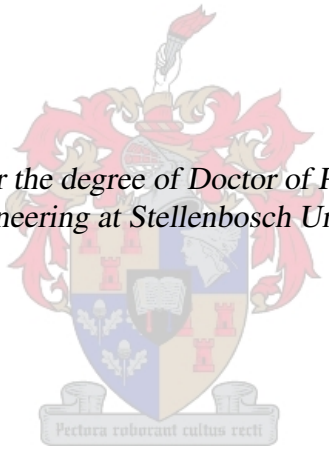
Training Neural Word Embeddings for Transfer Learning and Translation

by

Stephan Gouws



Dissertation accepted for the degree of Doctor of Philosophy in the Faculty of Engineering at Stellenbosch University

Promoters:

Prof. G-J van Rooyen (Stellenbosch University)

Prof. Yoshua Bengio (Université de Montréal)

Prof. Eduard Hovy (Carnegie Mellon University)

March 2016

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualifications.

Date: . . . . . .

Abstract

In contrast to only a decade ago, it is now easy to collect large text corpora from the Web on any topic imaginable. However, in order for information processing systems to perform a useful task, such as answer a user's queries on the content of the text, the raw text first needs to be parsed into the appropriate linguistic structures, like parts of speech, named-entities or semantic entities. Contemporary natural language processing systems rely predominantly on supervised machine learning techniques for performing this task. However, the supervision required to train these models are expensive to come by, since human annotators need to mark up relevant pieces of text with the required labels of interest. Furthermore, machine learning practitioners need to manually engineer a set of task-specific features which represents a wasteful duplication of efforts for each new related task.

An alternative approach is to attempt to automatically *learn* representations from raw text that are useful for predicting a wide variety of linguistic structures. In this dissertation, we hypothesise that neural word embeddings, i.e. representations that use continuous values to represent words in a learned vector space of meaning, are a suitable and efficient approach for learning representations of natural languages that are useful for predicting various aspects related to their meaning. We show experimental results which support this hypothesis, and present several contributions which make inducing word representations faster and applicable for monolingual and various cross-lingual prediction tasks.

The first contribution to this end is **SimTree**, an efficient algorithm for jointly clustering words into semantic classes while training a neural network language model with the hierarchical softmax output layer. The second is an efficient subsampling training technique for speeding up learning while increasing accuracy of word embeddings induced using the hierarchical softmax. The third is **BilBOWA**, a bilingual word embedding model that can efficiently learn to embed words across multiple languages using only a limited sample of parallel raw text, and unlimited amounts of monolingual raw text. The fourth is **Barista**, a bilingual word embedding model that efficiently uses additional semantic information about how words map into equivalence classes, such as parts of speech or word translations, and includes this information during the embedding process. In addition, this dissertation provides an in-depth overview of the different neural language model architectures, and a detailed, tutorial-style overview of the available popular techniques for training these models.

Acknowledgements

A PhD is a large undertaking, and in my case would most likely not have been completed without the support of several people and organisations:

- My promoter, Gert-Jan van Rooyen, for providing a great research environment during the early years and valued support in the later years.
- My second advisor, Ed Hovy, for very useful discussions about my half-baked initial ideas, and for teaching me that PhDs start with "high ambition but little experience", and that with the years the former decreases as the latter increases, eventually culminating in the dissertation.
- My third advisor and research mentor, Yoshua Bengio, from whom I have learned a great deal about machine learning and deep learning in particular. Thank you for taking a chance on me and for the valuable guidance along the way.
- My friends and family who supported me through the long process, and in particular to Dirk Hovy who proofread many manuscripts, shared valuable advice, and shared in the victory beers (or sometimes frustration beers) along the way.
- My former manager, Greg Corrado, for letting me work on BilBOWA during my internship at Google.
- The MIH Medialab and the National Research Foundation (NRF) for providing much appreciated financial funding along the way.

Dedication

Opgedra aan my pa, Johan Gouws, in baie opsigte die eerste Dr. Gouws.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

$\alpha(w^+)$	NCE weight for positive words
$\beta(w^-)$	NCE weight for negative words
δ^θ	Derivative of the output loss $\mathcal{L}$ wrt θ
$\delta^\theta(w)$	Contribution to δ^θ resulting from word w
$\gamma(x)$	Importance weight $P(x)/Q(x)$ for item x
$\sigma(z)$	Sigmoid function, $1/(1+e^{-z})$
θ	Model parameter ("weight")
$\Gamma(x)$	Normalizer for importance weights $\Gamma = \sum_i \gamma(x_i)$
$\Omega_{\mathbf{A}}(\mathbf{R}^e, \mathbf{R}^f)$	Cross-lingual regularizer on English and French embeddings $\mathbf{R}^e$ and $\mathbf{R}^f$ given alignments $\mathbf{A}$
a_{ij}	Word-alignment score of word i in the source with word j in the target language
b_i	Bias weight for label i
$\mathbb{E}_{P(x)}\, f(x)$	Expected value of $f(x)$ under $P(x)$
$f_\theta(h)$	Hidden-layer representation for context words h (see also r_h)
$\tilde{h}_j$	Empirical average context-vector for word j
h	Context-words $h = \{w_1, w_2, \cdots, w_n\}$
$\mathbf{H}$	Matrix where $\mathbf{H}_{[j,:]} = \tilde{h}_j$
k	Number of contrastive words used during training
$L(w)$	Path in hierarchical softmax tree for word w
$\mathcal{L}$	Training loss function (objective function)
$\mathcal{P}$	Distribution of positive (observed) words used during training
$\mathcal{N}$	Distribution of negative (contrastive) words used during training
$P_\theta(w\|h)$	Normalized model distribution
$\tilde{p}_\theta(w)$	Unnormalized model distribution

$P_n(w)$	Noise distribution
q_{w_i}	Output word embedding for word w_i
$\mathbf{Q}$	Output word embeddings
$Q(w)$	Proposal distribution
r_h	Hidden-layer representation $r_h = f_\theta(h)$ for context h
r_{w_i}	Input word embedding for word w_i
$r(x)$	Vector representation for object x
$\mathbf{R}$	Input word embeddings
s_{en}	English parallel sentence
s_{fr}	French parallel sentence
V	Number of words in vocabulary
$\mathcal{V}$	Set of words in vocabulary
w_t	Target word
$\tilde{x}$	Corrupted input
$\mathcal{X}$	Input space
$\mathcal{Y}$	Output (label) space
$Z_\theta(h)$	Context-dependent normalization constant

Abbreviations

BIS	Biased Importance Sampling
BAE	Bilingual Autoencoder
BiCVM	Bilingual Compositional Vector Model
BOW	bag-of-words
CBOW	Continuous Bag-Of-Words model
CLDC	cross-language document classification task
HS	Hierarchical Softmax
LBL	Log-bilinear Language Model
LSA	Latent Semantic Analysis
MSE	Mean squared-error
NCE	Noise-contrastive Estimation
NER	named-entity recognition
NLL	negative log-likelihood
NLM	Neural Language Model

NLP	Natural Lanuage Processing
NPLM	Neural Probabilistic Language Model
POS	part-of-speech
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SS	Supersense
SVD	Singular Value Decomposition

Chapter 1

Introduction

1.1 Motivation

Language is a complex phenomenon with a non-trivial mapping from words to meaning. Humans learn how to extract meaning from the words in a sentence from an early age in a seemingly effortless manner. However, getting machines to parse text into useful linguistic structures representing its meaning (see for instance Fig. 1.1) has been a significant computational challenge driving research in Artificial Intelligence in general, and Natural Language Processing (NLP) in particular, over the last several decades.

