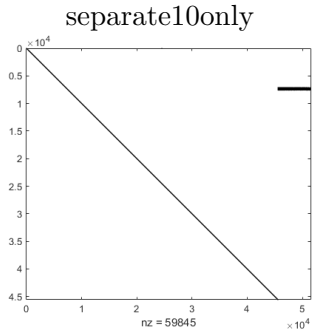


Figure 3.5: Separate10only distributional matrix, same for all normalization schemes and coefficients. Corresponds to Table 3.7.

Description

To explore normalization schemes for separate models, we use as a basis separate10only model to which we apply CN and RN normalization with coefficients 1.0, 0.5, and 0.1.

Distributional representations

In separate10only model, only words with frequency 10 receive distributional representations – in a separate manner as described in Section 3.3.2.2. In the distributional vectors

	normalization coefficient			none
	1	0.5 (12 for S)	0.1 (60 for S)	
CN	171.80(3e, .04)	**171.44(2e, .06)**	**171.34(2e, .06)**	171.54(2e, .06)
RN	**171.25(2e, .06)**	**171.38(2e, .06)**	**171.32(2e, .06)**	–
S	–	**171.40(2e, .06)**	**171.45(2e, .06)**	–

Table 3.7: Perplexity of vLBL models with normalized separate10only initializations, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Normalization schemes: *CN* – column normalization, *RN* – row normalization, *S* – scale normalization (see Section 3.3.2.3). Models that perform better than one-hot baseline (171.49) are **highlighted**. Normalized separate models perform better than one-hot.

creation, raw positioned PPMI values are re-scaled according to S, or RN, or CN normalization schemes as described in Section 3.3.2.3. For all normalization schemes, distributional matrices look the same as they differ only in the values amplitude. We depict only one of them in Figure 3.5.

<UNK> representation is averaged.

LBL training

For vLBL training, initial learning rate is set to 0.5; CN models are trained for 10 epochs, RN models – for 20 epochs.

Results

Results are presented in Table 3.7. Separate10only models with different normalization techniques slightly outperform one-hot baseline: from .04 to .24 perplexity points.

Discussion

Normalization. Almost all separate models with normalization work slightly better than one-hot baseline, that suggest that careful usage of normalization can make an impact on LM performance.

CN and RN comparison. Models with RN normalization work a bit better than models with CN normalization. The reason may be that RN normalization intensifies words' identity signal against the background activation of the connections between words. CN normalization, on other hand, can end up assigning for different words more or less the same level of activation among different dimensions, that will confuse language model.

CN_coeffs and RN_coeffs. There is no big difference in performance of RN models for different coefficients. On other hand, the performance of CN models go up when the upper limit of the normalization interval goes down. This can be due to the reduction of ambiguity in the distributional vectors: the difference in the word identity signal on

the diagonal and signals of connections between words (non-diagonal elements) becomes bigger.

CN-none. Model with normalization where entries of distributional vectors were divided by the frequency of contexts performs worse than one-hot baseline, and also worse than models with S normalization scheme. The reason can be that, despite of scaling of frequent contexts, some values in the vector still are several times higher than diagonal elements, that introduces a lot of noise at the input layer of NN.

Conclusions

Separate10only models with S, RN, and CN normalization show better results than models with one-hot initialization. Normalization is a promising technique that needs further investigation.

3.5.7 Change of learning rate hyper-parameter

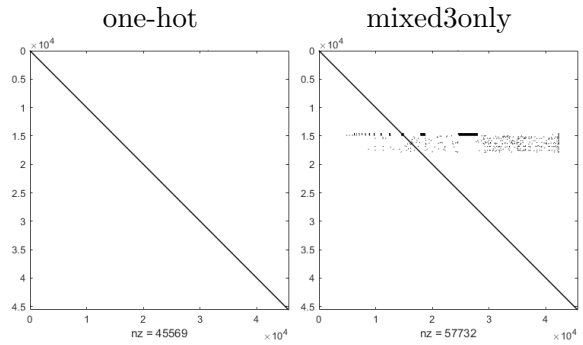


Figure 3.6: One-hot and mixed3onlyW0.1noo10 distributional representations as in Table 3.8.

	learning rate				
	1	0.5	0.1	0.05	0.01
one-hot	172.46(4e, .02)	171.68(2e, .06)	**171.11(3e, .08)**	171.61(13e, .06)	172.48(50e, .04)
mixed3only	172.17(3e, .04)	171.41(2e, .06)	**170.84(3e, .08)**	171.31(11e, .06)	172.50(50e, .04)

Table 3.8: Perplexity of vLBL models with one-hot and mixed3onlyW0.1nn10 initializations, trained with different initial learning rates, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Models that perform the best in each row are **highlighted**. The best performance for one-hot model is reached with learning rate 0.1.

Description

Hyper-parameter settings usually have a great influence on models performances. Good choice of them can lead to a faster convergence and better overall results. There are advanced techniques to perform the hyper-parameter optimization, such as exhaustive grid search and manual search (e.g., Larochelle et al. (2007), Hinton (2010)), random search (Bergstra and Bengio, 2012), bayesian optimization (Snoek et al., 2012).

In our case, for the learning rate hyper-parameter, we decide to explore performance of models for several commonly used values and then pick the best value for our further experiments. To do so, we train LMs with one-hot and mixed3only initializations with learning rate set to {1.0, 0.5, 0.1, 0.05, 0.01} and analyze the results [5].

Distributional representations

We use one-hot and mixed3onlyW0.1noo10 distributional models. For mixed3only model, distributional representations are constructed for words with frequency 3 as described in Section 3.3.2.2. Parameters are set as follows: SIMILARITY_WEIGHT = {1 for diagonal elements, 0.1 for non-diagonal elements}; NUMBER_OF_ONES = 10. <UNK> representation is averaged for mixed models, and onehot for one-hot models.

Used sparse distributional matrices are shown in Figure 3.6.

LBL training

We train vLBL models for 10 epochs for learning rate $\in$ {1.0, 0.5, 0.1}, and for 50 epochs for learning rate $\in$ {0.05, 0.01}.

Results

Results are presented in Table 3.8. Both one-hot and mixed3only models perform better with initial learning rate of 0.1. For learning rate 0.01, we stop training after 50 epochs. Perplexity of this model is going down very slow, but is still higher than for other learning rates, so we decided not to train models till convergence.

Discussion

Choice of appropriate initial learning rate is very important for NN training. In case learning rate is very small, convergence is slow and can take a lot of time as we have seen in case of learning rate 0.05 and 0.01. High initial learning rate can slow down the training by bringing the model far away from an optimal solution at the very beginning; probably,

[5] At the same time, new implementation of training method is introduced with main changes in batches population and iterations through the epochs. These changes do not influence on models architecture. The changes in the implementation and learning rate lead to slight changes in the model performances, as expected, that makes us to re-train some models to make the evaluation comparison valid. We are going to report the changed performance in the next sections.

this is what happens with learning rate 0.5 and 1. It is important to note that different models have different range of suitable learning rates that needs to be investigated. Our investigation shows that for our experiments the optimal learning rate is 0.1.

Conclusions

Best results are obtained for learning rate = 0.1, so we keep this value for further training.

We also changed number of training epochs to 10 as we found out that models do usually converge much faster.

With the change of learning rate, baseline is also changed: one-hot model perplexity is 171.11(3e, .08), mixed3only model perplexity is 170.84(3e, .08). Since learning rate is an important hyper-parameter that can severely influence the performance, in order to make the comparisons valid we re-train some of the previous models. We are going to refer to the previous models and experiments as "old (lr=0.5)", and to new ones as "new (lr=0.1)".

3.5.7.1 Re-training of positioned PPMI mixed models with interval θ

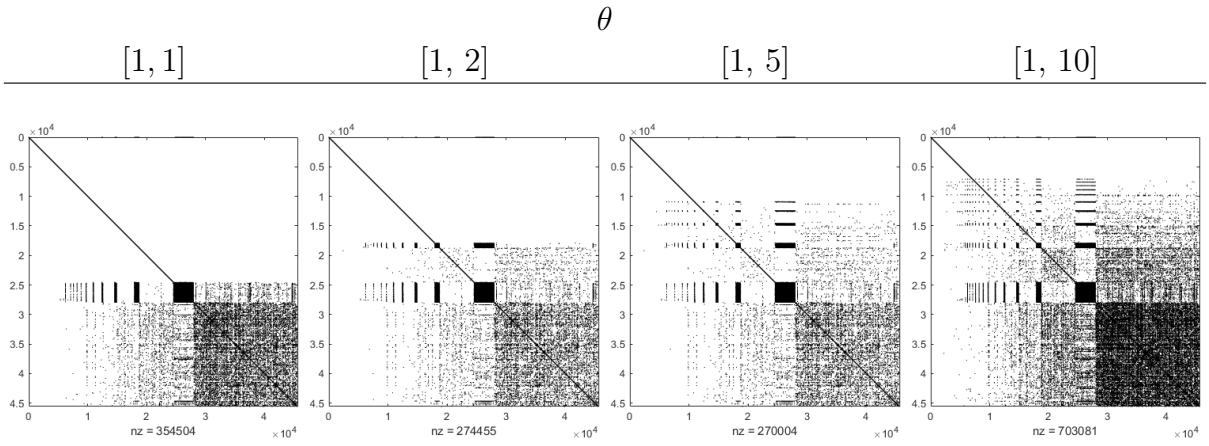


Figure 3.7: Positioned PPMI mixed distributional representations for interval θ as in Table 3.9.

one-hot	θ			
	[1, 1]	[1, 2]	[1, 5]	[1, 10]
171.11(3e, .08)	171.18(3e, .06)	171.18(3e, .06)	171.28(3e, .06)	171.35(3e, .06)

Table 3.9: Perplexity of vLBL models with one-hot and positioned PPMI mixed initializations with interval θ values, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. All models perform worse than one-hot baseline (171.11), same as in Section 3.5.4.

Description

In this section, we report results of re-training LMs with positioned PPMI mixed distributional representations of words with frequencies in θ from {[1, 1], [1, 2], [1, 5], [1, 10]}.

Distributional representations

Distributional models creation is described in Section 3.3.2.2. Parameters of mixed models are set as follows: SIMILARITY_WEIGHT = {1 for diagonal elements, 0.1 for non-diagonal elements}, NUMBER_OF_ONES = 10. <UNK> representation is averaged.

Sparse distributional matrices are shown in Figure 3.7.

LBL training

vLBL training is performed for 10 epochs, learning rate is set to 0.1.

Results

Results are presented in Table 3.9. Same as in Section 3.5.4, all positioned PPMI mixed models for different θ intervals perform worse than one-hot baseline.

Discussion

The performance of mixed models slightly differs for analyzed intervals of θ, but is still worse than one-hot model. The suggestion for further examination here is the same as in Section 3.5.4: perform fine-grained analysis with ONLY models.

Conclusions

In order to investigate reasons of poor performance of positioned PPMI mixed models with interval θ, we need to perform fine-grained analysis with ONLY models: this is done in Section 3.5.7.2 and Section 3.5.7.3.

3.5.7.2 Re-training of separate normalized ONLY models

Description

Since learning rate is changed, we retrain several separate ONLY models.

Distributional representations

For separate ONLY models, we decided to re-train the best model for θ values from {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} ($\theta = 4$), one model on the investigation interval border ($\theta = 10$) and one outside of the interval ($\theta = 20$). Normalization space is widely explored:

- CN-none and CN with coefficients from {1, 0.5, 0.1, 0.05, 0.01} is applied;

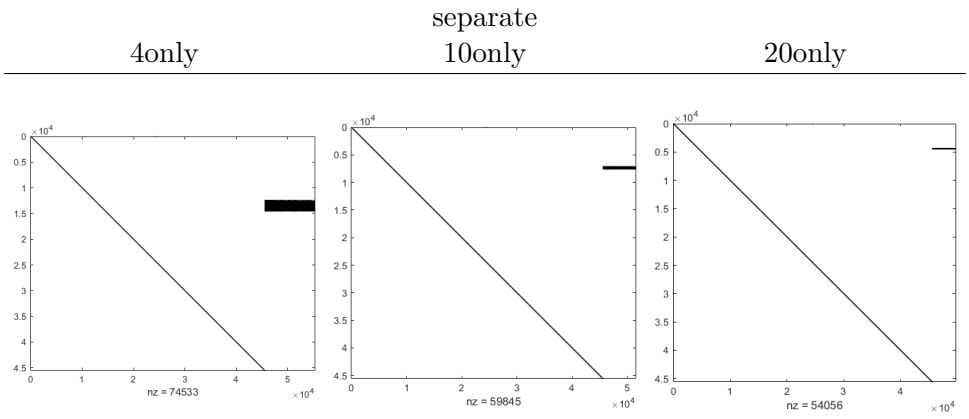


Figure 3.8: Separate ONLY distributional representations as in Table 3.10. Matrices for different normalization schemes look the same since only amplitude of values is different: zero and non-zero values remain their property. Therefore only one matrix for each θ is shown.

- RN is applied with coefficients from {1, 0.5, 0.1};
- S is applied with with coefficients 6, 12, or 60 – to scale values to the intervals [0,1], [0, 0.5], and [0, 0.1] respectively.

<UNK> representation is averaged. Sparse distributional matrices are shown in Figure 3.8.

LBL training

vLBL training is performed for 10 epochs, learning rate is set to 0.1.

Results

Results are presented in Table 3.10. S normalized separate4only models with S = 6 and 12 perform better than one-hot baseline; other examined models perform worse. Normalized separate models work better than models without normalization.

Discussion

Use of normalization. Separate 4only, 10only and 20only models with normalized vectors work better than corresponding models without normalization. This suggests limitations that are put by normalization are beneficial for separate distributional models.

CN, RN, and S comparison. There is no big difference in perplexity results for different normalization techniques applied to separate 10only and separate 20only models. For separate 4only, models with S normalization perform a bit better than models with RN normalization (from .05 to .21 perplexity points), which in their turn perform slightly better than models with CN normalization (from .07 to .28 perplexity points); though these differences are not conclusive.

normalization schemes	coeffs	separate 4only	separate 10only	separate 20only
CN	–	**171.11(5e, .06)**	171.23(5e, .06)	171.28(5e, .06)
	1	171.42(5e, .06)	171.23(5e, .06)	171.24(5e, .06)
	0.5	171.42(5e, .06)	171.22(5e, .06)	171.19(5e, .06)
	0.1	171.30(5e, .06)	171.22(5e, .06)	171.19(5e, .06)
	0.05	171.28(5e, .06)	171.22(5e, .06)	171.20(5e, .06)
	0.01	171.23(5e, .06)	171.22(5e, .06)	171.22(5e, .06)
RN	1	171.19(5e, .06)	171.24(5e, .06)	171.15(5e, .06)
	0.5	171.14(5e, .06)	171.21(5e, .06)	171.20(5e, .06)
	0.1	171.23(5e, .06)	171.22(5e, .06)	171.23(5e, .06)
S	6	**171.07(5e, .06)**	171.19(5e, .06)	171.14(5e, .06)
	12	**170.93(5e, .06)**	171.22(5e, .06)	171.16(5e, .06)
	60	171.18(5e, .06)	171.23(5e, .06)	171.17(3e, .06)
no normalization		171.42(7e, .06)	171.68(6e, .06)	171.78(8e, .06)

Table 3.10: Perplexity of vLBL models with separate ONLY initialization, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Normalization schemes: *CN* – column normalization, *RN* – row normalization, *S* – scale normalization (see Section 3.3.2.3). Models that perform better than one-hot baseline (171.11) are **highlighted**. Models that use normalization perform better than models without normalization, though still mostly worse than one-hot baseline.

The same argument as in Section 3.5.6.2 is valid here: Models with RN normalization work a bit better than models with CN normalization. The reason may be that RN normalization intensifies words' identity signal against the background activation of the connections between words. CN normalization, on other hand, can end up assigning for different words more or less the same level of activation among different dimensions, that confuses language model.

CN_coeffs. As we observed earlier in Section 3.5.6.2, the performance of CN models goes up when the upper limit of the normalization interval goes down: for separate 4only and 10only models. This can be due to the reduction of ambiguity in the word vectors. For separate20only model, the performance is stable among different coefficients.

CN-none. Model with normalization where entries of distributional vectors were divided by the frequency of context performs worse than one-hot baseline. The reason can be that, despite of scaling of frequent contexts, some values in the vector still are several times higher than diagonal elements, that introduces a lot of noise at the input layer of NN.

RN_coeffs. There is no big difference in performance of RN models for different coefficients for all 3 values of θ explored.

S_coeffs. There is no big difference in performance of S models for different coefficients for

separate 10only and 20only models. For separate4only model, the variance is higher.

Performance depending on learning rate. There is a difference in performance for separate models with respect to one-hot baseline for models trained with learning rate 0.5 (Section 3.5.6.2) and models trained in this section, with learning rate 0.1: the former models perform slightly better than the baseline, and the latter perform slightly worse. In both cases, the difference with the baseline is quite small (less than .25 perplexity points), that makes these results inconclusive.

Conclusions

LMs where distributional representations is constructed with normalization tend to perform better than LMs where representations without normalization are used, though more exploration of normalization schemes is needed in order to beat one-hot baseline.

3.5.7.3 Re-training of mixed ONLY models

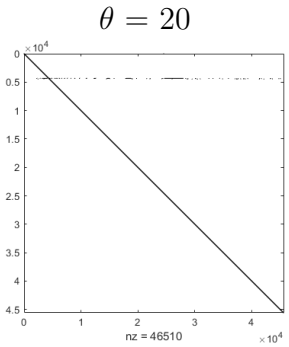


Figure 3.9: MixedW0.1noo10 distributional representations as in Table 3.11. Given matrix corresponds to $\theta = 20$. The matrices for θ from 1 to 10 are the same as in Figure 3.3, left column.

Description

Since learning rate is changed, we retrain several mixed ONLY models.

Distributional representations

For mixed models, we decided to re-train ONLY models for $\theta \in [1, 10]$ and one extra model with $\theta = 20$. Distributional representation are built as described in Section 3.3.2.2 with following parameters: SIMILARITY_WEIGHT ={1 for diagonal elements, 0.1 for non-diagonal elements}, NUMBER_OF_ONES = 10. <UNK> representation is averaged.

Sparse distributional matrices are shown in Figure 3.9.

θ	mixed
1	171.18(3e, .06)
2	**170.87(3e, .08)**
3	**170.84(3e, .08)**
4	**170.93(5e, .06)**
5	**170.95(3e, .08)**
6	**170.91(5e, .08)**
7	**170.90(3e, .08)**
8	**170.93(3e, .08)**
9	**170.85(3e, .08)**
10	**170.90(5e, .08)**
20	**170.86(3e, .08)**

Table 3.11: Perplexity of vLBL models with mixedW0.1noo10 ONLY initializations, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. All models except for one with $\theta = 1$ perform better than one-hot baseline (171.11); this suggests importance of normalization and different treatment for words with frequency 1.

LBL training

vLBL training is performed for 10 epochs with learning rate set to 0.1.

Results

Results are presented in Table 3.11. Mixed ONLY models with θ from 2 to 10 and 20 perform better than one-hot model.

Discussion

We observe a similar behavior to the experiment from Section 3.5.5: all models for $\theta \in [2, 10]$ and $\theta = 20$ perform better than one-hot model; also their performances are indistinguishable. Mixed model for $\theta = 1$ performs worse than one-hot model and models with other θ values. Such behavior suggests future research for different treatment of words with frequency 1 and words with other frequencies.

3.5.8 NUMBER_OF_ONES for mixed3only model

NOO=10	NOO=30
170.84(3e, .08)	**170.95(7e, .06)**

Table 3.12: Perplexity of vLBL models with mixed3onlyW0.1 initialization with different NUMBER_OF_ONES values, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Both models perform better than one-hot baseline (171.11), and model with NOO=10 performs better than with NOO=30.