Early efforts in NLP involved the design of human-curated rules which governed how linguistic units combine to collectively contribute to the *meaning* of the text [2]. However, this approach soon ran into significant problems, as the number of rules required to accurately capture the rich ambiguity and expressiveness of natural languages quickly exploded, making rule-based approaches only useful in very limited and reduced domains (such as the SHRDLU blocks world [3]).

In its place, a statistical approach to NLP emerged in the early 1990s which replaced the human-designed rules with a complex ensemble of machine-derived rules. Each rule is defined in terms of parameters that are estimated from training data in order to collectively enable the model to successfully predict the linguistic structures of interest [4, 5]. Over the last 20 years, this approach has proven very successful. However, there are two serious

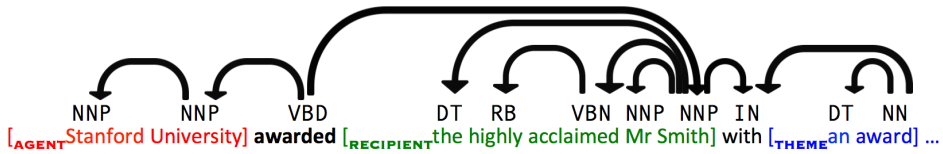


Figure 1.1: An example of the types of linguistic structures that we are interested in predicting in natural language processing.

drawbacks associated with it:

1. Firstly, most successful statistical methods are trained in a supervised training regime where we need to have annotated data in the form of text marked up with the specific labels that we would like the models to predict. Obtaining this level of supervision is a costly process, and requires human annotators to read and mark up sections of text.

2. Secondly, these techniques generally require the human practitioner to manually engineer a set of discriminative features for representing the natural language data, in order to serve as inputs for training the models. It is a hard and time-consuming problem to come up with features that work well for more than one task, but it is even harder for cross-lingual prediction tasks, i.e. predicting attributes of linguistic meaning across different languages.

In this dissertation we study a problem that is central to natural language processing:

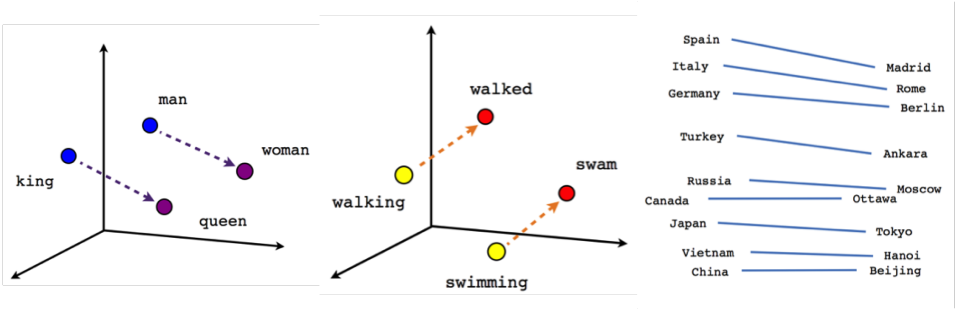
> *How can we **efficiently** learn to represent natural languages in order to accurately predict **different** aspects of their meaning through computational means?*

This formulation captures the two important guiding aspects of the research objectives that underlie this study:

Firstly, how can one *learn* representations of natural languages that are not task-dependent, but can accurately capture underlying important linguistic regularities that are useful for predicting a variety of linguistic attributes, such as semantic topics, parts-of-speech, etc.?

Secondly, how can one do this *efficiently*, both in terms of computational resources, but also in terms of supervised resources? The first consideration (computational efficiency) is important, since it is much more desirable to have a model that trains in hours as opposed to a model that requires days or weeks to train. However, the second consideration (efficient reuse of supervised resources) is equally important, as it is extremely desirable that the learned representations are able to capture useful regularities across different languages, since this means that we do not need to manually provide supervision for every new language. Instead, we can efficiently reuse annotations that we may have for one language and apply it across a variety of different languages. While the contributions in this dissertation by no means represent an exhaustive solution to these problems, we make some progress in this direction by presenting different models that efficiently learn cross-lingual correspondences directly from raw text. The learned representations are shown to preserve enough linguistic information to enable us to reach state-of-the-art performances on a variety of important cross-lingual prediction tasks, with minimal supervision required.

The overarching theme of this work is to employ neural network models for the task of learning representations of natural languages. Over the last few years, neural network



(a) Male-female.

(b) Verb tense.

(c) Country-capital.

Figure 1.2: Visualisations of the different linguistic relationships that neural language models have been found to learn and encode in its embedding space in an unsupervised fashion from large collections of text [15].

models, under the guise of **deep learning**, have seen a resurgence in the machine learning community by significantly improving upon previous state-of-the-art approaches on a number of important tasks, such as object recognition [6], speech recognition [7], and also NLP [8–11]. In the case of NLP, the prevailing approach is to use neural network language models (NLMs) in order to induce unsupervised *neural word embeddings*, which are then subsequently applied as features within a predictive model across a range of different tasks. Neural word embeddings are *distributed* representations that use continuous values to represent words in a learned vector space of meaning. A distributed representation of a symbol is a vector where the meaning of the symbol is distributed across a combination of different elements in the vector.

NLMs were originally proposed as an alternative for n-gram language models [12]. However, two seminal papers by Collobert and Weston [8, 13] showed that these unsupervised embedding features are directly useful for predicting various different linguistic attributes, such as syntactic **parts-of-speech (POS)** like NOUN or VERB, **named-entities** like PERSON or LOCATION, and **semantic role labels** such as AGENT or THEME. In fact, it was discovered that embedding spaces encode a surprising amount of useful "real-world" information such as `male-female` relationships, verb tense, and even `country-capital` relationships in terms of specific dimensions (directions) in the embedding space (see Fig 1.2). Several follow-up work proposed various simplifications to the original model architectures and more efficient ways for training the models [14] to reduce the time it takes to train these models if one is only interested in obtaining the learned embedding features.

However, at the outset of this research, word embedding models could only be trained in one language. Bilingual models soon emerged [16, 17]. However, they were extremely unwieldy and slow to train and usually required the practitioner to perform an entire word-alignment step prior to training the embedding models.

In this dissertation, we develop efficient neural network models for learning distributed word representations across one or more languages by training directly on raw text. We exploit the modelling power of neural network language models for learning common distributed representations for words that are similar in one language, and extend these models to enable efficient cross-lingual feature learning. We then demonstrate the applicability of the induced representations by successfully utilising them as features for training statistical models for a variety of cross-lingual prediction tasks, across several different languages.

The benefits of this approach are that it requires very little manual feature engineering, and the limited available training data in one language (e.g. English POS data) is automatically transferred to new languages (e.g. French POS labels) via the common, learned feature space.

1.2 Research Objectives

This dissertation studies the problem of automatically *learning* to represent natural languages in order to successfully predict *different* aspects of their meaning by computational means. Specifically, we study the problem of automatically learning distributed representations of words that capture useful linguistic information in one or more languages. By inducing these representations jointly across different languages, we are furthermore able to reuse and transfer expensive labeled training data from one language to other languages where we may not have much, or any, labeled information.

The **main hypothesis** underlying this research can be stated as:

> *Neural word embeddings induced from raw text can jointly represent two or more languages in order to accurately predict different aspects of their meaning through computational means.*

Through the course of this dissertation, we present experimental evidence that strongly supports this hypothesis.

As an example, consider a source dataset annotated with parts-of-speech (e.g. VERB, NOUN, or PREPOSITION) in a domain consisting of English newswire text. Given a target dataset in Swedish, our goal is to develop a target model that can predict Swedish parts-of-speech by utilising only the annotations we have for the English data (which are expensive to obtain), and any amount of unlabelled English and Swedish text (which is cheap and easy to obtain).

This problem is very important for contemporary natural language processing systems for two reasons:

Firstly, training supervised statistical models (currently the most accurate method for constructing language processing systems) requires obtaining annotated training data. This process is expensive in terms of human annotator time. Automated methods for adapting

models from one domain to another can therefore result in significant savings in terms of financial cost and annotator time, as well as model training time.

Secondly, contemporary methods require the practitioner to manually engineer a set of useful discriminative features for predicting the linguistic structures of interest. This often represents a wasteful duplication of efforts between related tasks. The techniques discussed in this dissertation learn features that are useful across a wide variety of tasks and languages, thereby saving on manual feature-engineering effort and annotator time.

1.3 Contributions

At a high level, the contributions in this work all relate to the general objective of learning to represent natural languages in order to successfully predict different aspects of their meaning by computational means. Specifically, we contribute to the understanding of this problem by developing efficient methods for learning *distributed representations* that capture the similarities between words *across different languages* purely from large quantities of raw text, and using minimal supervision. We demonstrate the versatility of the learned representations across a range of tasks, namely for predicting the topics of documents in two languages, predicting labels annotated at the word level that describe the syntactic (POS) and semantic (supersense) meaning of words and phrases, and as a successful method for translating words in a source language to a target language by using only the learned representations.

The main contributions of this work can be broken up into three categories:

1. **Theoretical**: We present a thorough introduction to and overview of the field of distributional methods which encode word meaning as learned distributed vectors, and with a particular emphasis on neural language models. We present all distributional methods within a common unsupervised learning framework where the model learns to compress high-dimensional contextual co-occurrence information into a lower-dimensional representation for each word in order to retain the most important information. We introduce a novel geometrical framework that identifies the connections between the wide array of word embedding models, by geometrically viewing the learning process as working to balance forces acting on the learned representations within the induced vector space of linguistic meaning. We show how popular contemporary shallow `word2vec`-type embedding models fit into this framework, thereby highlighting the connections between existing distributional methods and the several recent successful neural word embedding models.

2. **Practical**: We present the first comprehensive tutorial of the field of neural word embedding models (§ 2.3 and Chapter 3), with a particular focus on the practical implementation details of training these models. We furthermore provide efficient

implementations of all the novel bilingual models considered in this dissertation, and make our code available as open-source to the research community at large[1].

Being a computational and largely empirical field, a significant component of new research is spent on understanding and reimplementing successful models. In releasing our code, we hope to afford ongoing researchers with a warm-start in using and extending upon the methods presented in this work.

3. **Empirical**: We experimentally demonstrate that the representations learned using the proposed methods are able to significantly improve results on a variety of tasks and languages (and generally outperform the state of the art), demonstrating the wide range of their applicability. Namely:

   a) *language modelling*, where we show that the SimTree algorithm can be used to reduce the perplexity of a binary hierarchical softmax output layer, by efficiently learning a likelihood-optimal tree structure during training (§ 4.2.2). We show that the algorithm leads to a reduction of 37 points in perplexity over a random tree, and 20 over a Huffman tree in a language modelling task.

   b) *analogical reasoning*, where we show that a simple subsampling technique can be used to improve the quality of word vectors learned using the hierarchical softmax output layer on a standard analogical reasoning task, while speeding up training of the vectors at the same time (§ 4.3.4).

   c) *cross-lingual document topic classification*, where the **BilBOWA** model is shown to outperform several strong competing baseline methods and other distributed methods for predicting the topics of German documents using only the learned representations and the annotated topics of a small training set of English documents (§ 5.5.2). We furthermore show that the model achieves this increase in prediction accuracy despite a computational speedup of several orders of magnitude compared to most other bilingual distributed models.

   d) *cross-lingual part-of-speech tagging*, where the **Barista** model is shown to learn useful representations for learning to predict the parts-of-speech of Spanish, German, Danish, Swedish, Dutch, Italian and Portuguese, by training only on the annotated parts-of-speech of a sample of English text and the learned word representations (§ 6.5.2.2). We show that our method outperforms off-the-shelf bilingual embeddings for German and Spanish and two very strong baselines for the Dutch and German language-pairs. The model is furthermore able to learn these rich representations by training only on raw Wikipedia text and using a limited dictionary obtained from Google Translate in roughly half an hour on a standard desktop machine.

[1] https://github.com/gouwsmeister

e) *word-level translation* for English-Spanish, where we show that the BilBOWA model is able to learn fine-grained word-level translations by training only on a limited sample of sentence-aligned raw text, and unlimited amounts of monolingual raw text (§ 5.5.3). We show that such a raw-text-only approach – i.e. without resorting to word-level alignments or dictionaries – is able to improve significantly on the previous state-of-the-art for this task which makes use of dictionaries for learning the translations.

f) *supersense tagging*, where embeddings induced using the Barista model is shown to increase English-Danish cross-lingual supersense tagging accuracy by 10% absolute (§ 6.5.3).