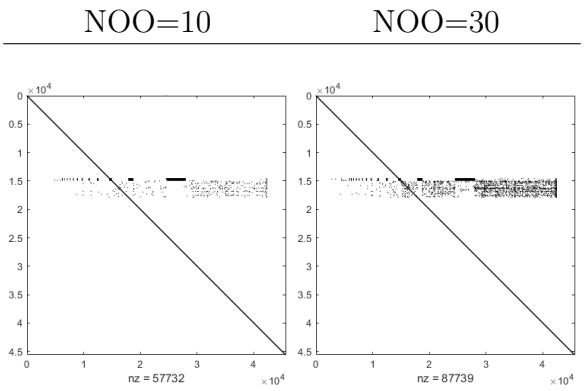


Figure 3.10: Mixed3onlyW0.1 distributional representations for different values of parameter NUMBER_OF_ONES as in Table 3.12.

Description

In the current experiment, we decided to investigate the effect of NUMBER_OF_ONES parameter on the mixed ONLY model performance.

Distributional representations

Mixed3only models are trained with parameter NUMBER_OF_ONES set to 10 or 30. Other parameters are set as follows:

- SIMILARITY_WEIGHT = {1 for diagonal elements, 0.1 for non-diagonal elements}.
- <UNK> representation is averaged.

Sparse distributional matrices are shown in Figure 3.10. There is approximately 30,000 more non-zero values in the distributional matrix for NOO=30.

LBL training

vLBL training is performed for 10 epochs, learning rate is set to 0.1.

Results

Results are presented in Table 3.12. Both mixed models with NUMBER_OF_ONES = 10 and 30 perform better than one-hot model. Model with NUMBER_OF_ONES = 10 performs better than one with 30.

Discussion

The same argument as in Section 3.5.4 is valid here: when NOO is set to 30 the values of subsidiary signals from context word embeddings is getting higher, hence the input to a NN becomes more noisy. The increase of ambiguity worsens the performance of LM.

Conclusions

For the future experiments, we decide to set parameter NUMBER_OF_ONES to 10.

3.5.9 Letter 3-gram models

In the next set of experiments, we use distributional representations that are built by means of letter 3-grams as described in Section 3.3.2.2. We explore mixed and separate models, initializing words with frequency in intervals θ, together with θ ONLY models where words with frequency = θ receive distributional initialization. Performances of models with different normalization schemes is also investigated.

3.5.9.1 Letter 3-gram models with interval θs

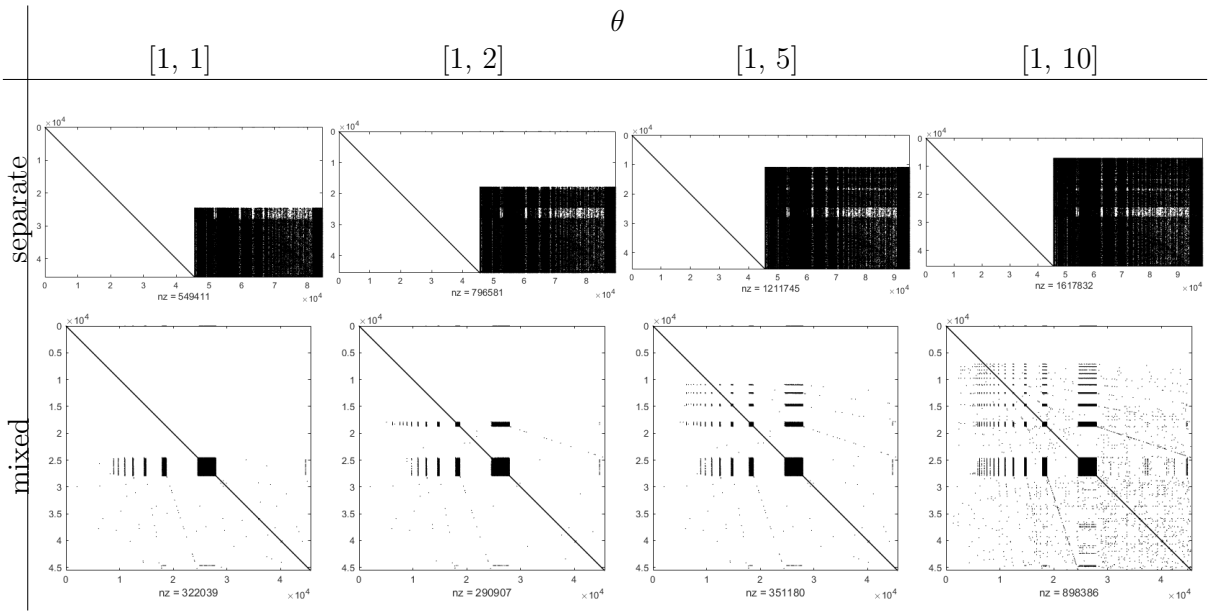


Figure 3.11: Mixed and separate normalized letter 3-gram distributional representations for words with frequencies in intervals θ as in Table 3.13. Separate matrices have the same structure independently of used normalization scheme.

Description

In the first experiment with letter 3-grams, we investigate the performance of LMs initialized with letter 3-gram distributional representations with interval θ from {[1, 1], [1, 2], [1, 5], [1, 10]}.

		coeffs	θ [1, 1]	[1, 2]	[1, 5]	[1, 10]
separate	CN	1	172.04(7e, .04)	171.95(6e, .04)	172.03(6e, .04)	171.72(4e, .04)
		0.5	172.07(7e, .04)	171.87(5e, .04)	171.81(3e, .04)	172.08(6e, .04)
		0.1	171.68(5e, .06)	171.64(7e, .04)	171.90(5e, .04)	171.64(7e, .06)
	RN	1	172.10(7e, .04)	172.01(5e, .04)	172.02(5e, .04)	171.93(5e, .04)
		0.5	172.05(5e, .04)	171.99(5e, .04)	172.04(4e, .04)	172.04(4e, .04)
		0.1	171.90(4e, .04)	171.87(4e, .04)	171.94(4e, .04)	171.90(4e, .04)
	S	6	171.96(7e, .04)	172.15(7e, .04)	172.41(6e, .04)	171.93(7e, .04)
		12	171.91(6e, .04)	171.93(5e, .04)	172.22(6e, .04)	172.05(3e, .04)
		60	171.56(3e, .06)	171.62(4e, .04)	171.76(3e, .06)	171.44(3e, .06)
	mixed		171.23(5e, .06)	171.30(3e, .06)	171.17(4e, .06)	171.24(6e, .06)

Table 3.13: Perplexity of vLBL models with mixedW0.1noo10 and separate letter 3-gram initialization with different normalization schemes, interpolated with KN3. Normalization schemes: *CN* – column normalization, *RN* – row normalization, *S* – scale normalization (see Section 3.3.2.3). In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. All models perform worse than one-hot baseline (171.11). Detailed analysis with ONLY models is needed.

Distributional representations

We build mixed distributional representations of words as described in Section 3.3.2.2. Parameters are set as follows:

- SIMILARITY_WEIGHT = {1 for diagonal elements, 0.1 for non-diagonal elements};
- NUMBER_OF_ONES = 10;
- PPMI values are used;

<UNK> representation is averaged.

Separate distributional representations are built as described in Section 3.3.2.2 with different normalization schemes (see Section 3.3.2.3 for details) and with different coefficients: CN and RN coefficients are in {1, 0.5, 0.1}, and S coefficients are in {6, 12, 60}.

Sparse distributional matrices are shown in Figure 3.9.

LBL training

vLBL training is performed for 10 epochs, learning rate is set to 0.1.

Results

Results are presented in Table 3.11. All models with interval θ perform worse than one-hot baseline.

Discussion

θ values. Performance of mixed and separate models for different interval θs is worse than one-hot baseline, and also not stable. The conclusion here would be the same as in earlier related experiments (Section 3.5.3, Section 3.5.4, Section 3.5.7.1): fine-grained analysis with ONLY models is needed.

Separate models with different normalization schemes. All separate models with different normalization schemes perform similarly to each other on different interval θs.

Normalization coefficients. All models, except of CN on $\theta = [1, 5]$, with upper bound 0.1 (CN_coeff = RN_coeff = 0.1, S_coeff = 60) perform slightly better than models with other coefficients. This supports our hypothesis that activation of non-diagonal elements should be bounded in order to emphasize identity signal on the diagonal.

Conclusions

Fine-grained analysis with ONLY letter 3-gram models is needed.

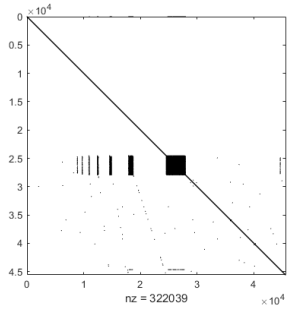
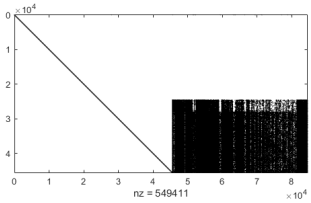
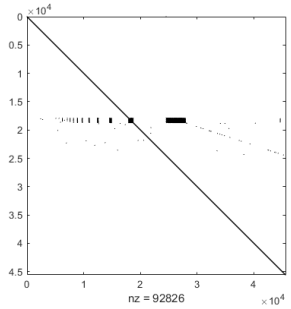
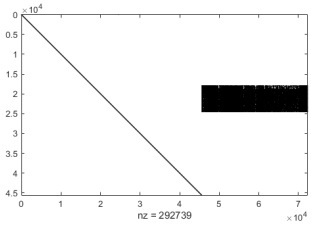
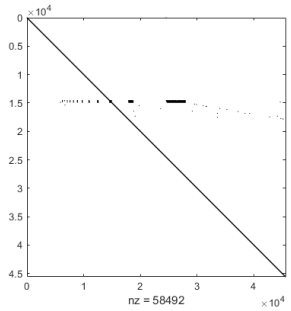
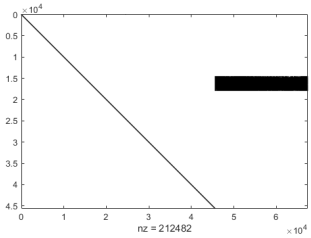
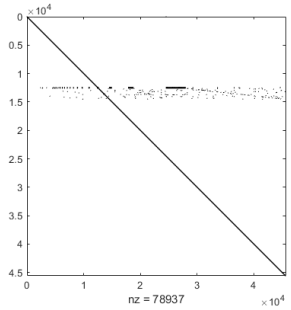
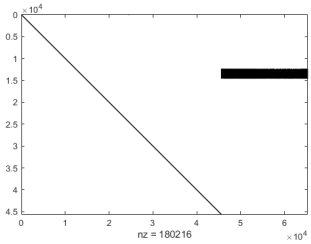
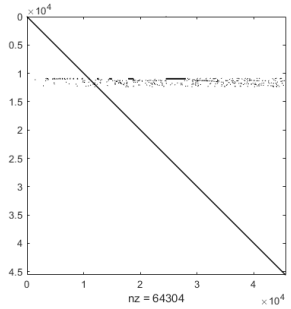
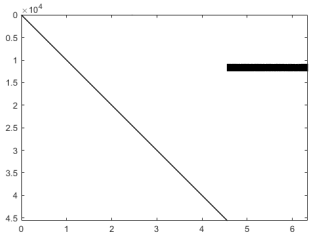
3.5.9.2 Letter 3-gram ONLY models

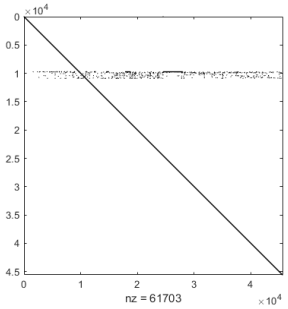
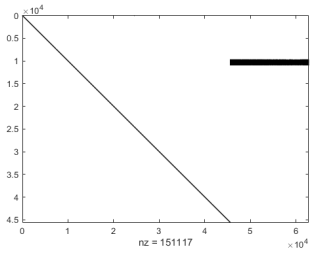
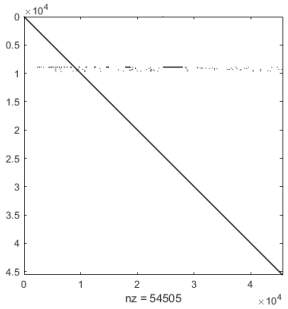
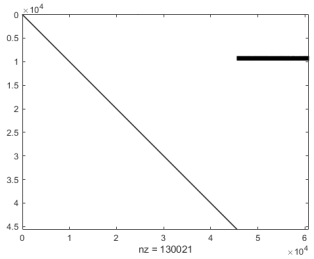
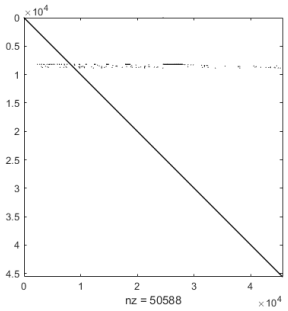
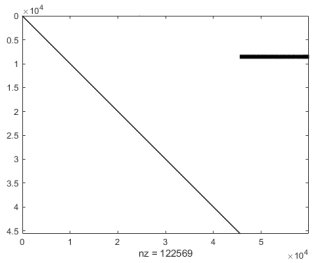
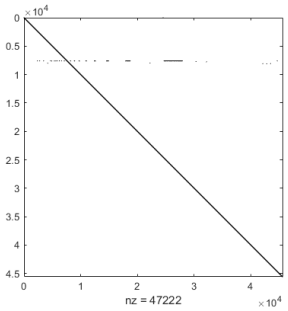
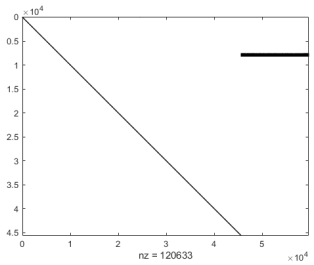
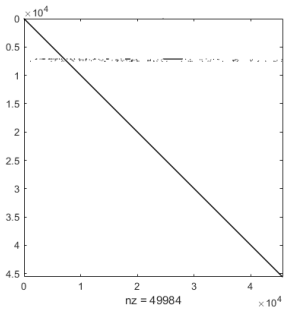
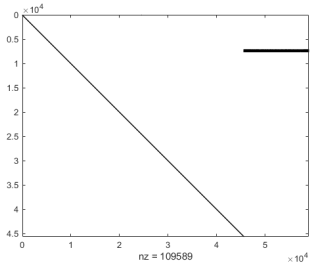
θ	mixed
1	171.23(5e, .06)
2	**170.88(5e, .08)**
3	**170.85(5e, .08)**
4	**170.93(5e, .08)**
5	**170.93(3e, .08)**
6	**170.79(5e, .08)**
7	**170.87(5e, .08)**
8	**170.88(5e, .08)**
9	**170.85(5e, .08)**
10	**170.90(5e, .08)**
20	**170.85(3e, .08)**

Table 3.14: Perplexity of vLBL models with mixedW0.1noo10 letter 3-gram ONLY initialization, where words with frequency = θ receive distributional representations, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Models that perform better than one-hot baseline (171.11) are **highlighted**.

Description

Next we would like to look on the performance of ONLY mixed and separate models to assess the influence of letter 3-gram distributional initialization of words with frequency

θ	mixed	separate
1		
2		
3		
4		
5		

θ	mixed	separate
6		
7		
8		
9		
10		

separate CN	coefficient		
θ	1	0.5	0.1
1	172.04(7e, .04)	172.07(7e, .04)	171.68(5e, .06)
2	171.47(5e, .06)	171.60(5e, .06)	171.33(5e, .06)
3	171.46(5e, .06)	171.27(5e, .06)	171.33(5e, .06)
4	**170.93(5e, .08)**	171.15(5e, .06)	171.30(5e, .06)
5	171.50(5e, .06)	171.25(5e, .06)	171.28(5e, .06)
6	**170.86(5e, .08)**	171.20(5e, .06)	171.34(5e, .06)
7	171.34(5e, .06)	171.27(5e, .06)	171.22(5e, .06)
8	171.72(5e, .06)	171.34(5e, .06)	171.35(5e, .06)
9	171.39(5e, .06)	171.32(5e, .06)	171.22(5e, .06)
10	171.35(5e, .06)	171.26(5e, .06)	171.24(5e, .06)
20	171.54(5e, .06)	171.26(5e, .06)	171.21(5e, .06)

separate RN	coefficient		
θ	1	0.5	0.1
1	172.10(7e, .04)	172.05(5e, .04)	171.90(4e, .04)
2	171.30(5e, .06)	171.17(5e, .06)	171.22(5e, .06)
3	171.34(5e, .06)	171.40(5e, .06)	171.34(5e, .06)
4	171.19(5e, .06)	171.20(5e, .06)	171.23(5e, .06)
5	171.28(5e, .06)	171.25(5e, .06)	171.24(5e, .06)
6	171.34(5e, .06)	171.37(5e, .06)	171.31(5e, .06)
7	171.21(5e, .06)	171.22(5e, .06)	171.22(5e, .06)
8	171.34(5e, .06)	171.32(5e, .06)	171.34(5e, .06)
9	171.23(5e, .06)	171.24(5e, .06)	171.23(5e, .06)
10	171.24(5e, .06)	171.22(5e, .06)	171.22(5e, .06)
20	171.18(5e, .06)	171.20(5e, .06)	171.22(5e, .06)

separate S	coefficient		
θ	6	12	60
1	171.96(7e, .04)	171.91(6e, .04)	171.56(3e, .06)
2	171.31(5e, .06)	171.19(5e, .06)	171.31(5e, .06)
3	171.35(5e, .06)	171.21(5e, .06)	171.36(5e, .06)
4	**171.05(5e, .06)**	171.14(5e, .06)	171.17(5e, .06)
5	171.50(5e, .06)	171.39(5e, .06)	171.24(5e, .06)
6	171.20(5e, .06)	**170.99(5e, .06)**	171.38(5e, .06)
7	171.39(5e, .06)	171.29(5e, .06)	171.24(5e, .06)
8	171.68(5e, .06)	171.30(5e, .06)	171.35(5e, .06)
9	171.37(5e, .06)	171.33(5e, .06)	171.24(5e, .06)
10	171.40(5e, .06)	171.22(5e, .06)	171.25(5e, .06)
20	171.41(5e, .06)	171.24(5e, .06)	171.19(5e, .06)

Table 3.15: Perplexity of vLBL models with separate letter 3-gram ONLY initialization (where words with frequency = θ receive distributional representations) with different normalization schemes, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Normalization schemes: *CN* – column normalization, *RN* – row normalization, *S* – scale normalization (see Section 3.3.2.3). Models that perform better than one-hot baseline (171.11) are **highlighted**.

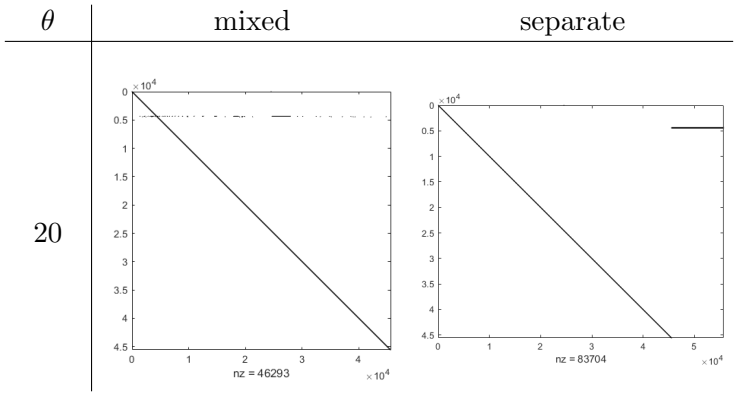


Figure 3.12: Initialization matrices for mixed and separate letter 3-gram ONLY models, where words with frequency = θ receive distributional representations. Matrices correspond to Table 3.14 (mixed column) and Table 3.15 (separate column).

equal θ on LM performance.