1.4 Publications

The work in this dissertation primarily relates to the following peer-reviewed articles:

1. *Simple task-specific bilingual word embeddings*, **Stephan Gouws**, Anders Søgaard, *North American Chapter of the Association for Computational Linguistics*, NAACL 2015.

2. *BilBOWA: Fast Bilingual Distributed Representations without Word Alignments* **Stephan Gouws**, Yoshua Bengio and Greg Corrado, *Deep Learning Workshop*, NIPS 2014.

3. *Learning Structural Correspondences Across Different Linguistic Domains Using Synchronous Neural Language Models*, **Stephan Gouws**, G-J van Rooyen and Yoshua Bengio, *xLite Workshop on Cross-Lingual Technologies*, NIPS 2012.

4. *Deep Unsupervised Feature Learning For Natural Language Processing* **Stephan Gouws**, *Student Research Workshop*, NAACL-HLT 2012, Montréal, Canada.

5. *BilBOWA: Fast Bilingual Distributed Representations without Word Alignments*[2] **Stephan Gouws**, Yoshua Bengio and Greg Corrado, ICML 2015.

Furthermore, the following article is currently under review:

6. *SimTree: Learning to Predict N labels in $O(N \log N)$ Time using Neural Networks*, **Stephan Gouws**, Submitted to *IEEE Transactions on Neural Networks and Learning Systems*.

[2] This represents a significant rework of the workshop version of the same article.

Parts of Chapter 2 and the entire Chapter 3 is based on a tutorial that I have presented at Carnegie Mellon University in Dec 2012, Microsoft Research (Israel) in Nov 2013, and at University of Copenhagen in Nov 2014[3].

Finally, while not directly related, the following articles played a significant role during the initial development of the ideas presented in this dissertation:

7. *Unsupervised Mining of Lexical Variants from Noisy Text*, **Stephan Gouws**, Dirk Hovy and Donald Metzler, *Unsupervised Methods in NLP Workshop*, EMNLP 2011, Edinburgh, Scotland.

8. *Contextual Bearing on Linguistic Variation in Social Media*, **Stephan Gouws**, Donald Metzler, CongXing Cai, Eduard Hovy, *Language in Social Media Workshop*, ACL 2011, Portland, Oregon, USA.

1.5 Overview

Chapter 2 provides an overview of the background information required for understanding the work in this dissertation, and acts as an introduction to the notation used throughout. We introduce the basics of probability theory leading up to expectations and Monte-Carlo integration, concepts that are used throughout. We introduce the basics of machine learning and neural networks, with a particular focus on the backpropagation algorithm, as it is central in this dissertation. We also provide a high-level overview of statistical natural language processing.

Chapter 3 provides a comprehensive overview of the field of neural language models, with a particular focus on its application to learning word embedding features. We introduce a novel geometrical framework for providing a geometrical understanding about how shallow `word2vec`-type neural word embedding models are trained.

In Chapter 4 we study tree-structured output layers (the *hierarchical softmax*) in monolingual models, and focus on how they can be used to significantly speed up training, but we also show how the structure of the tree impacts the quality of the model. Through the novel theoretical insight gained, we propose an efficient $O(N \log N)$ algorithm, for N the number of words in the vocabulary, called SimTree. It is an alternating optimization algorithm that jointly learns the tree structure with the model parameters. We present qualitative and quantitative experimental evidence that the algorithm learns useful semantic clusterings and show that the learned trees significantly improve the quality of the model in a language modeling task. We also introduce a simple subsampling technique for improving the quality of the embedding vectors learned using the hierarchical softmax, and show that the tech-

[3] See www.stephangouws.com/talks.

nique both speeds up training of the embeddings and improves their quality on a standard analogical reasoning task.

Chapter 5 introduces BilBOWA, our first bilingual word embedding model. BilBOWA learns bilingual distributed representations of words using a limited sample of parallel sentence-aligned text (e.g. Europarl) and unlimited amounts of monolingual text (e.g. Wikipedia). We evaluate the embeddings on a cross-lingual document classification task where we obtain state-of-the-art results. For this task, BilBOWA furthermore achieves these results in minutes on a standard desktop machine, compared to days reported by other methods. We also evaluate the cross-lingual embeddings on a word-translation task and obtain a large improvement over the previous state-of-the-art. The surprising fact is that this is achieved by training purely on raw text without resorting to word-alignments or dictionaries, as required by current alternative methods.

Chapter 6 introduces Barista, our second bilingual word embedding model. Barista is a simple, computationally-efficient wrapper technique for efficiently including external semantic information in the form of how words map into equivalence classes (e.g. parts of speech) during the embedding process. We evaluate the model on cross-lingual part-of-speech tagging, where the goal is to train on English data and then evaluate the model on seven other languages using primarily the learned embeddings as features. We show that our method is competitive with very strong baseline methods, and is able to outperform off-the-shelf bilingual embeddings and several strong baselines on the Dutch and German languages. We also apply the model to Danish supersense tagging where the embeddings yield a 10% absolute improvement in prediction accuracy. For all these experiments, the implementation trains in about half an hour on a standard desktop machine.

Finally, Chapter 7 provides a summary of the work presented in this dissertation, and offers final thoughts on the insights gained during the course of this research.

Chapter 2

Background

This chapter covers relevant background material on probability theory, machine learning, neural networks and statistical natural language processing. The goal is primarily to provide an introduction to the notation that we will be using during the rest of this dissertation, and secondarily to act as a short refresher on these topics. However, this chapter may safely be skipped or referred back to only when needed.

2.1 Probabilistic Models

Probability theory provides a formal calculus for reasoning under uncertainty. Several of the models discussed in this dissertation are trained within a probabilistic framework where we will appeal to concepts like the "expectation of some function under some distribution", or make use of Monte-Carlo averages to approximate expectations.

2.1.1 Probability basics

Random variables X represent outcomes or states that the world, or more specifically our model of the world, may be in. Random variables are defined in a *sample space* which is the space of all the possible outcomes or states that X may take on. This space my be discrete or continuous or even mixed, but in this dissertation we will only consider discrete spaces. The probability for an event X to take on the value x is a non-negative number defined in terms of a function called the *probability density function* (pdf) in the case of continuous random variables and *probability mass funtion* (pmf) in the case of discrete X. In particular, we will write lower-case $p(x)$ to denote pdfs and upper-case $P(x)$ for pmfs.