Distributional representations

Distributional representations are built as described in Section 3.3.2.2 and Section 3.3.2.1.

Parameters for mixed distributional representations are set as follows:

- SIMILARITY_WEIGHT = {1 for diagonal elements, 0.1 for non-diagonal elements},
- NUMBER_OF_ONES = 10.

For separate representations, we use CN, RN, and S normalization schemes with coefficients in {1, 0.5, 0.1} for CN and RN and in {6, 12, 60} for S.

<UNK> representation is averaged.

Sparse distributional matrices are shown in Figure 3.12.

LBL training

vLBL training is performed for 10 epochs, learning rate is set to 0.1.

Results

Results are presented in Table 3.14 and Table 3.15. In general, mixed models perform good (models with θ from {2, 3, 4, 5, 6, 7, 8, 9, 10, 20} are better than one-hot baseline), separate models perform poorly.

Discussion

Mixed models with θ from {2, 3, 4, 5, 6, 7, 8, 9, 10, 20} perform better than one-hot baseline; and performance is almost the same for different values of θ. It seems like the

main complication for the models with interval θs are words with frequency 1. This result, same as for positioned PPMI models, suggests different treatment for words with different frequencies.

Separate models. In all explored space of normalization schemes with different coefficients, only 4 models show better performance than one-hot baseline: separate 4only with CN_coeff = 1 and S_coeff = 6, and separate 6only with CN_coeff = 1 and S_coeff = 12.

1 ONLY models. Models where only words with frequency equals 1 receive distributional representation perform worse than any other ONLY models – for all normalization schemes, for all explored coefficients, for both mixed and separate models. This could be a reason interval models perform poorly: they include words with frequency = 1 in their receiving distributional initialization words set.

3.5.10 Experiments on preprocessed corpus

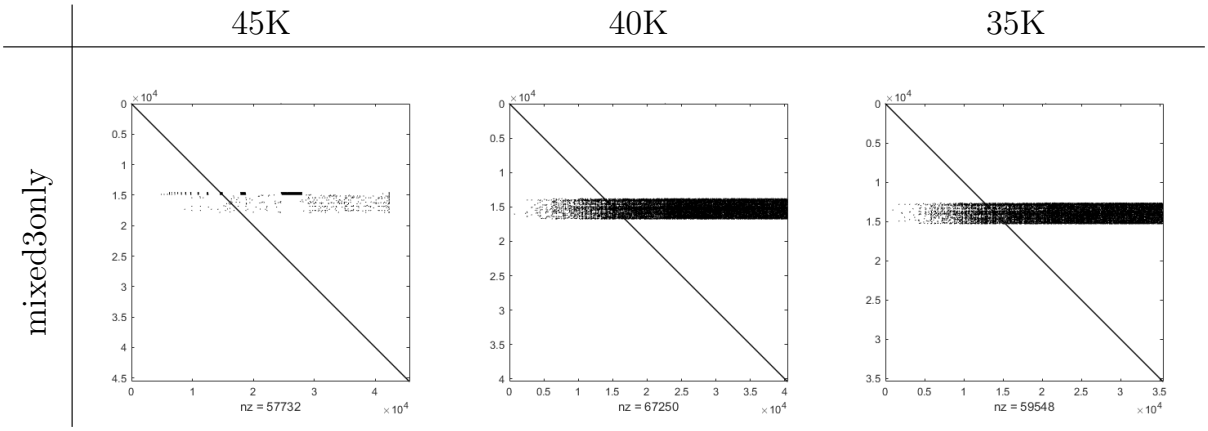


Figure 3.13: Mixed3onlyW0.1noo10 distributional representations as in Table 3.16. Only words with frequency 3 receive distributional representations.

	45K	40K	35K
onehot	171.11(3e, .08)	**162.29(5e, .12)**	**160.37(5e, .14)**
mixed3only	**170.84(3e, .08)**	163.11(4e, .12)	160.92(5e, .14)

Table 3.16: Perplexity of vLBL models with one-hot and mixed3onlyW0.1noo10 initializations with vocabularies for different preprocessed corpora, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Models that perform better in each row are **highlighted**. Better performance of mixed model for unchanged corpus (the largest vocabulary) suggests potential benefit in using distributional initialization on larger corpora.

Description

In current experiment, we want to investigate the performance of LM in case preprocessed corpus is used for its training and for creation of distributional representations. We apply 2 preprocessing strategies:

- replace all digits with one token: this leads to a target vocabulary of size 40,370 (40K);
- replace all digits with one token and lowercase all words: this leads to a target vocabulary of size 35,385 (35K).

Distributional representations

We train 2 models – one-hot and mixed3only – for each corpus version. Sparse distributional matrices are shown in Figure 3.13.

For mixed distributional representations, we set SIMILARITY_WEIGHT = {1 for diagonal elements, 0.1 for non-diagonal elements}, NUMBER_OF_ONES = 10. <UNK> representation is averaged for mixed models, and is one-hot for one-hot model.

LBL training

vLBL training is performed for 10 epochs, learning rate is set to 0.1.

Results

Results are presented in Table 3.16. Mixed3only model outperforms one-hot on unchanged corpus. For preprocessed corpora, one-hot initialization works better.

Discussion

Though the one-hot model outperforms mixed model on preprocessed corpora, better results of mixed distributional initialization on the unchanged corpus can mean better scores of models with distributional initialization on larger corpora.

3.6 Discussion

In the course of experiments, we compare performance of models with distributional initialization with one-hot initialized model. Here we would like to compare LM performances with respect to hyper-parameters of distributional representations: different combination, normalization and association measurement schemes.

3.6.1 Combination schemes

3.6.1.1 Mixed vs. Separate initialization

In contrast to Chapter 2, it would be incorrect to directly compare performance of LMs that employ mixed and separate initialization due to the fact that we construct distributional vectors differently: for separate models, distributional vectors are just concatenations of positioned context vectors, while for mixed models, distributional vectors are built based on similarity of the concatenated positioned vectors. We also applied different normalization schemes for mixed and separate models with different hyper-parameters.

If we ignore these facts and look on the numbers different separate and mixed models achieved for the same upper bound on the non-diagonal elements (1, 0.5 or 0.1), we can see that in most of the cases separate models perform poorer than mixed models. The possible reasons could be that:

1. mixed models enjoy connections between words provided by distributional representations – through sharing the vectors space; whereas word representation vectors for words with distributional and one-hot representation in separate models live in orthogonal spaces,

2. the hyper-parameter space for separate representations needs more exploration in order to tune the parameters well.

3.6.1.2 Mixed and separate initializations vs. ONLY initialization

It is not possible to directly compare performance of ONLY models and mixed or separate models as words that receive distributional initialization are different[6].

3.6.2 Association measurement schemes

We examine LMs with different association measurement schemes: binary, positioned PPMI and letter 3-grams, with respect to old/new learning rates.

3.6.2.1 Binary, positioned PPMI and letter 3-gram models for interval θs

For $\theta \in \{[1, 1], [1, 2], [1, 5], [1, 10]\}$, we would like to compare performance of following models: binary and positioned PPMI mixed with old learning rate value (Section 3.5.3 and Section 3.5.4), positioned PPMI and letter 3-gram mixed with new learning rate value (Section 3.5.7.1 and Section 3.5.9.1).

We summarize these performances in Table 3.17.

Binary and positioned PPMI comparison (lines 1 and 2 of Table 3.17). On average, performance of binary and positioned PPMI mixed models for interval θ drops by 2.71 points,

[6]Except of ONLY models with $\theta = 1$ and models with $\theta = [1, 1]$ that have exactly the same distributional matrices, therefore this comparison is unnecessary.

mixed models	learning rate	window size	θ [1, 1]	[1, 2]	[1, 5]	[1, 10]	one-hot
binary –	old(0.5)	10+10	173.84(2e, .1)	175.18(4e, .1)	175.59(16e, .1)	175.94(20e, .1)	171.90(2e, .1)
p-ppmiW0.1noo10	old(0.5)	2+2	172.40(2e, .1)	172.49(2e, .1)	172.26(2e, .1)	172.45(2e, .1)	171.90(2e, .1)
p-ppmiW0.1noo10	new(0.1)	2+2	171.18(3e, .06)	171.18(3e, .06)	171.28(3e, .06)	171.35(3e, .06)	171.11(3e, .08)
3-gramW0.1noo10	new(0.1)	2+1+2	171.23(5e, .06)	171.30(3e, .06)	171.17(4e, .06)	171.24(6e, .06)	171.11(3e, .08)

Table 3.17: Perplexity of vLBL models with mixed initializations for interval θ, interpolated with KN3. Baseline performance is reported in the last column. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. *window size* corresponds to a number of words from left and right *(l+r)* of a target words used to build distributional representations; in case of letter 3-gram models (line *3-gram*), window includes also target word in the middle: 2+1+2. *binary* indicates binary models, *p-ppmi* indicates positioned PPMI models, and *3-gram* indicates letter 3-gram models.

with difference growing with interval expansion: $1.44 \rightarrow 2.69 \rightarrow 3.23 \rightarrow 3.49$. There are several possible explanation of such an effect:

- difference in hyper-parameters for binary and positioned PPMI models makes them incomparable, e.g., context size, weighting, NUMBER_OF_ONES value;
- usage of PPMI values in distributional representation construction together with limitation on NUMBER_OF_ONES emphasizes most relevant and discard least relevant information in the words representations, that allows positioned PPMI mixed models outperform binary mixed models.

Both models perform worse than corresponded one-hot baseline.

Positioned PPMI and letter 3-gram comparison (lines 3 and 4 of Table 3.17). Performances of positioned PPMI and letter 3-gram mixed do not differ much from each other for different θ values: the maximum difference is .12 perplexity points. Also both models perform worse than corresponded one-hot baseline. We assume this is due to the resemblance between positioned vectors with PPMI values and letter 3-grams counts that leads to the similar distributional representation of vocabulary. To see the difference, we would suggest to increase the training corpus size.

3.6.2.2 Positioned PPMI and letter 3-gram models for θ ONLY

Mixed and separate distributional representations for different θ ONLY values were tested on LMs, and here we would like to compare positioned PPMI and letter 3-gram schemes.

Mixed models. For mixed models, positioned PPMI representation was introduced in Section 3.5.7.3 and letter 3-gram representation in Section 3.5.9.2. We summarize models performances in Table 3.18.

Same as for models with interval θ, performance of positioned PPMI and letter 3-gram ONLY models do not differ much from each other for different values of θ (maximal delta

θ	mixedW0.1noo10	
	positioned PPMI	letter 3-gram
1	171.18(3e, .06)	171.23(5e, .06)
2	**170.87(3e, .08)**	**170.88(5e, .08)**
3	**170.84(3e, .08)**	**170.85(5e, .08)**
4	**170.93(5e, .06)**	**170.93(5e, .08)**
5	**170.95(3e, .08)**	**170.93(3e, .08)**
6	**170.91(5e, .08)**	**170.79(5e, .08)**
7	**170.90(3e, .08)**	**170.87(5e, .08)**
8	**170.93(3e, .08)**	**170.88(5e, .08)**
9	**170.85(3e, .08)**	**170.85(5e, .08)**
10	**170.90(5e, .08)**	**170.90(5e, .08)**
20	**170.86(3e, .08)**	**170.85(3e, .08)**

Table 3.18: Perplexity of vLBL models with mixedW0.1noo10 positioned PPMI and letter 3-gram ONLY initializations, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Models that perform better than one-hot baseline (171.11) are **highlighted**.

is .12). Suggested explanation is the same as above: positioned PPMI vectors are alike positioned letter 3-gram vectors, that leads to the similar distributional representations of vocabulary words.

Separate models. For separate models, positioned PPMI representation was employed in Section 3.5.7.2 and letter 3-gram representation in Section 3.5.9.2. We summarize models performances in Table 3.19.

Performance of 10 ONLY models does not differ much for positioned PPMI and letter 3-gram representations. Performance of 4 ONLY models is very similar for different distributional representations, with two exceptions: CN normalization with CN_coeff = 1 and S normalization with S_coeff = 12.

3.6.3 Normalization schemes

We use normalization schemes described in Section 3.3.2.3 to scale values in the word representation vectors. The overall finding is that normalization helps distributional models to perform better.

3.6.3.1 Normalization for mixed models

As we mentioned above, for mixed models "to normalize" means to define a value of SIMILARITY_WEIGHT W that is used for a non-diagonal elements in words distributional representations. The diagonal elements were always kept equal to 1.

In Section 3.5.4 and Section 3.5.5 (for mixed10only models with and without weighting),

		separate4only		separate10only	
	coeff	positioned PPMI	letter 3-gram	positioned PPMI	letter 3-gram
CN	1	171.42(5e, .06)	**170.93(5e, .08)**	171.23(5e, .06)	171.35(5e, .06)
	0.5	171.42(5e, .06)	171.15(5e, .06)	171.22(5e, .06)	171.26(5e, .06)
	0.1	171.30(5e, .06)	171.30(5e, .06)	171.22(5e, .06)	171.24(5e, .06)
RN	1	171.19(5e, .06)	171.19(5e, .06)	171.24(5e, .06)	171.24(5e, .06)
	0.5	171.14(5e, .06)	171.20(5e, .06)	171.21(5e, .06)	171.22(5e, .06)
	0.1	171.23(5e, .06)	171.23(5e, .06)	171.22(5e, .06)	171.22(5e, .06)
S	6	**171.07(5e, .06)**	**171.05(5e, .06)**	171.19(5e, .06)	171.40(5e, .06)
	12	**170.93(5e, .06)**	171.14(5e, .06)	171.22(5e, .06)	171.22(5e, .06)
	60	171.18(5e, .06)	171.17(5e, .06)	171.23(5e, .06)	171.25(5e, .06)

Table 3.19: Perplexity of vLBL models with separate 4only and 10only positioned PPMI and letter 3-gram initializations, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Normalization schemes: *CN* – column normalization, *RN* – row normalization, *S* – scale normalization (see Section 3.3.2.3). Models that perform better than one-hot baseline (171.11) are **highlighted**.

we compare performance of W=1 and W=0.1, and found out that models with W=0.1 perform better both for interval and for ONLY θ. Then, in Section 3.5.6.1, we built distributional representations setting W to values from {0.5, 0.1, 0.01, 0.001}, and there we found out that for good performance proper activation at the input layer of NN is crucial: the optimal value should not be very high or very low. In case when it is very high (1 or 0.5), the activation at the input layer becomes very noisy and makes model unable to distinguish different words. In case of very low (0.001) values, the signal of the additional information (non-diagonal elements) is not strong enough to improve the performance.

These findings recommend to set W = 0.1 for the best performance.

3.6.3.2 Normalization for separate models

We explore CN, RN, and S normalization schemes for separate models with different coefficients: {1, 0.5, 0.1, 0.05, 0.01} for CN, {1, 0.5, 0.1} for RN, and {6, 12, 60} for S. We also looked at the CN-none model, where vector's values were divided by the frequency of contexts. We have found that models with normalization perform better than models without normalization, that means that limitations put by normalization are beneficial. On other hand, only a couple of normalized separate models perform better than one-hot baselines, that suggest that careful application of normalization can enhance LM performance.

Comparison of different normalization schemes

We conducted 2 experiments (Section 3.5.6.2, Section 3.5.7.2) on positioned PPMI models and 2 experiments (Section 3.5.9.1, Section 3.5.9.2) on letter 3-gram models.

For positioned PPMI models, RN models slightly surpass CN models for separate10only in Section 3.5.6.2 (differences in [.02, .55]) and for separate4only in Section 3.5.7.2 (differ-

ences in [.07, .28]). Though in later experiment the performance of RN and CN models was indistinguishable for separate 10only and 20only models (differences in [-.01, .09]). The reason that RN models work a bit better than CN models may be that RN normalization intensifies words' identity signal against the background activation of the connection between words. CN normalization, on other hand, can end up assigning for different words more or less the same level of activation among different dimensions, that will confuse language model.

RN models also slightly outperform S models for separate10only in Section 3.5.6.2 (differences in [.02, .13]), but are beaten by S normalization in Section 3.5.7.2 with differences in [.01, .21] for separate 4only and 20only models; for separate10only, the performance is similar.

Performances of CN and S models do not differ much for separate10only in Section 3.5.6.2 (differences in [-.04, .11]) and Section 3.5.7.2 (differences in [-.01, .04]), and for separate20only in the later experiment (differences in [.02, .10]). For separate4only, S outperforms CN models with differences are in [.12, .49].

CN-none model evaluated in Section 3.5.6.2 showed no advantage over CN, RN, and S models for separate10only setting, together with one-hot baseline. In Section 3.5.7.2, performance of CN-none models is indistinguishable from other normalization schemes for separate 10only and 20only models, and slightly better than CN and RN models for separate4only.

For letter 3-gram models, no distinct performance pattern was observed: performance varied from 0 to .48 perplexity points for different normalization schemes and different coefficients with no advantage for a particular parameters choice.

Despite of small improvements observed for different normalization schemes, no conclusion can be drawn from the comparison. On other hand, almost all normalized models performed better than one-hot baseline, that suggests potential for future research here.

3.6.4 Unknown words treatment

We did not evaluate impact of different representations used to represent unknown token <UNK>.

3.6.5 Words frequency range for distributional initialization θ

3.6.5.1 Performance for interval θ

As summarized in Table 3.17 and Table 3.20, both mixed and separate models with explored distributional representations perform poorer that one-hot baseline (see experiments on binary models in Section 3.5.3, on positioned PPMI models in Section 3.5.4 and Section 3.5.7.1, on letter 3-gram models in Section 3.5.9.1). The common suggestion there was to perform fine-grained analysis with ONLY models to see what caused such results. The findings are described in Section 3.6.5.2.

			θ			
		coeff	[1, 1]	[1, 2]	[1, 5]	[1, 10]
letter 3-gram	CN	1	172.04(7e, .04)	171.95(6e, .04)	172.03(6e, .04)	171.72(4e, .04)
		0.5	172.07(7e, .04)	171.87(5e, .04)	171.81(3e, .04)	172.08(6e, .04)
		0.1	171.68(5e, .06)	171.64(7e, .04)	171.90(5e, .04)	171.64(7e, .06)
	RN	1	172.10(7e, .04)	172.01(5e, .04)	172.02(5e, .04)	171.93(5e, .04)
		0.5	172.05(5e, .04)	171.99(5e, .04)	172.04(4e, .04)	172.04(4e, .04)
		0.1	171.90(4e, .04)	171.87(4e, .04)	171.94(4e, .04)	171.90(4e, .04)
	S	6	171.96(7e, .04)	172.15(7e, .04)	172.41(6e, .04)	171.93(7e, .04)
		12	171.91(6e, .04)	171.93(5e, .04)	172.22(6e, .04)	172.05(3e, .04)
		60	171.56(3e, .06)	171.62(4e, .04)	171.76(3e, .06)	171.44(3e, .06)
positioned PPMI			173.10(4e, .1)	173.36(4e, .1)	173.83(5e, .1)	174.38(9e, .1)

Table 3.20: Performance of separate models with interval θ. All models perform worse than one-hot baselines: 171.90 for positioned PPMI models and 171.11 for letter 3-gram models. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Normalization schemes: *CN* – column normalization, *RN* – row normalization, *S* – scale normalization (see Section 3.3.2.3).

For binary mixed models, performance go down with the expansion of θ intervals (see Table 3.17). The possible explanation is that since all values in the distributional representations are equal 1, LM receives more and more ambiguous signals at the input layer with the grow of frequency interval. Indeed, employment of PPMI values and limitation of NOO to 10 to control this activation brings reduction of the perplexity.

For separate positioned PPMI model, performance go down with the expansion of θ (see Table 3.20). The reason can be the same as described above for binary mixed models. The hypothesis that activation of non-diagonal elements should be bounded in some way – either by putting limit on the maximal value or by the number of non-zero values in a vector – to emphasize identity signal on the diagonal is supported also by results for separate letter 3-gram models. As can be seen in Table 3.20, all models except of CN on θ = [1, 5] with maximal value 0.1 (CN_coeff = RN_coeff = 0.1, S_coeff = 60) perform slightly better than models with other coefficients.

3.6.5.2 Performance of ONLY models for different θ

The experiments (for more details see Section 3.5.5, Section 3.5.7.3, Section 3.5.9.2) showed that performance of mixed models for θ from {2, 3, 4, 5, 6, 7, 8, 9, 10, 20} is greater than one-hot baseline with differences up to .46 perplexity points. This finding shows that use of distributional representations for words of determined frequencies is beneficial, and future research on combination of different representations of words with different frequency looks promising.

ONLY models with $\theta = 1$ usually perform worse than ONLY models with other θ values altogether with one-hot baseline model (see first lines in Table 3.5, Table 3.11, Table 3.14,

θ	mixedW0.1noo10 positioned PPMI, old lr=0.5	mixedW0.1noo10 positioned PPMI, new lr=0.1	letter 3-gram
1	172.18(2e, .1)	171.18(3e, .06)	171.23(5e, .06)
2	**171.80(2e, .1)**	**170.87(3e, .08)**	**170.88(5e, .08)**
3	**171.44(2e, .1)**	**170.84(3e, .08)**	**170.85(5e, .08)**
4	**171.66(2e, .1)**	**170.93(5e, .06)**	**170.93(5e, .08)**
5	**171.81(2e, .1)**	**170.95(3e, .08)**	**170.93(3e, .08)**
6	**171.79(2e, .1)**	**170.91(5e, .08)**	**170.79(5e, .08)**
7	**171.84(2e, .1)**	**170.90(3e, .08)**	**170.87(5e, .08)**
8	**171.76(2e, .1)**	**170.93(3e, .08)**	**170.88(5e, .08)**
9	**171.90(2e, .1)**	**170.85(3e, .08)**	**170.85(5e, .08)**
10	**171.71(2e, .1)**	**170.90(5e, .08)**	**170.90(5e, .08)**
2-10	**171.81(2e, .1)**	–	–
20	–	**170.86(3e, .08)**	**170.85(3e, .08)**
one-hot	171.90(2e, .1)	171.11(3e, .08)	171.11(3e, .08)

Table 3.21: Perplexity of vLBL models with mixed ONLY initializations, interpolated with KN3. In parentheses: for the best value, number of the epoch and LBL interpolation weight λ. Models that perform not worse than one-hot baseline (last line) are **highlighted**.

Table 3.15). Such behavior needs more detailed investigation that also can help to improve performance of distributional models with interval θ.

Mixed2-10 and mixed with θ = [1, 10]

It is worth to mention that distributional matrix for $\theta = [1, 10]$ is not just a composition of distributional matrices for mixed2-10 model and mixed model with $\theta = 1$, since the values of the similarity threshold α in Algorithm 1 are different for these matrices. Therewith it is not completely correct to ground the low performance of model with $\theta = [1, 10]$ on the low performance of 1 ONLY model, albeit the perplexity results suggest this prominently. We would like to explore this possibility in the future work.

Separate models

From Section 3.5.5, performance of separate models with S_coeff=12 and S_coeff=60 normalizations is similar for θ from {2, 3, 4, 5, 6, 7, 8, 9, 10} with the biggest difference for models with $\theta = 1$. The same pattern was observed for mixed models.

For Section 3.5.7.2 where θ was set to 4, 10, or 20, we can see that performance of not normalized models is decreasing for θ changing in the order 4, 10, 20. While for normalized models the performance depends on normalization technique chosen: for CN, $\theta = 4$ performs worse than 10 and 20; for RN, models with different θ perform the same; for CN-none and S, $\theta = 4$ performs better than 10 and 20.

Such difference in the performances for combinations of θ and normalization schemes

brings us to the conclusion that performance of ONLY models for different θ mostly depends on other hyper-parameters, and in order to assess it appropriately more experiments needed. Same observation is true for the experiment in Section 3.5.9.2.

3.6.6 Change of learning rate

New learning rate was introduced and used since Section 3.5.7. This change led to the slight changes in the model performances that made us to re-train some models to make the comparison valid. In the analysis of the experiments, we did not compare performance of old and new, re-trained, models.

3.7 Conclusion

In this chapter, we described experiments conducted to test the performance of neural networks initialized with distributional vectors on language modeling task.

We proposed several distributional representations for words employing different combination schemes (mixed, separate, ONLY), different association functions (co-occurrence in a window, with use of positioned PPMI and positioned letter 3-grams), and different normalization schemes (non-diagonal elements were replaced with constant; scaled; rows or columns got normalized with different scaling coefficients).

We explored in detail behavior of LMs where different subsets of words received distributional initialization: parameter θ received values either from the set of intervals {[1, 1], [1, 2], [1, 5], [1, 10], [2, 2], [2, 5], [2, 10], [6, 6], [6, 10]}, or set to a constant value from {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20}.

As a neural language model, we chose vLBL model (Mnih and Kavukcuoglu, 2013) – a modification of log-bilinear language model introduced in (Mnih and Hinton, 2007). After the training, vLBL was interpolated with modified Kneser-Ney model of order 3 and the perplexity results were reported.

From the experiments conducted and analysis made, we can conclude following: Simple binary representations as in Chapter 2 do not work well enough for language modeling task with very simple vLBL model and small training corpus used. The introduced complexity of explored distributional representations did not pay back with strong and stable increase of performance. The improvements that were observed for different settings suggest utilization of mixed initialization with weighting scheme for words with frequencies up to 10 (with careful treatment of words with frequency 1).

To acquire better results on language modeling task, further research is required.

3.8 Future work

There are several ideas that look promising to implement right ahead. Experiments with ONLY models suggest *different treatment of words with frequency 1 and words with other frequencies*, e.g., apply one-hot representations for them in models with interval θ.

As it is mentioned in Section 3.3.6, perplexity is a tool of intrinsic evaluation; what is more important is how the particular LM can influence the performance of a practical system in, for example, machine translation or speech recognition field. That is why the next logical step will be *to embed LMs with one-hot and distributional initializations into such systems and assess the change in the performance.*

The experimental results we got in our work were tied to specific choices we had made beforehand on model architectures and representations used. There is a number of suggestions for extensions that can be applied to the current models.

One of the main changes that, as we hope, is going to make a difference is *utilization of the distributional representations for both input and output of the NN* of LM. Based on the results of Botha and Blunsom (2014), we expect this to make LM predictions more accurate.

We are also interested in *using bigger training corpora* (e.g., ukWac+Wackypedia, as in Chapter 2). Though small corpus makes training faster, more information is needed to model natural language for the real world applications. Complex neural LMs can benefit from usage of bigger training corpus; it also can make the advantages of neural LMs with distributional representations more vivid.

We have chosen simple vLBL as our neural language model mostly for the reason of fast training. Nevertheless we expect *neural LMs with more sophisticated architecture* (e.g., LBL, NNLM, LSTM) to be more suitable for the current task: position-awareness of LBL model and non-linearity of NNLM can play crucial role in tasks of natural language processing. The former takes into account the fact that words in sentences are usually ordered, the later promotes non-linear interactions between words in a sentence.

3.9 Related work

Different ways to improve modern NNLMs have been proposed in the research community. Here we would like to focus on recent works that, similar to our, incorporate extra information as a part of the input to the NNs in order to improve predicted probabilities.

It was shown that incorporation of *syntactic dependencies* in n-gram models (Gubbins and Vlachos, 2013), RNNs (Mirowski and Vlachos, 2015), or LSTMs (Zhang et al., 2016) is helpful for sentence completion task.

Information provided by *social networks* such as users relations and user characteristic topics can become a source of smoothing for trained LMs according to (Yan et al., 2015) and (Tseng et al., 2015).

Incorporation of information about *history context* was investigated in works (Mansur et al., 2013) and (Zhang et al., 2015). Mansur et al. (2013) proposed for prediction of the next word or character to use its history context feature set, that includes different combinations of unigrams and bigrams that are at a distance ≤ 2 from the target element. Zhang et al. (2015) proposed to use as an input of NN a combination of one-hot representation of the current word and fixed-size vector that accumulates the previous history. Though we put emphasis on use of global statistics, these ideas are akin to our work: in both

papers, the information about the previous/surrounding context influences on the word's representation.

One-hot representation of vocabulary words was enriched by concatenation with one-hot *vector of part-of-speech tags* in work of Adel et al. (2013). Such approach showed good results on language modeling for code-switching speech task with RNNs in comparison to 3-gram models, that suggests that sharing part of the representation among words is beneficial.

Kim et al. (2016) explored word representations that are based on the *convolution over embeddings of each character*; these representations are further fed to the input of LSTM for language modeling. Unlike our work, authors used much more complex architecture to construct words representations (CNN) and to train a language model (LSTM).

Probably, the closest to ours is approach of Botha and Blunsom (2014). Based on the idea that morphologically similar words should have close representations, they proposed to represent each word as a *sum of embeddings of its morphemes*. These additive representations were then used in LBL model training, on the input and output layers. Unlike the proposed approach, our models do not require morphological analysis at the pre-processing step.

Chapter 4

Variability of word2vec

4.1 Summary

Lately in the Deep Learning (DL) research field, the word2vec tool (Mikolov et al., 2013a) has become very popular both as a baseline to compare to for a variety of Natural Language Processing (NLP) tasks and as a source of high-quality word embeddings.

In this chapter, we would like to investigate whether the solution that word2vec finds is unique: how much do obtained embeddings differ from each other depending on the number of training epochs and on the random seed used to generate values for initialization of the word embedding matrix.

4.2 Introduction

word2vec was introduced in (Mikolov et al., 2013a). Code availability, the simplicity of usage and speed of training make word2vec very attractive and popular in NLP research community. Baroni et al. (2014) provide a comparison of count-based and context-predicting models on semantic tasks, where among the former models word2vec architectures are evaluated; and based on the results they "would certainly recommend anybody interested in using [distributional semantic models] for theoretical or practical applications to go for the predict models." Researchers present modifications of the tool [Ling et al. (2015a), Le and Mikolov (2014)], use it as a baseline [Schnabel et al. (2015), Liu et al. (2015a), Stratos et al. (2015)], pre-train word embeddings with it [Yu et al. (2015), Yu and Dredze (2015), Qian et al. (2015)], and generate embeddings for lexical units of different granularity [e.g., for word senses (Iacobacci et al., 2015), for synsets and lexemes (Rothe and Schütze, 2015), for word phrases with the word2phrase pre-processing tool[1]].

Though the popularity of word2vec increases, only a limited number of works tackle the question of how word2vec works or analyze the quality of the solutions it finds. In this chapter, we are going to present evaluations of the structure of the learned embeddings

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

spaces with respect to the random seeds used in the initialization of the word embedding matrix. The structure of the embedding space is evaluated by means of a number of common nearest neighbors and correlations of distances between word vectors. The learned semantics is evaluated with a word similarity judgment task. In the experiments, we explore four model types: two basic Neural Network (NN) architectures of word2vec: CBOW and Skip-gram, together with two hyper-parameter settings: hierarchical softmax and negative sampling.

In the rest of this chapter, we describe word2vec hyper-parameters and its random initialization in Section 4.3, followed by Section 2.4 with the description of a proposed modification that allows us to set a random seed for pseudo-random number generator, hyper-parameters settings, corpus and evaluation tasks. Section 4.5 talks about the obtained results. Section 4.6 lists works with focus on the word2vec tool itself. Conclusions and future work are presented in Section 4.7 and Section 4.8.

4.3 Methods

4.3.1 word2vec

We introduced the word2vec tool in Section 2.3.1 of Chapter 2, describing the *continuous bag-of-words model (CBOW)* and the *continuous skip-gram model (Skip-gram)*, together with *hierarchical softmax* approach to estimate predicted probabilities.

Another approach, an alternative to hierarchical softmax, proposed in (Mikolov et al., 2013b) is *negative sampling*. It resembles NCE (see Section 3.3.4.2 for more detailed description of NCE) in the objective it puts on the model: to distinguish the correct prediction from words picked from the noise distribution. According to (Mikolov et al., 2013b), "The main difference between the negative sampling and NCE is that NCE needs both samples and the numerical probabilities of the noise distribution, while negative sampling uses only samples. And while NCE approximately maximizes the log probability of the softmax, this property is not important for [learning high-quality distributed vector representations]."

Use of negative sampling with 5 samples is a default option of word2vec, and we would like to explore this setting in our experiments along with hierarchical softmax.

4.3.2 Random initialization of word2vec

There are several parameters that get their value assigned before or during the training with randomization involved. It is worth to mention that there is no call of any random number generator in word2vec, and when a random number is needed, it is generated in the way depicted in Algorithm 2. This code implements a linear congruential pseudo-random number generator introduced by (Lehmer, 1949) and described in (Knuth, 1997).

Such pseudo-randomization happens in several places in the word2vec code:

- first seed, initial value is 1:

```
// initiate random variable with a given value
next_random = initial_value;
// generate random values
while random value is needed do
    next_random = next_random * (unsigned long long)25214903917 + 11;
    // use generated pseudo-random value next_random
    {...};
end
```

Algorithm 2: Pseudo-random number generation in word2vec.

  - to initialize the matrix of word embeddings;

- second seed, initial value is thread ID:
  - during the subsampling of frequent words;
  - during the training, to choose a context window size for each target word;
  - during negative sampling, to choose indices of words that are used as negative examples;

In this chapter, we experiment only on setting the initial value for random number generation that is involved in the initialization of the word embedding matrix.

Another pieces that lead to different results of word2vec execution are *multithreading* and *asynchronous updates* of the NN weights. Before the training, the whole corpus is divided equally among the threads, and each thread passes through the part it is assigned to for as many epochs as specified. Asynchronous stochastic gradient descent is used for updates, that makes the order of updates depend on the order the treads are evoked and executed. Though there is no lock on the embedding matrix, authors claim that stochastic gradient descent will correct the parameter values during the next steps[2]. Thus such combination of multithreading and asynchronous updates makes the obtained embedding matrices different for different runs even when all hyper-parameters are kept unchanged.

In order to remove this source of randomization, we use 1 thread to train our models, so that all other pseudo-random values that depend on the thread ID are set in the same way and all updates are done in a predictable order.

4.4 Experimental setup

4.4.1 word2vec modification

We modified the word2vec code[3], by introducing an additional parameter *seed* that is then used as an initial value in pseudo-random number generation in initialization of the word

[2] https://groups.google.com/forum/#!topic/word2vec-toolkit/ms94b8b5QQQ

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

```
Input : Vocabulary size vocab_size;
        Size of hidden layer layer1_size;
        Random seed seed;
Variables :
real *syn0 ;                                    // matrix of word embeddings
unsigned long long next_random ;                   // pseudo-random variable

// allocate memory for the word embedding matrix
posix_memalign((void **)&syn0, 128, (long long)vocab_size * layer1_size * sizeof(real));
// initiate random variable with a given seed
next_random = seed;

// initiate word embedding matrix
for (a = 0; a < vocab_size; a++) do
    for (b = 0; b < layer1_size; b++) do
        next_random = next_random * (unsigned long long)25214903917 + 11;
        syn0[a * layer1_size + b] =
        (((next_random & 0xFFFF) / (real) 65536) - 0.5) / layer1_size;
    end
end
Result: matrix of word embeddings.
```

Algorithm 3: Initialization of the word2vec word embedding matrix with a given random seed.

embedding matrix as shown in Algorithm 3.

4.4.2 Model architectures and hyper-parameters

We train word2vec models exploring two architectures: continuous bag-of-words model (CBOW) and Skip-gram, with hierarchical softmax or with negative sampling (with 5 negative examples):

- the hyper-parameter *cbow* is responsible for the model architecture: CBOW ($cbow = 1$) or Skip-gram ($cbow = 0$);
- the hyper-parameter hs is responsible for usage of hierarchical softmax: $hs = 1$ turns it on and $hs = 0$ turns it off; this parameter is complementary to neg;
- the hyper-parameter neg specifies usage of negative sampling: $neg = 0$ turns it off, $neg = k$ turns it on and uses k negative examples.

Model types and hyper-parameter values are summed up in Table 4.1.

Models with different hyper-parameters are incomparable, so we report results for them separately.

model	label	cbow	hs	neg
Skip-gram with negative sampling, k=5	cbow0_hs0_neg5	0	0	5
Skip-gram with hierarchical softmax	cbow0_hs1_neg0	0	1	0
CBOW with negative sampling, k=5	cbow1_hs0_neg5	1	0	5
CBOW with hierarchical softmax	cbow1_hs1_neg0	1	1	0

Table 4.1: Parameters for 4 types of models explored.

We set the min-count parameter to 1 (instead of default 5) in order to train embeddings for all words in the corpus as we would like to explore the effect of random initialization on words with low frequency.

Other parameters are set to their default values: the size of the context window to 5 (5 words to the left and 5 to the right), embedding size to 100, sampling rate to 10^{-3}, and the initial learning rate alpha to 0.025 for Skip-gram and 0.05 for CBOW.

We fix the number of threads to 1 in order to minimize the effect of multithreading and asynchronous updates of NN matrices.

4.4.2.1 Number of epochs and seed parameters

The number of training epochs is set with the parameter *iter*, and the random seed is set with the parameter *seed*.

To evaluate the performance of word2vec for different numbers of training epochs, we fix the seed value (*seed*=1) and change *iter* in {1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

To evaluate the performance of word2vec for different random seeds, we fix *iter* and vary *seed* in {32, 291, 496, 614, 264, 724, 549, 802, 315, 77}. These numbers are randomly picked from the discrete uniform distribution on interval [1, 1000].

4.4.3 Corpus

As corpus, we choose the Wall Street Journal corpus (WSJ) created by Marcus et al. (1999) from Wall Street Journal articles of 1989. For training, we concatenate parts 00–20, lowercase it and replace every digit with "0". After the pre-processing step, the corpus contains 1,046,148 words and the vocabulary consists of 35,382 tokens.

4.4.4 Evaluation

4.4.4.1 Vocabulary split and evaluation words

We would like to assess the effect of randomization on learned embeddings for words with different frequencies. To do this, we first split the vocabulary according to the word frequencies and then randomly pick 20 words from each frequency interval to perform the evaluation task. We call these words *evaluation words*. The number of words in each frequency interval and the list of picked words are shown in Table 4.2.

frequency intervals	number of words	20 randomly picked words
$[1, 10]$	29043	-0.00, ayers, calisto, chiappa, el-sadr, flower-bordered, kidnappers, mattone, mountaintop, norma, piracy, subskill, configuration, loot, rexall, envisioned, plentiful, endorsing, curbs, templeton
$[10, 100]$	5670	bags, oils, belong, curry, deliberately, responses, constant, yale, tax-exempt, denies, jerry, chosen, iowa, 0000.0, cellular, hearings, extremely, ounce, option, authority
$[100, 1000]$	1013	age, won, announcement, france, plc, thought, thing, merrill, role, growing, 0.0000, black, stores, los, provide, increased, real, public, what, federal
$[1000, ...]$	105	&, all, corp., not, who, up, were, would, company, 000, have, he, its, mr., from, by, it, that, a, the
$[all\_words]$	35382	age-discrimination(1), citizenry(1), less-creditworthy(1), lugging(1), oneyear(1), ton(24), profit-margin(1), sewing(1), unwitting(1), rounds(2), simplicity(2), awesome(3), brush(5), cartoonist(3), brushed(5), unpredictable(5), wsj(7), aluminum(18), dennis(24), contributed(83)

Table 4.2: Number of words in vocabulary for each frequency interval, with 20 randomly picked words used for evaluation (evaluation words). In parentheses: word frequency in the corpus.