All probability distributions sum (integrate) to one. Intuitively, this means that a fixed quantity of total probability mass is distributed among a predetermined number of possible outcomes: making one outcome more likely necessarily means that the remaining outcomes must become less likely. Mathematically, this means that $\sum_x P(x) = 1$ and $\int_x p(x)dx = 1$.

Depending on the type of X (binary, categorical, or continuous) there are several popular distributions (pdfs or pmfs) which are parametrised families of distributions telling us how likely it is to get a certain outcome x under that particular distribution. For binary variables, the Bernoulli distribution is particularly common. For categorical variables (i.e. a generalisation from binary to k-ary outcomes) the multinomial or multinoulli distribution is a popular choice and for general continuous variables the Gaussian or normal distribution has a long and rich history in engineering and scientific modelling.

The **expectation** of a function $f(x)$ is the average value that $f(x)$ can be expected to take on, given that the values x are distributed according to $P(x)$, and in this dissertation we will write this as

$$\mathop{\mathbb{E}}_{x\sim P(x)}[f(x)] = \sum_x P(x)f(x), \tag{2.1.1}$$

although sometimes we will drop the "x is drawn from $P(x)$" and simply write it as $\mathbb{E}_{P(x)}[f(x)]$ or even $\mathbb{E}_P[f(x)]$ when the context makes it unambiguous.

Notice that if $f(x) = x$, then this is simply the mean $\mu = \sum_x P(x)x$, which, for a uniform distribution over N events, $P(x) = 1/N$, simply becomes the well-known arithmetic average $1/N \sum_x x$.

Furthermore if $f(x) = (x-\mu)^2$, 2.1.1 becomes the **variance** $\sigma^2 = \sum_x P(x)(x-\mu)^2$, or in words: variance is the average squared-deviation of x from the mean value. **Moments** are expectations of higher orders. The mean is the first moment of $P(x)$, variance the second.

This extends very simply to vector random variables $x \in \mathbb{R}^d$, where the mean $\mu = \mathbb{E}_P[x]$ and the **covariance** of x is defined as

$$\Sigma_x = \mathop{\mathbb{E}}_{P}\left[(x-\mu)(x-\mu)^\top\right]. \tag{2.1.2}$$

Notice the **outer product**, since the covariance describes how each component of x varies with or correlates with every other component, and hence it is an $(d \times d)$ matrix for x a d-dimensional vector. We can extend this notion of covariance from one to two variables as

$$\Sigma_{xy} = \mathop{\mathbb{E}}_{P}\left[(x-\mu_x)(y-\mu_y)^\top\right]. \tag{2.1.3}$$

In words, this measures how one component of x correlates with each component of y. This is somewhat related to the idea of **joint probability** $P(x,y)$, which represents the probability that $X = x$ *and* $Y = y$. If we think of all the combinations of discrete outcomes that X and Y can take on, it is useful to think of this as a two-dimensional *probability table* P where each entry P_{ij} represents the joint probability $P(X = x_i, Y = y_j)$. Since, by definition, all entries in this table must sum to one $\sum_{i,j} P(x_i, y_j) = 1$, it means that if we keep x_i fixed and sum over all y_j we get the **marginal probability** of event x_i happening regardless of the outcome of y,

$$P(x_i) = \sum_j P(x_i, y_j). \tag{2.1.4}$$

This process is called marginalisation because when the joint probability is written as a table, the marginals are found by summing the rows or columns and can then be written along the "margins" of the table. It is also referred to as the **sum-rule** of probability theory.

Whereas the joint probability describes two events being true simultaneously, the **conditional probability** $P(y|x)$ describes the probability of y being true after we have already observed x being true. We define this as

$$P(y|x) \triangleq P(x,y)/P(x) \tag{2.1.5}$$

which can be understood as restricting the total (joint) sample space to only those entries containing x. This definition ensures that $\sum_y P(y|x) = 1$. Rearranging 2.1.5 yields the **product-rule** of probability theory

$$P(x,y) = P(y|x)P(x). \tag{2.1.6}$$

which relates conditionals and marginals to the joint distribution.

2.1.2 Entropy and Cross-entropy

Given a distribution $P(x)$, the **entropy** is a concept from Information Theory [18] that measures the ambiguity or the uncertainty that is present in P, and is defined as:

$$H(P(x)) = -\sum_x P(x)\log(P(x)) = -\mathop{\mathbb{E}}_P\left[\log(P(x))\right]. \tag{2.1.7}$$

Recall that $\log(P(x))$ is a monotonically-increasing function on the $[0,1]$ interval (where $P(x)$ is defined) from $-\infty$ for $P(x) = 0$ to 0 when $P(x) = 1$. The binary logarithm of the probability of an outcome quantifies in bits the amount of information that is present in observing that event, and is inversely related to its probability of occurrence: More frequent events are less surprising, or in information theory parlance, contain fewer bits of information. Therefore, 2.1.7 measures the average bits of information that are present in a distribution. Note that it is maximised under a uniform distribution, since, if all events are equally likely, we would be maximally surprised by observing any event.

Cross-entropy extends this idea to two distributions and in a sense measures the *difference in information* between two distributions. Cross-entropy is intimately related to the Kullback-Leibler (KL) divergence which measures the "distance" between two distributions P and Q as

$$KL[P||Q] = \sum_x P(x)[\log P(x)/Q(x)] = \mathop{\mathbb{E}}_P\left[\log P(x) - \log Q(x)\right]. \tag{2.1.8}$$

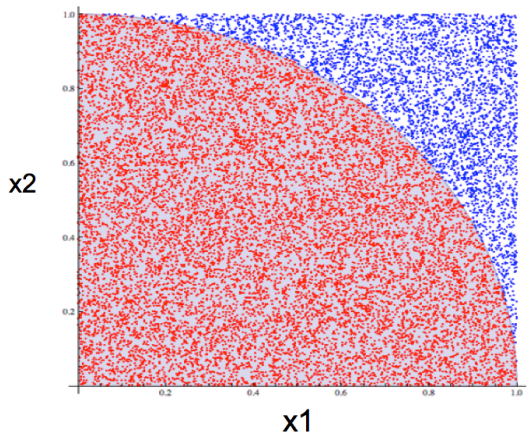


Figure 2.1: Approximating π by Monte Carlo integration: π can be approximated as the ratio of the surface area of a circle inscribed in a unit square to the area of the square. We can approximate this value by drawing points uniformly at random over the area of the square and evaluating the ratio of points that fall within the circle (picture taken from the Wikipedia page).

2.1.3 Monte Carlo methods

Monte Carlo techniques are a family of techniques that approximate numerical quantities by repeatedly drawing random samples [19]. For instance, if we are interested in the value for $\mathbb{E}_{P(x)}[f(x)]$, we can *approximate* this value by drawing N samples from $P(x)$, and averaging the values obtained as $1/N \sum_{x \sim P(x)} f(x)$.