4.4.4.2 Evaluation metrics

We apply 3 evaluation metrics:

- comparing 2 trained models:
  1. the ratio of the common words among the top 10 nearest neighbors of evaluation words – topNN;
  2. the correlation of distances between the embedding vectors of evaluation words and vocabulary words – rankNN.
- assessing quality of learned embeddings for a single model:
  3. the Spearman's correlation coefficients for word pairs from similarity data sets.

In the first case, for every evaluation word, we consider the top 10 nearest neighbors according to the embeddings learned by 2 models. We report then the average ratio of common words among these top 10 words for all evaluation words.

In the second case, for every evaluation word, we compute the Spearman's correlation coefficient of similarities for embeddings of this evaluation word and embeddings of vocabulary words. Then we report the average of the computed coefficients for all evaluation words.

In the third case, we employ the word similarity judgment task described in detail in Chapter 2, Section 2.4.2 with similarity data sets described in Section 2.4.3 of Chapter 2.

	RG	MC	MEN	WS	RW	SL
# pairs	65	30	3000	353	2034	999
# covered pairs	46	21	2212	321	405	907

Table 4.3: Number of covered pairs in the six similarity data sets used for evaluation.

Since the training corpus we use here differs from the one in Chapter 2, we observe different coverages of pairs from similarity data sets (see Table 4.3). The performance is reported with the Spearman's correlation coefficient.

As similarity measure, we use cosine similarity between embedding vectors.

4.4.4.3 Evaluation tasks

To evaluate the performance of word2vec with respect to the random seed and the epoch number hyper-parameters, we fix one of them and change the other in a given range as described below.

Numbers of training epochs
First, we would like to determine how many training epochs are needed for word2vec in order to find a solution. For this, we train models for different numbers of epochs up to 100, and then estimate performance of the trained models on a word similarity judgment task. When the performance reaches its maximum, we can say that word2vec found the solution.

Random seeds
After the number of epochs has been determined, we fix it and evaluate how different the solutions are that word2vec finds for different values of the random seed.

4.5 Experimental results and discussion

4.5.1 Different number of training epochs

First, we would like to determine the number of epochs that is needed for word2vec to find a solution: we train 11 models with the same seed=1, one thread, and the number of epochs in {1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}.

The results of the word similarity judgment task is reported in Table 4.4. The highest correlation values are concentrated around 10–30 epochs, that means that word2vec needs ∼20 epochs to learn meaningful embeddings on the WSJ corpus. Such behavior is quite consistent across different model types and different similarity data sets.

		data-sets	number of epochs 1	10	20	30	40	50	60	70	80	90	100
Skip-gram	hs0_neg5	MC	-.17	.25	**.42**	**.42**	.38	.37	.35	.31	.30	.33	.30
		MEN	.07	.10	.17	.21	**.22**	**.22**	**.22**	**.22**	**.22**	**.22**	**.22**
		RG	.12	.18	**.20**	**.20**	.13	.11	.09	.06	.03	.04	.03
		RW	.03	.27	**.29**	.27	.25	.23	.22	.22	.22	.22	.22
		SL	.06	.15	**.17**	.16	.15	.14	.13	.12	.11	.11	.11
		WS	.04	.20	.30	**.31**	**.31**	.30	.29	.29	.29	.29	.29
	hs1_neg0	MC	-.19	.38	**.43**	.32	.29	.27	.28	.32	.27	.26	.21
		MEN	.09	.20	**.23**	**.23**	.22	.22	.21	.21	.20	.20	.20
		RG	.01	**.26**	.24	.22	.22	.22	.25	**.26**	.23	.21	.19
		RW	.11	**.30**	.29	.28	.26	.25	.23	.22	.23	.21	.21
		SL	**.11**	.09	.08	.07	.07	.06	.05	.05	.05	.05	.04
		WS	.11	**.34**	.32	.31	.31	.31	.30	.30	.30	.30	.29
CBOW	hs0_neg5	MC	-.07	**.55**	.44	.36	.33	.24	.18	.19	.21	.24	.27
		MEN	.04	.13	.18	**.19**	**.19**	**.19**	**.19**	**.19**	**.19**	.18	.18
		RG	.18	**.37**	.31	.24	.19	.13	.12	.07	.06	.05	.04
		RW	-.07	.16	**.17**	.16	.15	.14	.14	.13	.13	.13	.13
		SL	-.00	.13	.13	**.14**	**.14**	**.14**	.13	.13	.13	.13	.13
		WS	.09	.30	**.35**	**.35**	.34	.33	.33	.33	.32	.32	.32
	hs1_neg0	MC	.11	**.40**	.34	.30	.32	.29	.32	.35	.31	.31	.34
		MEN	.08	**.15**	.13	.12	.11	.10	.09	.09	.08	.07	.07
		RG	.13	**.26**	**.26**	.23	.20	.16	.17	.15	.12	.11	.12
		RW	.02	**.17**	**.17**	**.17**	.16	.16	.15	.15	.15	.16	**.17**
		SL	.03	.08	.07	.05	.05	.04	.04	.03	.03	.03	.02
		WS	.17	**.25**	.22	.20	.18	.17	.17	.17	.15	.15	.14

Table 4.4: Word similarity judgment task, 1 thread. For same seed=1 and different number of iterations, the Spearman's correlation coefficients are reported. The best values for each data set are highlighted. These results suggest 20 as an optimal number of epochs for word2vec on WSJ training corpus.

4.5.1.1 Conclusion

Experimenting with models with the same seed and increasing epoch numbers we found out that trained models were able to better capture semantics when they were trained for 10–30 epochs. In order to get reliable results in our next experiment on models trained with different random seeds, we fix the number of epochs to 20.

4.5.2 Different random seeds

The results for the first metric, topNN, for 4 model types are reported in Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4 (summarized in Table 4.5). The ratio of the common words among the top 10 nearest neighbors of evaluation words stays almost the same for different seeds with respect to model types and frequency intervals: the standard deviations are in [0.0128, 0.0229]. The average number of common words varies from 5.7 to 9.2 for different model types and intervals, which suggests that the learned models are close but still differ from each other.

The drawback of the topNN metric is that only a limited number of words (10 from ∼35K) are taken into account for every evaluation word. In order to better explore the structure of the learned embedding space, we perform rankNN evaluation. Results for rankNN evaluation (reported in Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8, summarized in Table 4.6) show that the correlation between models with different seeds is very high (above 0.94 on average), while standard deviations are very small (less than 0.0049). These findings suggest that learned embedding spaces for models with different random seeds have very similar structure.

Evaluation on a word similarity judgment task, reported in Table 4.7, shows that found solutions reveal the same quality of learned embeddings (standard deviation does not exceed 0.0651). Nevertheless, for two data sets – MC and RG – the model performances vary much. The reason for such a large variability may be the small size of the MC and RG evaluation data sets: there are only 21 pairs in MC and 46 pairs in RG. For other data sets with more pairs the difference between the obtained results is very small and word2vec results are remarkably stable.

In order to evaluate the difference in the performances on word similarity task for different random seeds, we apply paired t-test ($p < .05$) for every seed pair, treating the correlation coefficients for each data set as a pair in a sample. None of the seeds is found significantly different from others ($.065 < p < 1$).

4.6 Related work

There is no work we are aware of that aims to explore the embeddings spaces trained with word2vec. Tough there are several attempts in the research community to dive and explain how and why word2vec works.

There are two descriptive works to mention: Goldberg and Levy (2014) explain an equation for negative sampling from (Mikolov et al., 2013b); Rong (2014) explains the

	frequency intervals									
	$[1, 10]$		$[10, 100]$		$[100, 1000]$		$[1000, ...]$		$[all]$	
models	avg	std	avg	std	avg	std	avg	std	avg	std
Skip-gram hs0_neg5	.85	.0140	.80	.0173	.80	.0213	.77	.0159	.82	.0148
Skip-gram hs1_neg0	.84	.0192	.75	.0212	.81	.0189	.90	.0155	.78	.0198
CBOW hs0_neg5	.77	.0139	.77	.0146	.83	.0128	.86	.0158	.76	.0229
CBOW hs1_neg0	.68	.0182	.61	.0214	.70	.0192	.87	.0176	.64	.0190

Table 4.5: topNN, 1 thread, 20 epochs, summary of Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4. For all pairs of seeds, the averaged ratio of common words in 10 nearest neighbors is shown for 20 randomly chosen words from each frequency interval. High values suggest that the models with different seeds are similar to each other.

	frequency intervals									
	$[1, 10]$		$[10, 100]$		$[100, 1000]$		$[1000, ...]$		$[all]$	
models	avg	std	avg	std	avg	std	avg	std	avg	std
Skip-gram hs0_neg5	1.00	.0000	.99	.0000	.98	.0000	.98	.0049	1.00	.0000
Skip-gram hs1_neg0	.98	.0000	.96	.0021	.96	.0031	.98	.0000	.98	.0048
CBOW hs0_neg5	.98	.0025	.98	.0000	.97	.0025	.98	.0000	.98	.0000
CBOW hs1_neg0	.95	.0025	.94	.0026	.94	.0029	.97	.0000	.97	.0038

Table 4.6: rankNN, 1 thread, 20 epochs, summary of Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8. Averaged Spearman's correlation of distances between 20 randomly chosen words and all vocabulary word for pairs of models with different seeds is shown. High values suggest that the models with different seeds are similar to each other.

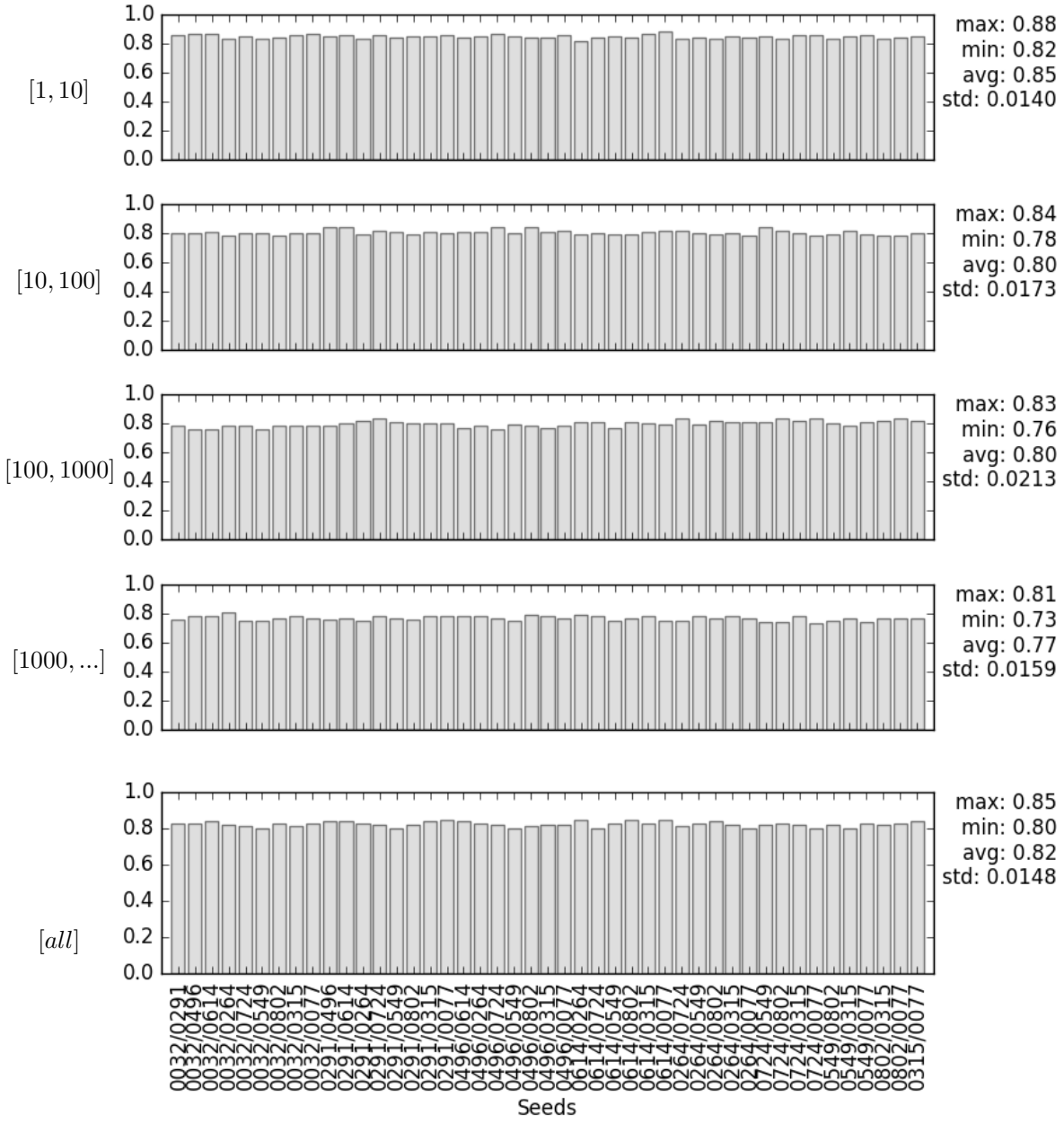


Figure 4.1: topNN, Skip-gram hs0_neg5, 1 thread, 20 epochs. For all pairs of seeds, the ratio of common words in 10 nearest neighbors is shown for 20 randomly chosen words from each frequency interval. High values suggest that the models with different seeds are similar to each other.

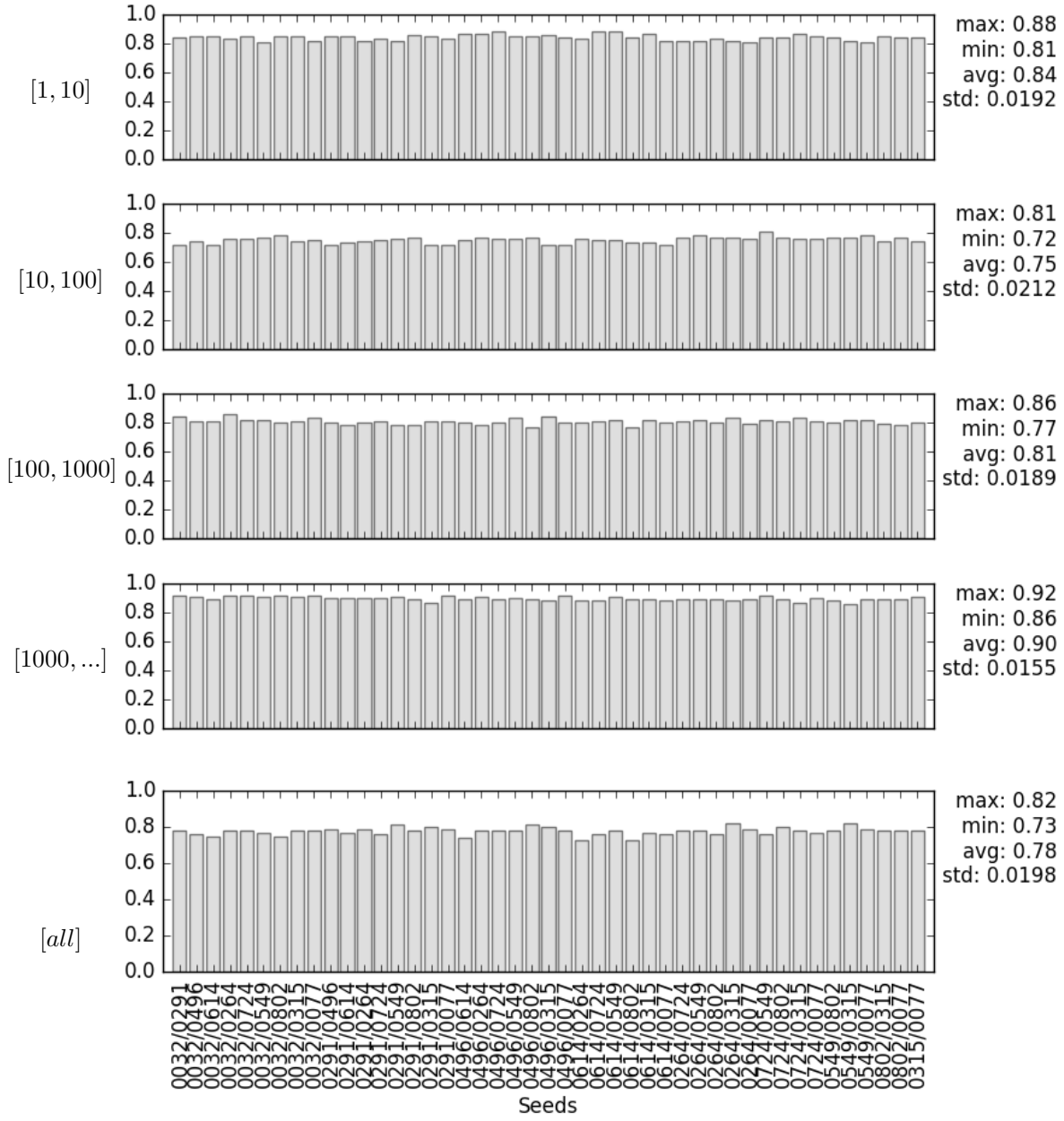


Figure 4.2: topNN, Skip-gram hs1_neg0, 1 thread, 20 epochs. For all pairs of seeds, the ratio of common words in 10 nearest neighbors is shown for 20 randomly chosen words from each frequency interval. High values suggest that the models with different seeds are similar to each other.

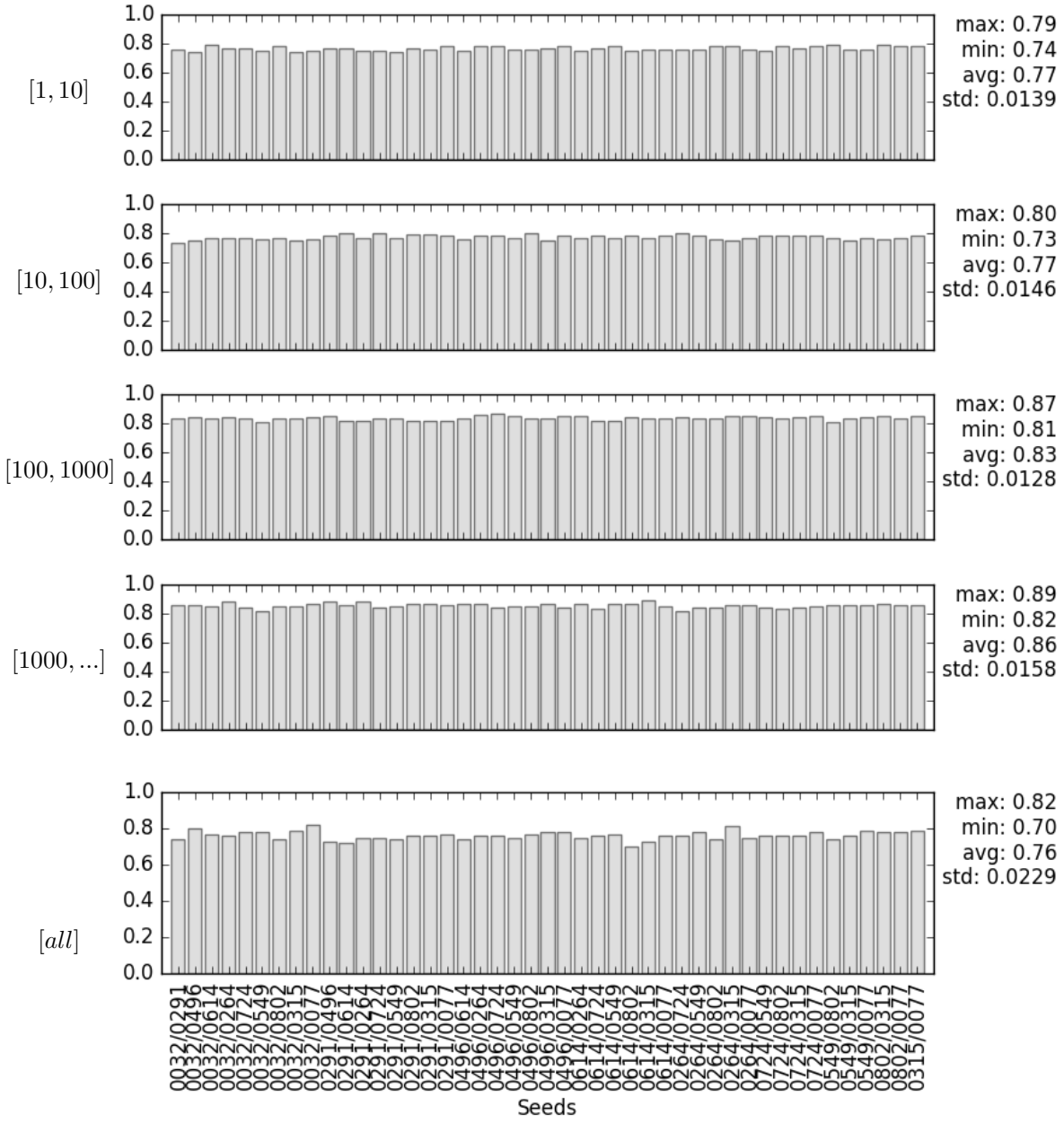


Figure 4.3: topNN, CBOW hs0_neg5, 1 thread, 20 epochs. For all pairs of seeds, the ratio of common words in 10 nearest neighbors is shown for 20 randomly chosen words from each frequency interval. High values suggest that the models with different seeds are similar to each other.

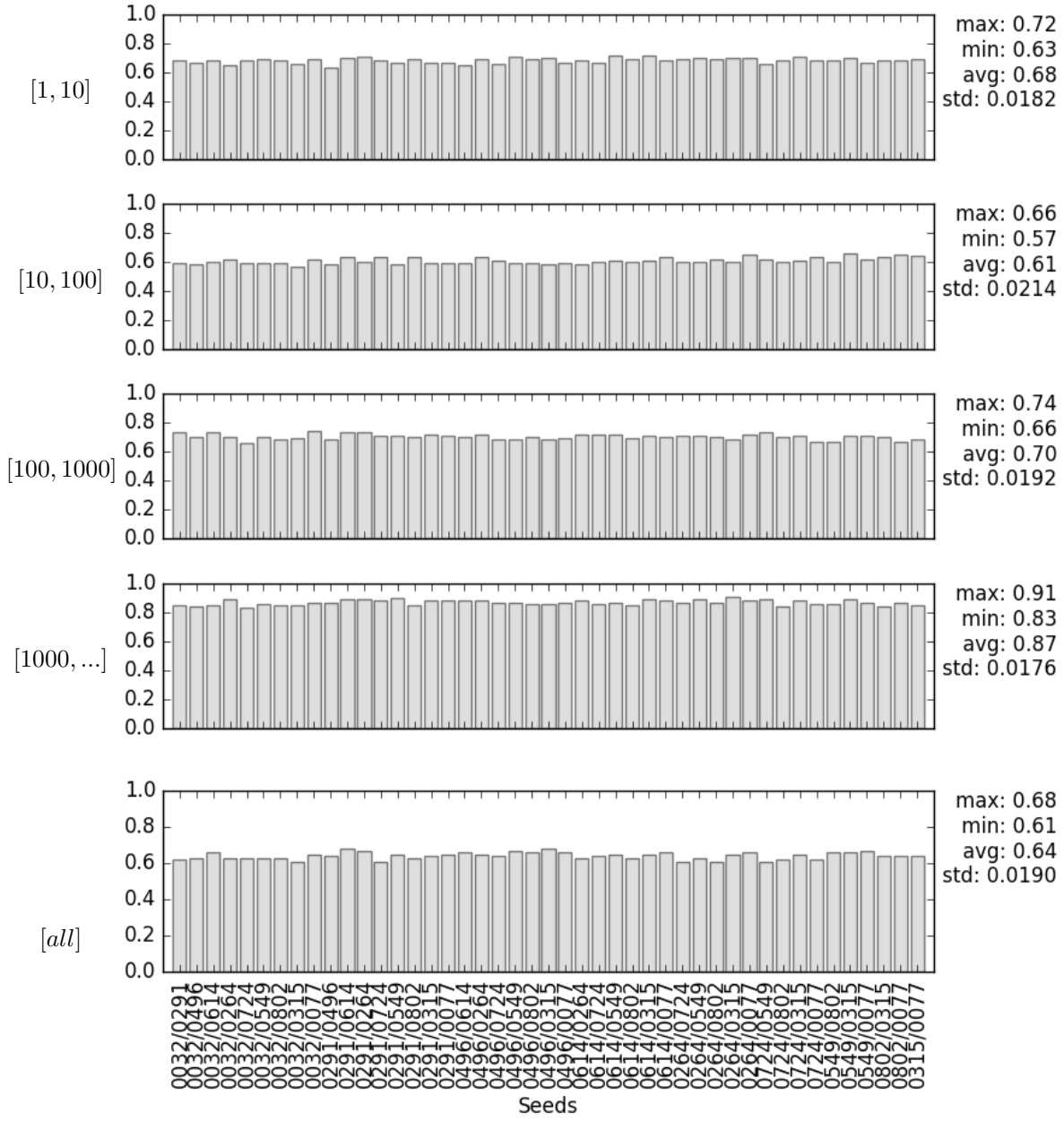


Figure 4.4: topNN, CBOW hs1_neg0, 1 thread, 20 epochs. For all pairs of seeds, the ratio of common words in 10 nearest neighbors is shown for 20 randomly chosen words from each frequency interval. High values suggest that the models with different seeds are similar to each other.

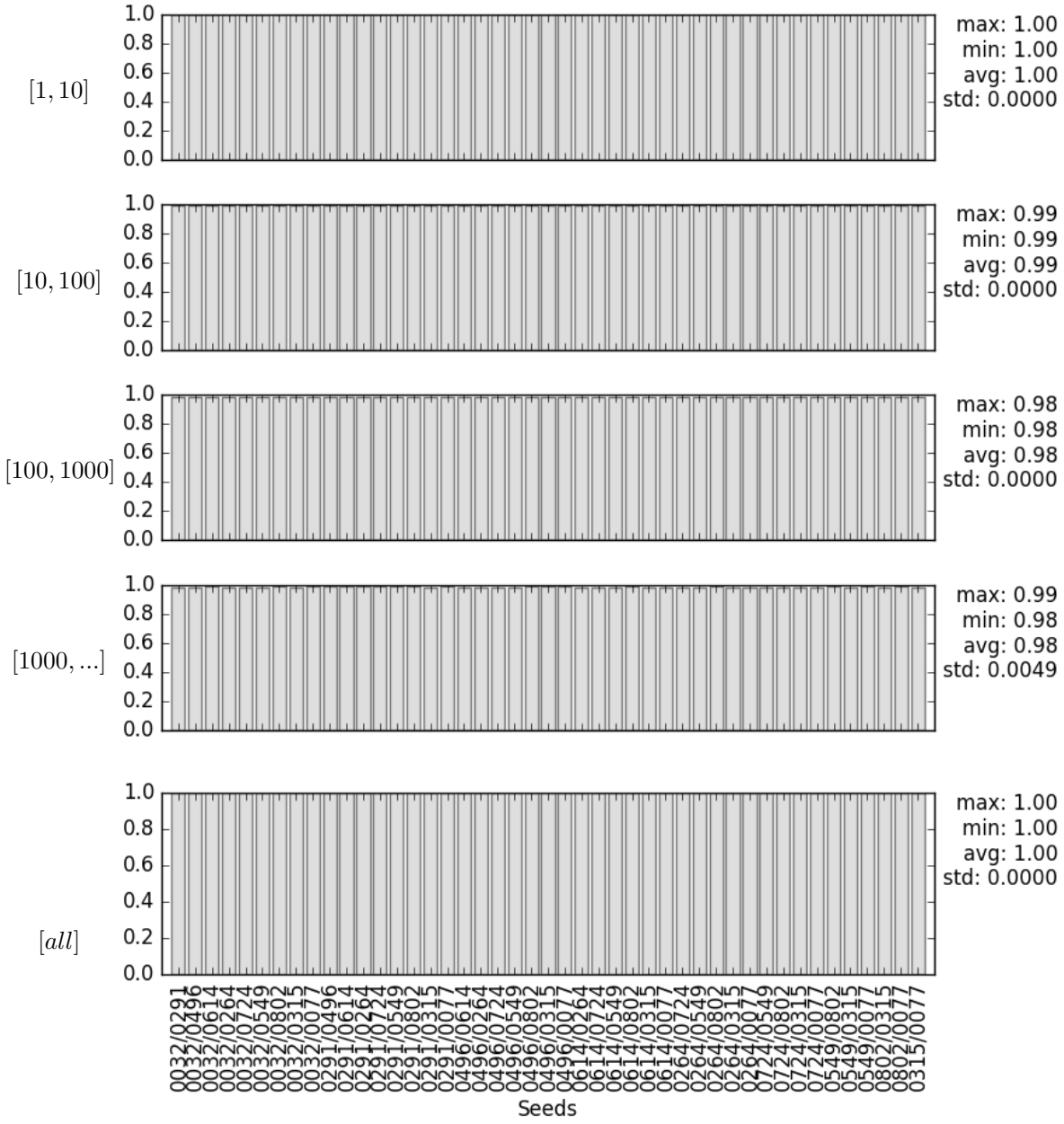


Figure 4.5: rankNN, Skip-gram hs0_neg5, 1 thread, 20 epochs. Averaged Spearman's correlation of distances between 20 randomly chosen words and all vocabulary word for pairs of models with different seeds is shown. High values suggest that the models with different seeds are similar to each other.

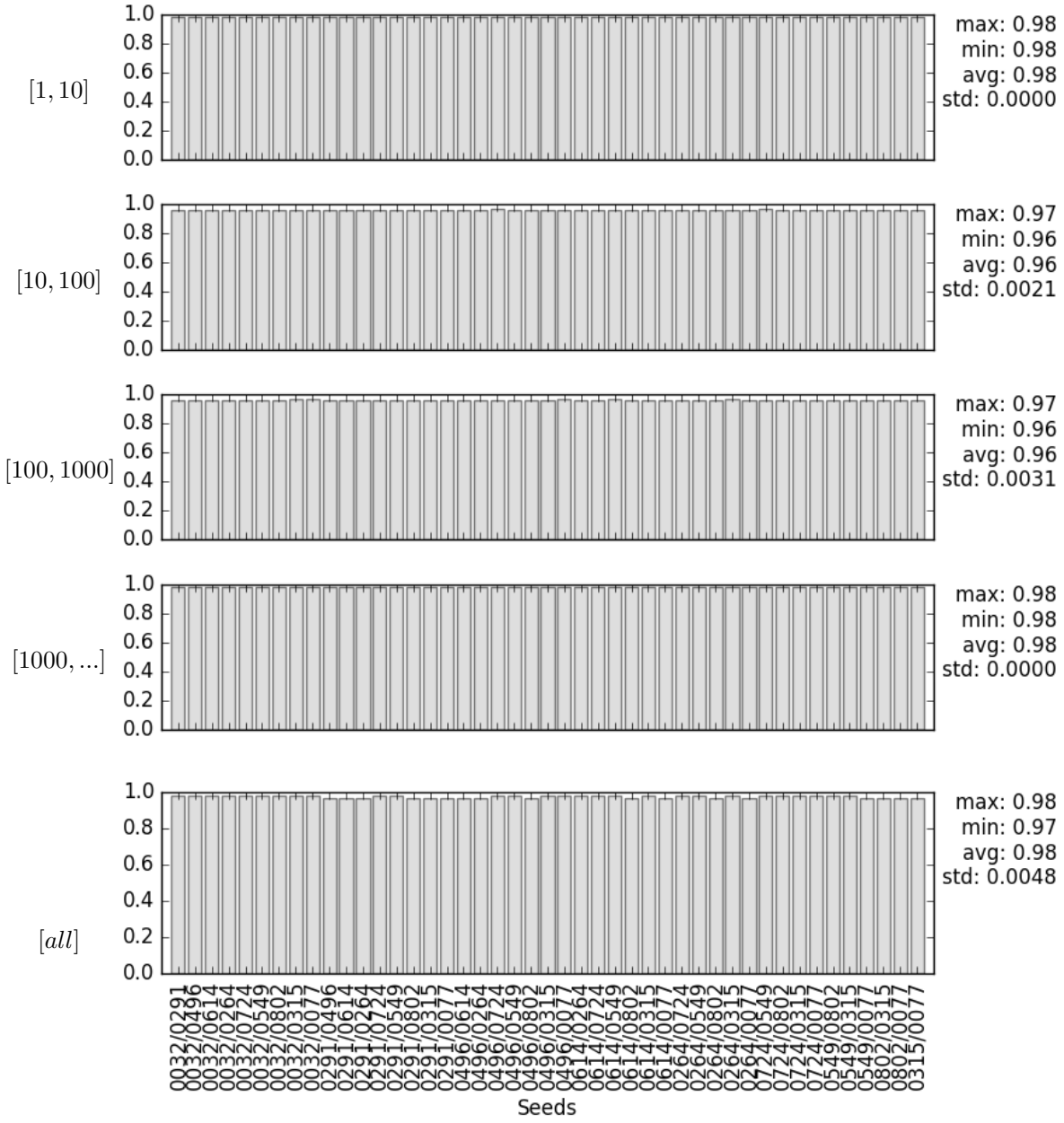


Figure 4.6: rankNN, Skip-gram hs1_neg0, 1 thread, 20 epochs. Averaged Spearman's correlation of distances between 20 randomly chosen words and all vocabulary word for pairs of models with different seeds is shown. High values suggest that the models with different seeds are similar to each other.

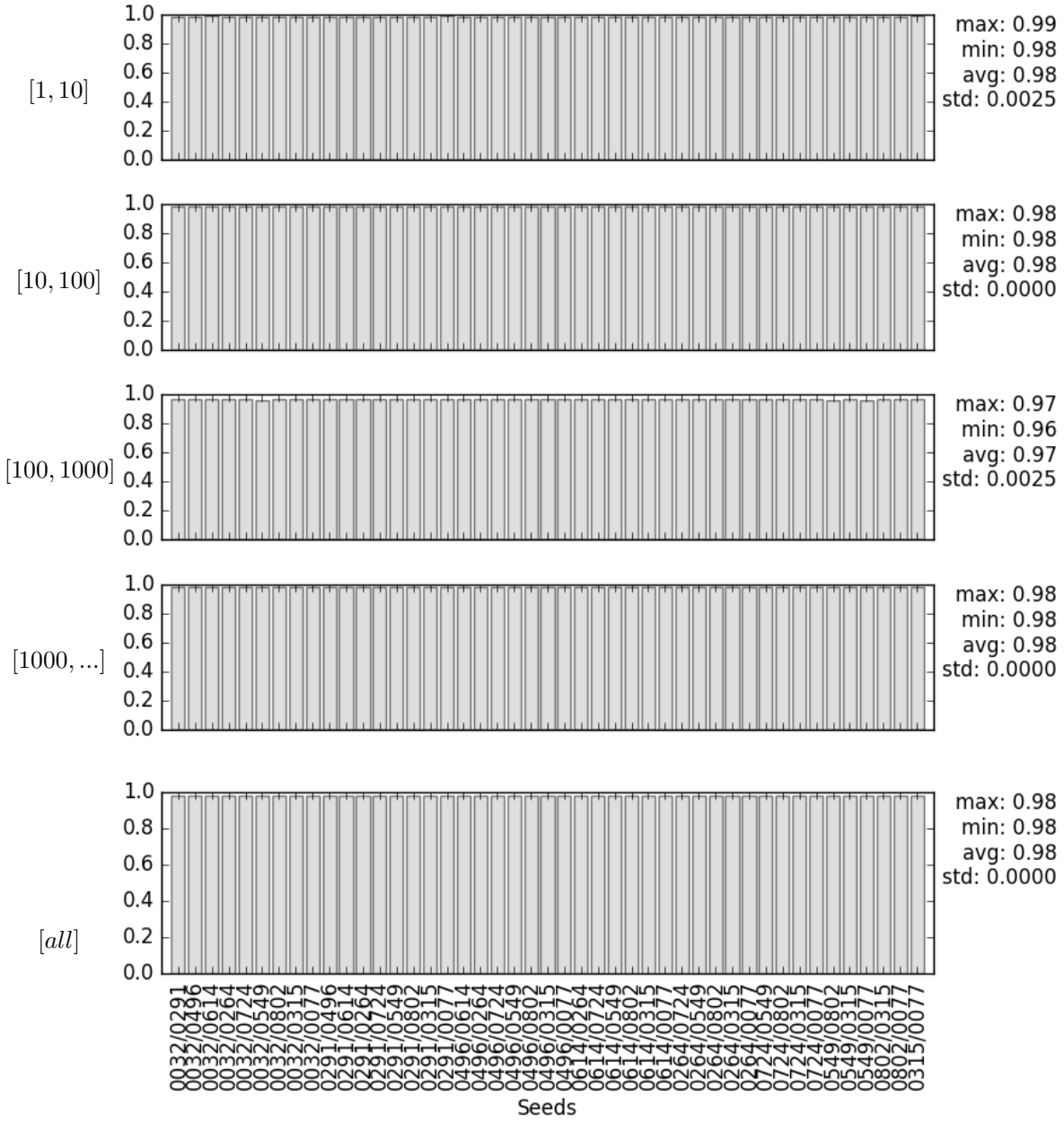


Figure 4.7: rankNN, CBOW hs0_neg5, 1 thread, 20 epochs. Averaged Spearman's correlation of distances between 20 randomly chosen words and all vocabulary word for pairs of models with different seeds is shown. High values suggest that the models with different seeds are similar to each other.