In this dissertation, we will be interested in approximating expectations by sampling, so we will illustrate the general idea by using a standard example: approximating the value of π by sampling. Consider a circle inscribed in a unit square (see Fig. 2.1). We know that the ratio of the area of the circle to that of the square in any one quadrant is equal to $\pi/4$. If we define $P(x_1, x_2)$ as the uniform distribution over $0 \leq x_1 \leq 1$ and likewise for x_2, then the value of π can be written as an expectation which we can approximate using Monte Carlo sampling by drawing N samples and computing the empirical average ratio:

$$\pi = 4 * \mathop{\mathbb{E}}_{P(x_1,x_2)} [\mathbb{1}[(x_1, x_2) \text{ in circle}]] \tag{2.1.9}$$

$$\approx 4 * 1/N \sum_{x_1,x_2 \sim P(x_1,x_2)} \mathbb{1}[(x_1, x_2) \text{ in circle}]. \tag{2.1.10}$$

where $\mathbb{1}[x]$ is the indicator function which evaluates to 1 if x is true, and 0 otherwise.

2.2 Machine Learning

This dissertation provides a comprehensive coverage of neural network language models which are a particular family of machine learning models for learning from data. In this section we provide a high-level overview of the aspects of machine learning most related to the work presented in this dissertation. This is by no means a comprehensive overview. For that the reader is directed to [20] or [21]. In the rest of this chapter, we will delve into more technical detail when we look at neural networks as a *particular* family of machine learning models.

2.2.1 Learning from data

Machine learning studies the problem of designing models that can learn from data. One widely accepted definition defines *learning* as the process whereby a *model* improves its performance with respect to some *task*, as measured in terms of some *performance measure*, by learning from examples from some *training set* [22]. Let us consider each of these components in a little more detail.

A model is a function or algorithm which aims to extract important regularities from training examples which allows it to perform well on the task. Although models can be non-parametric, meaning some function of the training set which can not be encoded using a parameter vector θ (for example nearest-neighbour methods), in this dissertation we will focus only on parametric models, where the model is defined in terms of a vector of parameters θ. In this setting, each model belongs to a family of models, and each different θ 'indexes' or selects one particular member from that family. The goal of the learning procedure is therefore to select the θ which leads to the best performance (which we will describe below).

There are a wide variety of tasks to which a model can be applied, and these are roughly divided into supervised and unsupervised tasks. In **supervised learning** the training data takes the form of a training *pair* of inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$ provided by a teacher or oracle. The goal is for the learner to learn to predict the outputs from the inputs. We refer to the task as classification when the outputs are discrete identifiers and regression when the outputs are real-valued numbers. Examples of classification include predicting the label of an object in a picture or predicting the part of speech of a word in a sentence. An example of regression is to predict the temperature from inputs such as past temperature or air pressure.

In **unsupervised learning** the training data is not associated with any labels, i.e. the learner only observes the inputs $x \in \mathcal{X}$. The goal is to automatically discover interesting structure in the training data by clustering data into different groups or learning the probability density function on the space of the training data. For example, if we have such a density model trained on images of faces, it may allow us to query the model about how

'likely' some new input image is to be a face or not.

In this dissertation we will only be concerned with the classification task, however both in the unsupervised setting, when we train neural language models to predict target words from context words, as well as in the supervised setting when we train classifiers to predict the topics, part-of-speech tags, etc., of words and documents.

The goal of training is to find the model that leads to the best performance. We therefore need an appropriate performance measure for how well the model is doing on the training data (also called an objective function, loss function or error function). However, in general, what we really care about is how well the model will *generalise to* unobserved, future test data. For this one typically makes use of regularisation techniques, which we will discuss in § 2.2.2.

The objective function plays a crucial part in training the model. In Chapter 3 we will go into much detail analysing different objective functions for training neural network language models. However in general the three main types of objectives relate to whether we are training a probabilistic model or non-probabilistic model. For probabilistic models we will generally minimise the cross-entropy of the model predictions (see § 2.3.2.3). This corresponds to applying the *maximum likelihood learning* principle of maximising the predictions that the model makes on the training data. For non-probabilistic models we will employ either the squared-error if the goal is to predict real-valued numbers (see § 2.3.2.1), or alternatively the hinge-loss when the goal is to score the data samples at least some margin higher than some contrastive set of noise samples (see Sections 2.3.2.2).

2.2.2 Regularisation

A crucial learning component is to prevent the model from overfitting or underfitting the training data. A model *overfits* when its performance on the training data is still improving while its performance on test data is decreasing. This happens when the model memorises every spurious nuance of the training data (which might not exist in the test data) instead of extracting the underlying regularities which should also be present in the future test data in which we are interested. Likewise, a model is *underfitting* when its performance is still decreasing on both the training and test set.

One very popular method for training models is called **maximum likelihood estimation**. The maximum likelihood inductive principle is to select the parameters θ which maximise the probability that the model $P_\theta(x)$ assigns to the data $\{x_1, x_2, \cdots\} \in X$, i.e.

$$\hat{\theta} = \arg\max_{\theta} \Pi_{x_i \in X} P_\theta(x_i) \tag{2.2.1}$$

The problem with this estimator is that it trusts the training data implicitly as being predictive of the future test data. When training data is limited, the model might overfit.

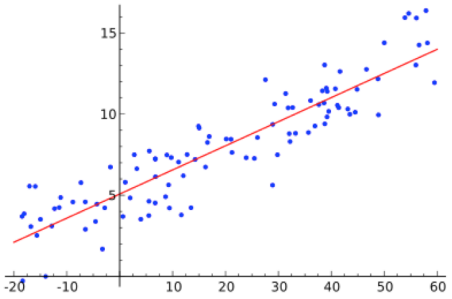


Figure 2.2: A linear regression model $y = wx+b$ that models the linear relationship between input variable x (horizontal axis) and output variable y (vertical axis).

One way of preventing overfitting is to use a regulariser. Essentially, a regulariser penalizes more expressive models to encourage the optimizer to select only simpler models that can not memorise all the peculiarities of the training data. For example, in the above example, this may lead to a model that assigns a lower total probability to the training data, but can be expected to generalise better to unseen test data.

Another common regularisation method employed with neural networks is *early stopping*, where we monitor the model's performance on a held-out *validation set* of data during training, and stop training once the performance on the validation set starts decreasing. We will look at this in more detail in § 2.3.3.2. In this dissertation, we employ both regularisation and early stopping for training the models.

2.3 Neural Networks

Neural networks are powerful machine learning models for learning relationships between input and output variables. A neural network model is a composition of simpler models.