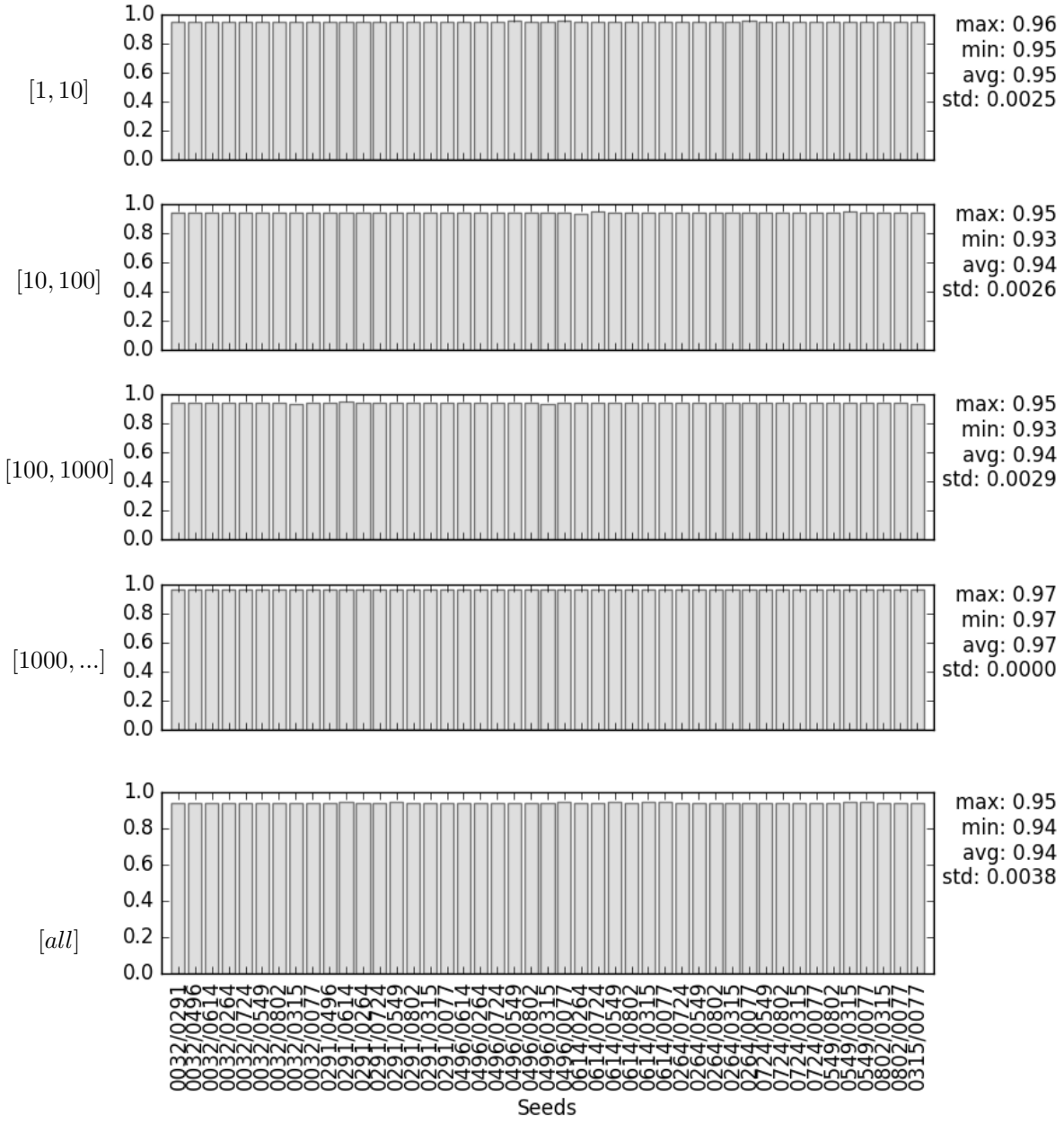


Figure 4.8: rankNN, CBOW hs1_neg0, 1 thread, 20 epochs. Averaged Spearman's correlation of distances between 20 randomly chosen words and all vocabulary word for pairs of models with different seeds is shown. High values suggest that the models with different seeds are similar to each other.

		data sets	seeds 32	291	496	614	264	724	549	802	315	77	min	max	mean	std
Skip-gram	hs0_neg5	MC	.44	.43	.44	.46	.45	.43	.47	.46	.42	.44	.42	.47	.44	.0146
		MEN	.17	.17	.17	.17	.17	.17	.17	.17	.17	.17	.17	.17	.17	.0018
		RG	.19	.21	.19	.22	.20	.20	.22	.21	.21	.20	.19	.22	.21	.0104
		RW	.29	.29	.29	.29	.28	.28	.29	.29	.29	.29	.28	.29	.29	.0048
		SL	.17	.17	.17	.18	.17	.17	.17	.17	.17	.18	.17	.18	.17	.0044
		WS	.30	.31	.30	.33	.31	.31	.30	.30	.30	.30	.30	.33	.31	.0108
	hs1_neg0	MC	.29	.38	.29	.40	.36	.37	.30	.25	.34	.40	.25	.40	.34	.0527
		MEN	.24	.23	.23	.23	.24	.23	.23	.24	.23	.24	.23	.24	.23	.0025
		RG	.20	.19	.16	.25	.16	.24	.21	.15	.20	.20	.15	.25	.19	.0325
		RW	.25	.26	.27	.28	.28	.27	.27	.28	.27	.27	.25	.28	.27	.0086
		SL	.09	.09	.09	.08	.07	.08	.08	.09	.08	.08	.07	.09	.08	.0076
		WS	.30	.29	.31	.31	.31	.32	.30	.30	.31	.31	.29	.32	.31	.0079
CBOW	hs0_neg5	MC	.38	.44	.41	.42	.34	.44	.40	.44	.45	.45	.34	.45	.42	.0360
		MEN	.18	.17	.18	.17	.17	.17	.17	.17	.17	.18	.17	.18	.17	.0029
		RG	.27	.29	.28	.26	.25	.29	.29	.30	.30	.30	.25	.30	.28	.0168
		RW	.16	.18	.17	.17	.17	.19	.17	.17	.19	.18	.16	.19	.17	.0072
		SL	.13	.12	.13	.13	.12	.13	.13	.13	.13	.13	.12	.13	.13	.0025
		WS	.33	.33	.33	.34	.33	.33	.34	.34	.34	.33	.33	.34	.33	.0049
	hs1_neg0	MC	.39	.40	.35	.33	.37	.27	.39	.47	.33	.25	.25	.47	.36	.0651
		MEN	.15	.13	.14	.14	.15	.14	.14	.14	.13	.14	.13	.15	.14	.0063
		RG	.25	.18	.19	.19	.18	.20	.22	.26	.18	.25	.18	.26	.21	.0327
		RW	.14	.15	.14	.14	.16	.13	.12	.14	.15	.14	.12	.16	.14	.0120
		SL	.09	.06	.07	.08	.07	.08	.07	.07	.06	.06	.06	.09	.07	.0093
		WS	.22	.22	.22	.22	.22	.22	.22	.22	.24	.22	.22	.24	.22	.0071

Table 4.7: Word similarity judgment task, 1 thread, 20 epochs. For different seeds, the Spearman's correlation coefficients are reported. Small variation (*std* column) between results for models with different seeds suggests that the found solutions have similar quality of the learned word embeddings.

training process in details together with a tool[4] that visualizes training mechanics.

In (Levy and Goldberg, 2014c) authors analyze Skip-gram models with negative sampling, and conclude that it can be seen as an implicit factorization of the PMI matrix of word/context pairs, shifted by a global constant.

In (Levy et al., 2015) authors delve deeply into the pre-processing hyper-parameters of word2vec such as dynamic context window, subsampling of frequent words and rare words removal. Taking into account hyper-parameters of the GloVe model (Pennington et al., 2014), they make all hyper-parameters explicit for 4 word representation methods: PPMI matrix, singular value decomposition of this matrix, Skip-gram, and GloVe. Afterward they compare the performance of these methods with the same values of hyper-parameters on word similarity and word analogy tasks, and find that no method consistently outperforms the others. The practical recommendations are to use the Skip-gram model with many negative samples.

4.7 Conclusion

In this chapter, we conducted experiments on the word2vec tool in order to analyze the solutions it finds: how much the learned word embeddings differ for different initial random seeds. For this, we modified the word2vec code and introduced a parameter that is responsible for initial random seed in the pseudo-random number generator.

Results of evaluations for models with the same seed and different numbers of epochs suggest that word2vec is able to learn meaningful word embeddings within 10–30 epochs on the WSJ training corpus.

Based on the results of the experiment with different numbers of epochs, we fixed the epoch number to 20 and trained 10 models with different random seeds. The learned models appeared to be different, yet the structure of the learned embedding spaces was found to be very similar.

The achieved results suggest that word2vec seems to produces remarkably stable results for different initial random seeds employed.

4.8 Future work

In future work, we would like to investigate several extensions of our approach:

- to learn transformation matrices between learned embedding spaces for different models and evaluate their differences;
- to perform total randomization of word2vec where all pseudo-random variables are generated based on a given seed;
- to introduce different initialization strategies for the word embedding matrix and evaluate the quality of learned word embeddings.

[4] https://docs.google.com/document/d/1qUH1LvNcp5msoh2FEwTQAUX8KfMq2faGpNv4s4WXhgg/pub

Chapter 5

Conclusion

In this work, we proposed to use sparse distributional vectors to initialize Neural Networks (NNs) aiming to improve quality of word embeddings learned for rare words. Learned word embeddings were tested on word similarity judgment task We also incorporated distributional initialization in training of neural language model, and evaluated performance on language modeling task.

In Chapter 2, we described how to apply our proposal to representation learning with word2vec tool. We proposed four schemes to build distributional representation for rare words: employing binary {0, 1} or positive pointwise mutual information (PPMI) values with mixed or separate integration strategies. BINARY approach pays attention on whether target and context words appeared in the same window in the corpus; PPMI approach also respects degree of association between target and context words. In separate scheme, initialization vectors of frequent and rare words are put in orthogonal spaces. In mixed scheme, they share the space, that also means they share weights of embedding vectors.

We evaluated performance of the embeddings learned with proposed distributional initializations on six word similarity data sets: RG, MC, MEN, WS, RW and SL. All words from these data sets, together with words whose frequency was $\leq$ 10, 20, 50, or 100, received distributional initialization[1]; while other words received one-hot initialization. The measured Spearman's correlation coefficients were significantly higher for models with distributional initialization than for models with traditional one-hot initialization, when distributional initialization was applied for words with frequency up to 20. For medium rare words with frequencies between 50 and 100, no or small improvement was detected. Our final suggestion is to use mixed models with PPMI values for words with frequencies up to 20.

We discussed our findings analyzing scalability of the proposed approach, variance of the results, and proposed distributional schemes. In the future, we would like to investigate

[1]More details on the downsampling procedure of words from similarity data sets can be found in Section 2.4.4

where the achieved improvement came from, and what effect the distributional initialization of rare words produces to frequent words representations.

In Chapter 3, we applied distributional initialization to neural language model vector LBL (vLBL). We trained language model (LM) on WSJ corpus, parts 00–20, interpolated with modified Kneser-Ney 3-gram model learned on the same corpus, and reported the perplexity achieved by the interpolated model on parts 21–22 of WSJ corpus.

We proposed several distributional representations for words employing different combination schemes (mixed, separate, ONLY), different association functions (co-occurrence in a window, with use of positioned PPMI and positioned letter 3-grams), and different normalization schemes (non-diagonal elements were replaced with constant; scaled; rows or columns got normalized with different scaling coefficients).

We also explored the performance of LMs when different sets of words that received distributional initialization. These sets were formed from words with frequency laying either in intervals {[1, 1], [1, 2], [1, 5], [1, 10], [2, 2], [2, 5], [2, 10], [6, 6], [6, 10]}, or equal to a constant value from {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20}.

The experiments showed that neither simple binary representations as in Chapter 2, nor explored distributional initializations achieved stable reliable improvement over a baseline model with one-hot initialization. The provided analysis suggested careful treatment of words with frequency 1, and potential improvement of the performance for mixed initialization with weighting scheme. In the future, the promising extensions are to introduce distributional representations on both input and output layers of NN, to employ more complex NN architectures, and to use a bigger corpus to train language model.

In Chapter 4, we evaluated the word2vec tool by analyzing the structure of the solutions it found for different initial random seeds that were used in a pseudo-random number generator that initialized the matrix of word embeddings. To set a random seed value, we modified the word2vec source code.

First, we trained models with 1 thread for {1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100} epochs in order to decide what number of epochs is needed for word2vec to learn meaningful embeddings on the WSJ corpus. The word similarity judgment task results showed that optimal number of epochs is ∼20 (10–30) for different model types and similarity data sets (we used the same data sets as in Chapter 2: RG, MC, MEN, WS, RW and SL). According to these findings, we fixed the number of epochs to 20 and trained 10 models with random seeds from {32, 291, 496, 614, 264, 724, 549, 802, 315, 77} (ten randomly picked numbers from [1, 1000]). The obtained embeddings spaces were evaluated by means of the number of the common words in the top 10 nearest neighbors, the distances between embedding vectors, and to compare the quality of the learned word embeddings we employed a word similarity judgment task.

The evaluation showed that learned by word2vec models are different, yet had similar structure of the embedding spaces. Thus we concluded that word2vec produced similar results independently of a random seed used during the embedding matrix initialization.

To summarize, our work contributes to the list of attempts to address the problem of building representations of rare words for Natural Language Processing (NLP) tasks. New and borrowed words, together with morphological forms of words in languages with reach morphology, are often rare. Having good representations for them is beneficial for NLP applications: it provides an access to the semantic information that is usually ignored with traditional treatment of such words.

Appendix A

word2vec call

```
make word2vec-dist-ppmi

wesize=100
window=10
datafolder="/ukwac/data"
resultfolder="/ukwac/results"
mincount=1
iter=1

for run in {1..5}; do

  rundir=${resultfolder}/run${run}
  mkdir $rundir

  for rthresh in 10 20 50 100; do

    for modeltype in ppmi bin; do

      for model in onehot mixed separate; do

        traincorpus=${datafolder}/${run}/ukwac_wacky_sent+1unk
           .hide${rthresh}
        readvocab=${traincorpus}_${modeltype}_${model}.txt
        output=${rundir}/ukwacWE_${modeltype}_${model}_hide${
           rthresh}_s${wesize}_w${window}_i${iter}.txt
        savevocab=${rundir}/ukwac_vocab_${modeltype}_${model}
           _hide${rthresh}_i${iter}.txt
        logfile=${rundir}/log_${modeltype}_${model}_hide${
           rthresh}_i${iter}.txt
```

```
        ./word2vec-dist-ppmi \
        -train $traincorpus \
        -output $output \
        -size $wesize \
        -window $window \
        -sample 1e-3 \
        -hs 1 \
        -negative 0 \
        -threads 23 \
        -iter $iter \
        -min-count $mincount \
        -save-vocab $savevocab \
        -read-vocab $readvocab \
        -cbow 0 >> $logfile

      done
    done
  done
done
```

Appendix B

SRILM calls

```
#order 3, train
ngram-count -order 3 \
-write-vocab wsj00-20_vocab_3.txt \
-text /WSJ/corpus/train-wsj-00-20.sent \
-lm lm_kn3_train_wsj_00-20 \
-unk \
-kndiscount \
-gt1min 1 -gt2min 1 -gt3min 1 \

#order 3, predict test
ngram -lm lm_kn3_train_wsj_00-20 \
-order 3 \
-ppl /WSJ/corpus/test-wsj-21-22.sent \
-debug 2 \
-unk > ppl_kn3_test_wsj_21-22.txt

#order 5, train
ngram-count -order 5 \
-write-vocab wsj00-20_vocab_5.txt \
-text /WSJ/corpus/train-wsj-00-20.sent \
-lm lm_kn5_train_wsj_00-20 \
-unk \
-kndiscount \
-gt1min 1 -gt2min 1 -gt3min 1 -gt4min 1 -gt5min 1 \

#order 5, predict test
ngram -lm lm_kn5_train_wsj_00-20 \
-order 5 \
-ppl /WSJ/corpus/test-wsj-21-22.sent \
-debug 2 \
-unk > ppl_kn5_test_wsj_21-22.txt
```

Bibliography

[Adel et al.2013] Heike Adel, Ngoc Thang Vu, and Tanja Schultz. 2013. Combination of recurrent neural networks and factored language models for code-switching language modeling. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 206–211, August.

[Alexandrescu and Kirchhoff2006] Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 1–4, June.

[Auli and Gao2014] Michael Auli and Jianfeng Gao. 2014. Decoder integration and expected BLEU training for recurrent neural network language models. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 136–142, June.

[Baltescu and Blunsom2015] Paul Baltescu and Phil Blunsom. 2015. Pragmatic neural language modelling in machine translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 820–829, May–June.

[Baroni et al.2009] Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. 2009. The WaCky Wide Web: A Collection of Very Large Linguistically Processed Web-Crawled Corpora. *Language Resources and Evaluation*, 43(3):209–226.

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

[Bastien et al.2012] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. 2012. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

[Bengio et al.2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, March.

[Bengio et al.2007] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy Layer-Wise Training of Deep Networks. In *Advances in Neural Information Processing Systems 19*, pages 153–160.

[Bergstra and Bengio2012] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305, February.

[Bergstra et al.2010] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June. Oral Presentation.

[Bian et al.2014] Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Knowledge-powered deep learning for word embedding. In *Machine Learning and Knowledge Discovery in Databases*, volume 8724 of *Lecture Notes in Computer Science*, pages 132–148.

[Bilmes and Kirchhoff2003] Jeff A. Bilmes and Katrin Kirchhoff. 2003. Factored language models and generalized parallel backoff. In *Companion Volume of the Proceedings of HLT-NAACL 2003 - Short Papers*, May–June.

[Blei et al.1993] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 1993. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022.

[Botha and Blunsom2014] Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, June.

[Brown et al.1992] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December.

[Bruni et al.2012] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam Khanh Tran. 2012. Distributional semantics in Technicolor. In *ACL*, pages 136–145.

[Bullinaria and Levy2007] John A. Bullinaria and Jodeph P. Levy. 2007. Extracting semantic representations from word cooccurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510–526, August.

[Celikyilmaz et al.2015] Asli Celikyilmaz, Dilek Hakkani-Tur, Panupong Pasupat, and Ruhi Sarikaya. 2015. Enriching word embeddings using knowledge graph for semantic tagging in conversational dialog systems. In *AAAI Spring Symposium Series*, January.

[Chen and Goodman1999] Stanley F. Chen and Joshua Goodman. 1999. An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech and Language*, 13(4):359–394, October.

[Chen et al.2013] Kuan-Yu Chen, Hung-Shin Lee, Chung-Han Lee, Hsin-Min Wang, and Hsin-Hsi Chen. 2013. A study of language modeling for Chinese spelling check. In *Proceedings of the Seventh SIGHAN Workshop on Chinese Language Processing*, pages 79–83, October.

[Cheng and Kartsaklis2015] Jianpeng Cheng and Dimitri Kartsaklis. 2015. Syntax-aware multi-sense word embeddings for deep compositional models of meaning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1531–1542, September.

[Church and Hanks1990] Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, March.

[Collobert et al.2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

[Cotterell and Schütze2015] Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1287–1292, May–June.

[Cui et al.2015] Qing Cui, Bin Gao, Jiang Bian, Siyu Qiu, Hanjun Dai, and Tie-Yan Liu. 2015. KNET: A general framework for learning word embedding using morphological knowledge. *ACM Transactions on Information Systems (TOIS)*, 34(1):4:1–4:25, August.

[Deerwester et al.1990] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

[Devlin et al.2015a] Jacob Devlin, Hao Cheng, Hao Fang, Saurabh Gupta, Li Deng, Xiaodong He, Geoffrey Zweig, and Margaret Mitchell. 2015a. Language models for image captioning: The quirks and what works. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 100–105, July.

[Devlin et al.2015b] Jacob Devlin, Chris Quirk, and Arul Menezes. 2015b. Pre-computable multi-layer neural network language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 256–260, September.

[Dhillon et al.2012] Paramveer Dhillon, Jordan Rodu, Dean Foster, and Lyle Ungar. 2012. Two Step CCA: A new spectral method for estimating vector models of words. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 1551–1558, July.

[dos Santos and Guimarães2015] Cicero dos Santos and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. In *Proceedings of the Fifth Named Entity Workshop*, pages 25–33, July.

[dos Santos and Zadrozny2014] Cicero dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

[dos Santos et al.2016] Cicero dos Santos, Ming Tan, Bing Xiang, and Bowen Zhou. 2016. Attentive pooling networks. arXiv:1602.03609 [cs.CL].

[Dreyfus1972] Hubert L. Dreyfus. 1972. *What Computers Can't Do: A Critique of Artificial Reason*. Harper & Row, New York, 1 edition.

[Duchi et al.2011] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12:2121–2159, July.

[Dumais et al.1988] Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. 1988. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '88, pages 281–285.

[Finkelstein et al.2001] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *International World Wide Web Conference (WWW10)*, pages 406–414.

[Fried and Duh2015] Daniel Fried and Kevin Duh. 2015. Incorporating both distributional and relational semantics in word representations. *International Conference on Learning Representations (ICLR)*. arXiv:1412.5836 [cs.CL].

[Gao et al.2014] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, and Li Deng. 2014. Modeling interestingness with deep neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2–13, October.

[Goldberg and Levy2014] Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv:1402.3722 [cs.CL].

[Goodman2001] Joshua T. Goodman. 2001. A bit of progress in language modeling: Extended version. Technical Report MSR-TR-2001-72, Microsoft Research, August.

[Gubbins and Vlachos2013] Joseph Gubbins and Andreas Vlachos. 2013. Dependency language models for sentence completion. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1405–1410, October.

[Gutmann and Hyvärinen2010] Michael U. Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. *The Journal of Machine Learning Research W&CP*, 9:297–304.

[Gutmann and Hyvärinen2012] Michael U. Gutmann and Aapo Hyvärinen. 2012. Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. *The Journal of Machine Learning Research*, 13:307–361, February.

[Harris1954] Zellig Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

[Hermansky and Morgan1994] Hynek Hermansky and Nelson Morgan. 1994. RASTA processing of speech. *IEEE Transactions on Speech and Audio Processing*, 2(4):578–589, October.

[Hermansky1990] Hynek Hermansky. 1990. Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4):1738–1752, April.

[Hill et al.2015] Felix Hill, Roi Reichart, and Anna Korhonen. 2015. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, December.

[Hinton et al.2006] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, July.

[Hinton2010] Geoffrey Hinton. 2010. UTML TR 2010-003. A practical guide to training restricted boltzmann machines. Version 1. Technical report, Department of Computer Science, University of Toronto, August.

[Hotelling1936] Harold Hotelling. 1936. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377.

[Huang et al.2012] Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–882, July.

[Huang et al.2013] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 2333–2338.

[Huang et al.2015] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. arXiv:1508.01991 [cs.CL].

[Huffman1952] David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September.

[Iacobacci et al.2015] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2015. Sensembed: Learning sense embeddings for word and relational similarity. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 95–105, July.

[Iyer et al.1994] Rukmini Iyer, Mari Ostendorf, and J. Robin Rohlicek. 1994. Language modeling with sentence-level mixtures. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 82–87.

[Jelinek and Mercer1980] Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, May.

[Katz1987] Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, March.

[Kim et al.2016] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, January.

[Kneser and Ney1995] Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184, May.

[Knuth1997] Donald E. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Kuhn1988] Roland Kuhn. 1988. Speech recognition and the frequency of recently used words: a modified Markov model for natural language. In *Proceedings of the 12th International Conference on Computational Linguistics*, volume 1, pages 348–350, August.

[Larochelle et al.2007] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 473–480.

[Le and Mikolov2014] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of The 31st International Conference on Machine Learning*, volume 32, pages 1188–1196, June.

[Le et al.2010] Hai Son Le, Alexandre Allauzen, Guillaume Wisniewski, and François Yvon. 2010. Training continuous space language models: Some practical issues. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 778–788.

[Le et al.2011] Hai Son. Le, Ilya Oparin, Abdel. Messaoudi, Alexandre Allauzen, Jean-Luc Gauvain, and François Yvon. 2011. Large vocabulary SOUL neural network language models. In *Proceedings of InterSpeech 2011*, pages 1480–1484.

[Lebret and Collobert2015] Rémi Lebret and Ronan Collobert. 2015. Rehabilitation of count-based models for word vector representations. In *Computational Linguistics and Intelligent Text Processing*, volume 9041 of *Lecture Notes in Computer Science*, pages 417–429.

[Lebret et al.2013] Rémi Lebret, Joël Legrand, and Ronan Collobert. 2013. Is deep learning really necessary for word embeddings? In *NIPS Workshop on Deep Learning*.

[Lehmer1949] D. H. Lehmer. 1949. Mathematical methods in large-scale computing units. In *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery*, pages 141–146.

[Levy and Goldberg2014a] Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, June.

[Levy and Goldberg2014b] Omer Levy and Yoav Goldberg. 2014b. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180, June.

[Levy and Goldberg2014c] Omer Levy and Yoav Goldberg. 2014c. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27*, pages 2177–2185. Curran Associates, Inc.

[Levy et al.2015] Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

[Li and Fung2014] Ying Li and Pascale Fung. 2014. Language modeling with functional head constraint for code switching speech recognition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 907–916, October.

[Li et al.2006] Ping Li, Trevor J. Hastie, and Kenneth W. Church. 2006. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 287–296.

[Ling et al.2015a] Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015a. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, May–June.

[Ling et al.2015b] Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015b. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, September.

[Liu et al.2015a] Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. 2015a. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1501–1511, July.

[Liu et al.2015b] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015b. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 912–921, May–June.

[Lowe1999] David G. Lowe. 1999. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157.

[Lund and Burgess1996] Kevin Lund and Curt Burgess. 1996. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208.

[Luong et al.2013] Minh-Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, August.

[Luong et al.2015] Minh-Thang Luong, Michael Kayser, and Christopher D. Manning. 2015. Deep neural language models for machine translation. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 305–309, July.

[Mansur et al.2013] Mairgup Mansur, Wenzhe Pei, and Baobao Chang. 2013. Feature-based neural language model and Chinese word segmentation. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1271–1277, October.

[Marcus et al.1999] Mitchell Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. Treebank-3 LDC99T42. Web Download. Philadelphia: Linguistic Data Consortium.

[Masumura et al.2015] Ryo Masumura, Taichi Asami, Takanobu Oba, Hirokazu Masataki, Sumitaka Sakauchi, and Akinori Ito. 2015. Hierarchical latent words language models for robust modeling to out-of domain tasks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1896–1901, September.

[Mermelstein1976] Paul Mermelstein. 1976. Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Artificial Intelligence*, pages 374–388, 6.

[Mikolov et al.2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *Workshop at International Conference on Learning Representations*.

[Mikolov et al.2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

[Miller and Charles1991] George A. Miller and Walter G. Charles. 1991. Contextual correlates of semantic similarity. *Language & Cognitive Processes*, 6(1):1–28.

[Miller et al.2004] Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name Tagging with Word Clusters and Discriminative Training. In *HLT-NAACL 2004: Main Proceedings*, pages 337–342, May 2 - May 7.

[Miller1995] George A. Miller. 1995. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, November.

[Minsky and Papert1972] Marvin Minsky and Seymour Papert. 1972. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, Cambridge MA.

[Mirowski and Vlachos2015] Piotr Mirowski and Andreas Vlachos. 2015. Dependency recurrent neural language models for sentence completion. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 511–517, July.

[Mitchell1997] Tom M. Mitchell. 1997. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education.

[Mnih and Hinton2007] Andriy Mnih and Geoffrey Hinton. 2007. Three New Graphical Models for Statistical Language Modelling. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 641–648.

[Mnih and Hinton2008] Andriy Mnih and Geoffrey E. Hinton. 2008. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems 21*, pages 1081–1088.

[Mnih and Kavukcuoglu2013] Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26*, pages 2265–2273.

[Mnih and Teh2012] Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 1751–1758, July.

[Morita et al.2015] Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. 2015. Morphological analysis for unsegmented languages using recurrent neural network language model. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2292–2297, September.

[Murray and Chiang2015] Kenton Murray and David Chiang. 2015. Auto-sizing neural networks: With applications to n-gram language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 908–916, September.

[Ney et al.1994] Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependencies in stochastic language modeling. *Computer, Speech & Language*, 8(1):1–38.

[Niwa and Nitta1994a] Yoshiki Niwa and Yoshihiko Nitta. 1994a. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *COLING '94 Proceedings of the 15th conference on Computational linguistics - Volume 1*, volume 1, pages 304–309.

[Niwa and Nitta1994b] Yoshiki Niwa and Yoshihiko Nitta. 1994b. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, pages 304–309.

[Parikh et al.2014] Ankur P. Parikh, Avneesh Saluja, Chris Dyer, and Eric Xing. 2014. Language modeling with power low rank ensembles. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1487–1498, October.

[Pearson1901] Karl F.R.S. Pearson. 1901. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572.

[Pennington et al.2014] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, October.

[Pierce and Carroll1966] John R. Pierce and John B. Carroll. 1966. *Language and Machines: Computers in Translation and Linguistics*. National Academy of Sciences/National Research Council, Washington, DC, USA.

[Qian et al.2015] Qiao Qian, Bo Tian, Minlie Huang, Yang Liu, Xuan Zhu, and Xiaoyan Zhu. 2015. Learning tag embeddings and tag-specific composition functions in recursive neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1365–1374, July.

[Qiu et al.2014] Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 141–150, August.

[Rei2015] Marek Rei. 2015. Online representation learning in recurrent neural language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 238–243, September.

[Rong2014] Xin Rong. 2014. word2vec parameter learning explained. arXiv:1411.2738 [cs.CL].

[Rosenfeld1994] Ronald Rosenfeld. 1994. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. Ph.D. thesis, Carnegie Mellon University, April.

[Rothe and Schütze2015] Sascha Rothe and Hinrich Schütze. 2015. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1793–1803, July.

[Rubenstein and Goodenough1965] Herbert Rubenstein and John B. Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, October.

[S and Kaimal2012] Aji S and Ramachandra Kaimal. 2012. Document summarization using positive pointwise mutual information. *International Journal of Computer Science & Information Technology (IJCSIT)*, 4(2):47–55, April.

[Sahlgren2005] Magnus Sahlgren. 2005. An Introduction to Random Indexing. In *Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering*, August.

[Salimbajevs and Strigins2015] Askars Salimbajevs and Jevgenijs Strigins. 2015. Using sub-word n-gram models for dealing with OOV in large vocabulary speech recognition for latvian. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, pages 281–285, May.

[Schmid2000] Helmut Schmid. 2000. Unsupervised Learning of Period Disambiguation for Tokenisation. Technical report, IMS, University of Stuttgart.

[Schnabel et al.2015] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307, September.

[Schwenk2007] Holger Schwenk. 2007. Continuous space language models. *Computer Speech and Language*, 21(3):492–518, July.

[Sergienya and Schütze2015] Irina Sergienya and Hinrich Schütze. 2015. Learning Better Embeddings for Rare Words Using Distributional Representations. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 280–285, September.

[Shannon1948] Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July.

[Snoek et al.2012] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc.

[Socher et al.2013] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, August.

[Sperr et al.2013] Henning Sperr, Jan Niehues, and Alex Waibel. 2013. Letter n-gram-based input encoding for continuous space language models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 30–39, August.

[Stratos et al.2015] Karl Stratos, Michael Collins, and Daniel Hsu. 2015. Model-based word embeddings from decompositions of count matrices. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1282–1291, July.

[Sun et al.2015] Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. 2015. Learning word representations by jointly modeling syntagmatic and paradigmatic relations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 136–145.

[Suzuki et al.2009] Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 551–560, August.

[Tseng et al.2015] Bo-Hsiang Tseng, Hung-yi Lee, and Lin-Shan Lee. 2015. Personalizing universal recurrent neural network language model with user characteristic features by social network crowdsourcing. In *11th IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 7–12, December.

[Turian et al.2010] Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, July.

[Turney2001] Peter Turney. 2001. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the 12th European Conference on Machine Learning EMCL '01*, pages 491–502, September.

[Vaswani et al.2013] Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, October.

[Řehůřek and Sojka2010] Radim Řehůřek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks*, pages 46–50.

[Vu and Schultz2014] Ngoc Thang Vu and Tanja Schultz. 2014. Exploration of the impact of maximum entropy in recurrent neural network language models for code-switching speech. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 34–41, October.

[Wang and Cho2015] Tian Wang and Kyunghyun Cho. 2015. Larger-context language modelling with recurrent neural network. arXiv:1511.03729 [cs.CL].

[Wang et al.2014] Rui Wang, Hai Zhao, Bao-Liang Lu, Masao Utiyama, and Eiichiro Sumita. 2014. Neural network based bilingual language model growing for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 189–195, October.

[Xu et al.2014] Chang Xu, Yalong Bai, Jiang Bian, Bin Gao, Gang Wang, Xiaoguang Liu, and Tie-Yan Liu. 2014. RC-NET: A general framework for incorporating knowledge into word representations. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 1219–1228.

[Yan et al.2015] Rui Yan, Xiang Li, Mengwen Liu, and Xiaohua Hu. 2015. Tackling sparsity, the achilles heel of social networks: Language model smoothing via social regularization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 623–629, July.

[Yogatama et al.2015] Dani Yogatama, Manaal Faruqui, Chris Dyer, and Noah Smith. 2015. Learning word representations with hierarchical sparse coding. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 87–96.

[Yu and Dredze2014] Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 545–550, June.

[Yu and Dredze2015] Mo Yu and Mark Dredze. 2015. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3:227–242.

[Yu et al.2015] Mo Yu, Matthew R. Gormley, and Mark Dredze. 2015. Combining word embeddings and feature embeddings for fine-grained relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1374–1379, May–June.

[Zhang et al.2015] Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, and Lirong Dai. 2015. The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 495–500, July.

[Zhang et al.2016] Xingxing Zhang, Liang Lu, and Mirella Lapata. 2016. Top-down tree long short-term memory networks. arXiv:1511.00060 [cs.CL].

Glossary

<PAD> padding token 46, 48

<S> beginning of a sentence token 46

<UNK> unknown word token 40, 44–46, 54, 58, 61, 63, 65, 67, 68, 70, 72, 74, 79, 81, 86

<\S> end of a sentence token 46, 48

10K vocabulary from WSJ training set that includes top frequent 10K words xi, 46, 49–51

2vocab vocabulary from WSJ training set that contains words with frequency ≥ 2 46, 49, 51

35K vocabulary from WSJ training set where all digits are replaced with one token and all words are lower cased 46, 81

40K vocabulary from WSJ training set where all digits are replaced with one token 46, 81

45K full vocabulary from WSJ training set xiii, 46, 47, 49, 51, 52

AdaGrad adaptive gradient algorithm 41, 42, 48

AI artificial intelligence 1

CBOW continuous bag-of-words model xi, xii, 12, 13, 19, 94, 96, 97, 105, 106, 109, 110

CN column normalization scheme xiv–xvi, 39, 60, 62–64, 67–69, 74, 75, 78, 79, 84–88

CN-none column normalization scheme without coefficient 39, 64, 67, 69, 85, 86, 88

CN_coeff column normalization coefficient 39, 63, 69, 75, 80, 84, 87

CNN Convolutional Neural Network 91

CPU central processing unit 18, 48

DL Deep Learning 2, 3, 9, 11, 26, 93

DNN Deep Neural Network 10, 26

DR word word that receives distributional representation 34–38, 40, 60

evaluation word one of the randomly picked words from the vocabulary picked to perform evaluation task xvi, 97, 98, 101

FLM Factored Language Model 30

GHz Gigahertz 18, 48

GloVe GloVe model, created by (Pennington et al., 2014) 11, 112

HAL Hyperspace Analogue to Language 9

ID identifier 95

ivLBL inverse vLBL 41

KN3 3-gram modified Kneser-Ney language model xiii–xvi, 49, 51, 52, 59, 61, 63, 64, 66, 69, 71, 74, 75, 78, 80, 83–85, 88

LBL Log-Bilinear language model xiii–xvi, 10, 25, 31, 40, 41, 44, 46, 48, 49, 51, 52, 59–61, 63, 64, 66, 69, 71, 74, 75, 78, 80, 83–85, 87, 88, 90, 91

LDA Latent Dirichlet allocation 9

LM language model 29–34, 36, 38–40, 44, 46–49, 51, 60, 63, 65, 67, 70, 72, 73, 79, 81–83, 85, 87, 89, 90, 114

lr learning rate 66

LSA Latent Semantic Analysis 9

LSI Latent Semantic Indexing 9

LSTM Long-short term memory NN 25, 26, 30, 31, 90, 91

MC data set for a word similarity judgment task, created by (Miller and Charles, 1991) 16, 19, 101, 113, 114

MEN data set for a word similarity judgment task, created by (Bruni et al., 2012) 16, 113, 114

MFCC Mel-frequency cepstral coefficients 8

ML Machine Learning 2, 7, 8, 10

NCE noise-contrastive estimation 42, 43, 48, 94

NLP Natural Language Processing xix, 1–3, 8, 10, 11, 13, 14, 26, 29, 32, 42, 93, 115

NN Neural Network xix, 2–4, 7, 9–12, 22, 30, 31, 39, 42, 52, 55, 59, 60, 62, 64, 65, 69, 72, 85, 90, 94, 95, 97, 113, 114

NNLM Neural Network Language Model 10, 29–32, 40, 41, 90

NOO NUMBER_OF_ONES xi, xiii–xv, 36, 37, 39, 52–55, 58, 59, 61, 65, 67, 70–74, 79, 81, 83, 87

OHR word word that receives one-hot representation 34, 35

PCA Principal Component Analysis 10, 11

PMI pointwise mutual information xi, xiii, 9, 14, 38, 52–54, 112

PPMI positive pointwise mutual information xi, xiii–xvii, 9, 11, 13, 14, 21, 24, 31, 36–39, 52–55, 61, 63, 66, 67, 74, 80, 82–87, 89, 112–114

rankNN evaluation metrics for comparison of 2 models: reports the correlation of distances between the embedding vectors of evaluation words and vocabulary words xii, xvi, 98, 101, 102, 107–110

RASTA-PLP Relative Spectral Transform-Perceptual Linear Prediction 8

RG data set for a word similarity judgment task, created by (Rubenstein and Goodenough, 1965) 16, 19, 101, 113, 114

RN row normalization scheme xiv–xvi, 39, 60, 62–64, 68, 69, 74, 78, 79, 85–88

RN_coeff row normalization coefficient 39, 63, 69, 75, 87

RNN Recurrent Neural Network 25, 26, 30, 90, 91

RW The Stanford Rare Word, data set for a word similarity judgment task, created by (Luong et al., 2013a) 16, 17, 113, 114

S scale normalization scheme 39, 58–60, 63, 64, 68, 69, 74, 79, 84–86, 88

S_coeff scale normalization coefficient xi, xiv, 39, 58, 59, 69, 75, 80, 84, 87, 88

SIFT scale-invariant feature transform 8

SIMILARITY_WEIGHT SIMILARITY_WEIGHT hyper-parameter xiv, 37, 39, 54, 55, 59–61, 65, 67, 70, 72, 74, 79, 81, 84

Skip-gram continuous skip-gram model xi, xii, 12, 13, 18, 19, 25, 94, 96, 97, 103, 104, 107, 108, 112

SL SimLex-999, data set for a word similarity judgment task, created by (Hill et al., 2015) 16, 19, 113, 114

SVD singular value decomposition 9–11, 112

tf–idf term frequency–inverse document frequency 9

topNN evaluation metrics for comparison of 2 models: reports the ratio of the common words among the top 10 nearest neighbors of evaluation words xii, xvi, 98, 101–106

vLBL vector LBL xiii–xvi, 4, 31, 40–42, 48, 49, 51, 52, 54, 58, 59, 61, 63–69, 71, 72, 74, 75, 78–81, 83–85, 88–90, 114

W constant normalization scheme 37, 39, 55, 59–62, 84, 85

word2vec word2vec tool xi, xiii, xvi, xvii, xx, 3–5, 10–13, 18, 19, 21, 22, 24, 25, 27, 93–97, 99–101, 112–114

WS WordSim353, data set for a word similarity judgment task, created by (Finkelstein et al., 2011) 16, 19, 113, 114

WSJ Wall Street Journal corpus xvi, 46, 47, 97, 99, 100, 112, 114