In order to understand this better, let us consider a **linear regression** model

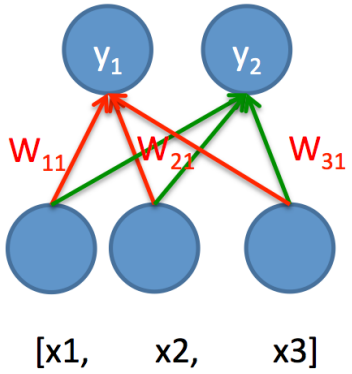
$$y = wx + b, \tag{2.3.1}$$

that models the linear relationship between an input scalar variable x and an output scalar variable $y \in \mathbb{R}$ in terms of the model parameters $\theta = [w, b]$ (see Fig. 2.2).

If instead our goal is to predict a binary (Bernoulli) variable $y \in \{0, 1\}$, we can pass the output y of the linear regression model through a *squashing function* $\sigma(z)$ and interpret the output as a probability

$$P(y = 1|w, x, b) = \sigma(wx + b) \tag{2.3.2}$$

where $\sigma(z) = 1/(1 + \exp(-z))$ maps all values on the real line to the $[0, 1]$ interval. This is known as **logistic regression**. Logistic regression is a linear, probabilistic, binary classifier.



(a) The network.

$$W^T x = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= [[W_{11}x_1 + W_{21}x_2 + W_{31}x_3], \\ [W_{12}x_1 + W_{22}x_2 + W_{32}x_3]]$$

(b) A visual illustration of the matrix-vector multiplication $\mathbf{W}^\top x$.

Figure 2.3: A linear neural network that maps vector inputs x to vector outputs y via a linear (affine) transformation $y = \mathbf{W}^\top x + b$, i.e. $y_j = \sum_i w_{ij} x_i + b_j$.

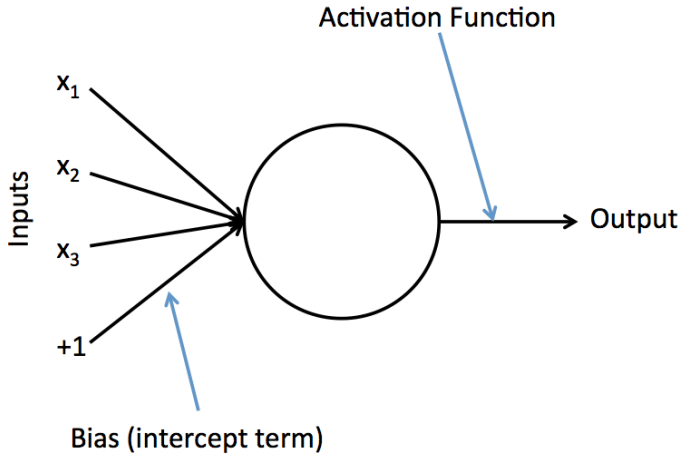


Figure 2.4: An illustration of a single neuron.

This is also known as a **neuron**, where we refer to the weighted input as the *activation*, the sigmoid as the *activation function* and the right-hand side as the *output* of the neuron. See Fig. 2.4 for an illustration.

We can easily extend this model to map m-dimensional *vector* inputs x to n-dimensional vector outputs y by replacing the scalar w with a $(m \times n)$-dimensional matrix $\mathbf{W}$,

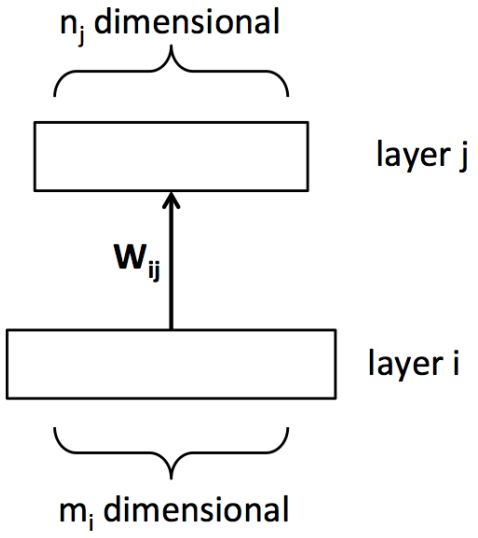


Figure 2.5: Two layers of a neural network.

$$\underbrace{a}_{(n\times 1)\text{ 'activation'}} = \underbrace{\mathbf{W}^\top}_{(n\times m)} \underbrace{x}_{(m\times 1)} + \underbrace{b}_{(n\times 1)} \tag{2.3.3}$$

$$\underbrace{y}_{(n\times 1)\text{ 'output'}} = \underbrace{g(a)}_{\text{element-wise}} \tag{2.3.4}$$

where we have introduced the notion of activations being passed through a linear activation function $g(a) = a$. This represents a **linear neural network** (see Fig. 2.3). Note that any cascade of linear functions $f(g(x))$ can always be compiled down into a single linear function $h(x)$. Therefore, if we wish to increase the capacity of the model, the final step is to replace the linear activation function with a squashing function (non-linearity) to transform the linear model into a non-linear model so that we can compose simpler models into more complex models. We achieve this by applying a nonlinear function $g(a)$ (e.g. the sigmoid from before) element-wise to the outputs (activations) of the linear neural network. This is a single hidden-layer neural network. By repeating this process, we introduce more layers and hence more *depth*, which increases the capacity of the model.

A neural network is therefore a composition of nonlinear functions $f(g(\cdots(x)\cdots))$. An "L-layer neural network" is a composition of $L-1$ functions applied to an input vector x. Each layer j recursively transforms an m_i-dimensional input vector x_i into an n_j-dimensional output vector y_j, by taking the matrix-vector product with an (n_j, m_i)-dimensional matrix $\mathbf{W}_{ij}$ to yield the activation a_j, adding a bias b_j and applying a non-linear activation function $y_j = g_j(a_j)$ element-wise to the individual activation values. See the illustration in Fig. 2.5.

2.3.1 Inference: Forward propagation

A neural network computes its predicted outputs from the given inputs in a process called **forward propagation**, or fprop for short. For a feedforward network defined as $y = g(f(\cdots(x)\cdots))$, fprop simply corresponds to visiting each layer of the network in turn and computing the activations on that layer from the activations of its input variables, starting with the input layer. In Python code this can be implemented as shown in Listing 1.

Listing 1 Python code implementing the fprop algorithm for a feedforward neural network.

```
def fprop(biases, weights, input):
  y = input      # initialise
  acts = []      # activations
  ys = [y]       # layer-wise outputs
  for W,b in zip(weights, biases):
    a = dot(W.T, y) + b
    y = sigmoid(a)
    ys.append(y)
    acts.append(a)
  return (acts, ys)  #ys[-1] contains the network output prediction
```

2.3.2 Loss functions

Fprop on input x gets us a prediction $\hat{y}$. In supervised training, we are presented with input-output training *pairs* (x_i, y_i^*). So what we really want is to predict the true target y^*. The **loss function** $\mathcal{L}$ measures the discrepancy between the network prediction and the true target.

2.3.2.1 Squared-error

As the name implies, squared-error measures the error as the squared difference between the network prediction $\hat{y}$ and the true label $y^* \in \mathbb{R}$, i.e.

$$\mathcal{L}(y^*, \hat{y}) = \frac{1}{2}(y^* - \hat{y})^2, \tag{2.3.5}$$

where the coefficient of $\frac{1}{2}$ is added for mathematical convenience for computing the derivative. Squared-error is used when the goal is to predict real-valued targets, i.e. for *regression* problems.

2.3.2.2 Hinge-loss

The hinge-loss is used for so-called **maximum-margin** approaches to *classification* (as opposed to, for instance, maximum likelihood). The hinge loss between a prediction and a

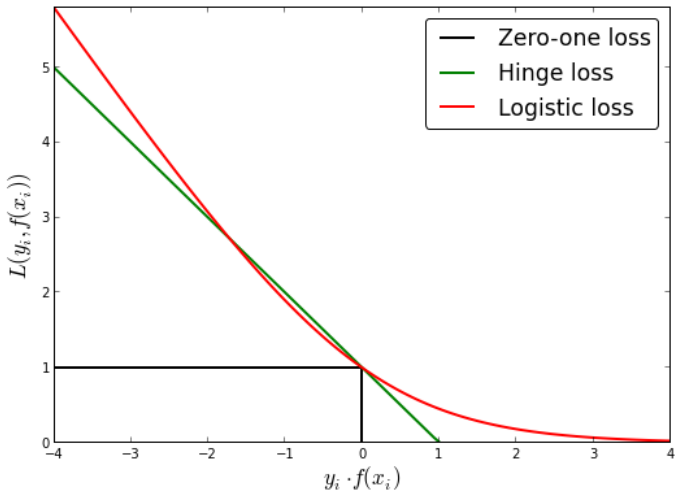


Figure 2.6: Illustration of the hinge loss and the log loss, compared to the (non-differentiable) zero-one loss function. Note that $f(x_i) = \hat{y}_i$, the prediction of the model for input x_i.

target is 0 when the prediction is correct, and increases linearly with the prediction $\hat{y} \in \mathbb{R}$ when it is incorrect if the goal is to predict $y^* \in \{-1, +1\}$:

$$\mathcal{L}(y^*, \hat{y}) = \max(0, 1 - y^* \cdot \hat{y}) \tag{2.3.6}$$

The name "hinge loss" refers to the geometrical shape of the linearly increasing function, which somewhat resembles a door on a hinge (see Fig. 2.6).

In this dissertation we will consider a particular **noise-contrastive hinge loss** in § 3.4.3.4, where the goal is to maximise the *margin* or difference $(\hat{y} - \hat{\hat{y}})$ between the prediction for the correct label $\hat{y}$ and the prediction for a stochastically-selected noisy label $\hat{\hat{y}}$:

$$\mathcal{L}(y^*, \hat{y}, \hat{\hat{y}}) = \max(0, 1 - (\hat{y} - \hat{\hat{y}})) \tag{2.3.7}$$

$$= \max(0, 1 - \underbrace{\hat{y}}_{\text{Maximise}} + \underbrace{\hat{\hat{y}}}_{\text{Minimise}}) \tag{2.3.8}$$

where we have highlighted that maximising this margin implies **maximising** the prediction for the true label and *minimising* the prediction for the noisy label.

2.3.2.3 Cross-entropy

Cross-entropy loss (also called log-loss or logistic loss) is a theoretically well-motivated loss function for training models with probabilistic outputs, although it can be motivated for different reasons. It is defined as maximising the logarithm of the probability assigned

by the model to the correct labels, or alternatively, minimising the negative log-likelihood (NLL) of the probability that the model assigns to the correct labels (see Fig. 2.6 for an illustration). If the goal is to predict a binary label $y^* \in \{0,1\}$ (i.e. binary classification), this can be written as

$$\mathcal{L}(y^*, \hat{y}) = -y^* \log \hat{y} - (1 - y^*) \log(1 - \hat{y}) \tag{2.3.9}$$

where $\hat{y} = P_\theta(y^* = 1|x)$. Note that the two coefficients follow from the fact that $P(y = 1|x) + P(y = 0|x) = 1$. We can generalise this idea to multi-label classification if we write the output labels as a binary *one hot vector* $y^* \in \{0,1\}^k$. Then

$$\mathcal{L}(y^*, \hat{y}) = \sum_{i=1}^{k} \underbrace{-y_i^*}_{\text{labels}} \underbrace{\log \hat{y}_i}_{\text{predictions}} . \tag{2.3.10}$$

where we see that for $k = 2$ (binary classification) 2.3.10 reduces to 2.3.9.

More importantly, 2.3.10 reveals the source of the name cross-entropy: Recall from § 2.1.2 that for a distribution P and Q, the **cross-entropy** is defined as $H(P,Q) = \mathbb{E}_P[-\log Q]$. Furthermore, notice that when we minimise 2.3.10 over the training distribution $P(x,y)$, the sum over the label-weighted log-probabilities can be written as an expectation over the training label distribution, and hence we can view the cross-entropy loss as minimising the cross-entropy between the model and the data distribution.

Indeed, one can furthermore easily show that cross-entropy is equal to the Kullback-Leibler divergence (a distance metric over distributions) between P and Q, plus an additive constant (the entropy of P): $H(P,Q) = H(P) + D_{KL}(P||Q)$. Therefore, the cross-entropy loss minimises the Kullback-Leibler divergence between the *model distribution* P and the *data generating distribution* Q. In other words, it is making the model distribution more similar to the data distribution, which is exactly the goal of training.

2.3.3 Training: Gradient-based training

The loss function quantifies the model's current *progress* during training. In order to reduce the loss function when training neural networks, the most common method is to use gradient-based optimisation where at each training step t, we update each model parameter θ by taking a step along the negative direction of the gradient of the loss function $\mathcal{L}$ with respect to θ, i.e. $-\frac{\partial \mathcal{L}}{\partial \theta}$. We therefore need to be able to calculate the derivative (gradient) of the loss function with respect to every model parameter. For that we will use the Backprop algorithm (see § 2.3.5). Once we have the gradient, the most widely used method for training neural networks is stochastic gradient descent, where at time t we select a random training pair $x^{(t)}, y^{(t)}$, and for each parameter θ we apply the following update rule

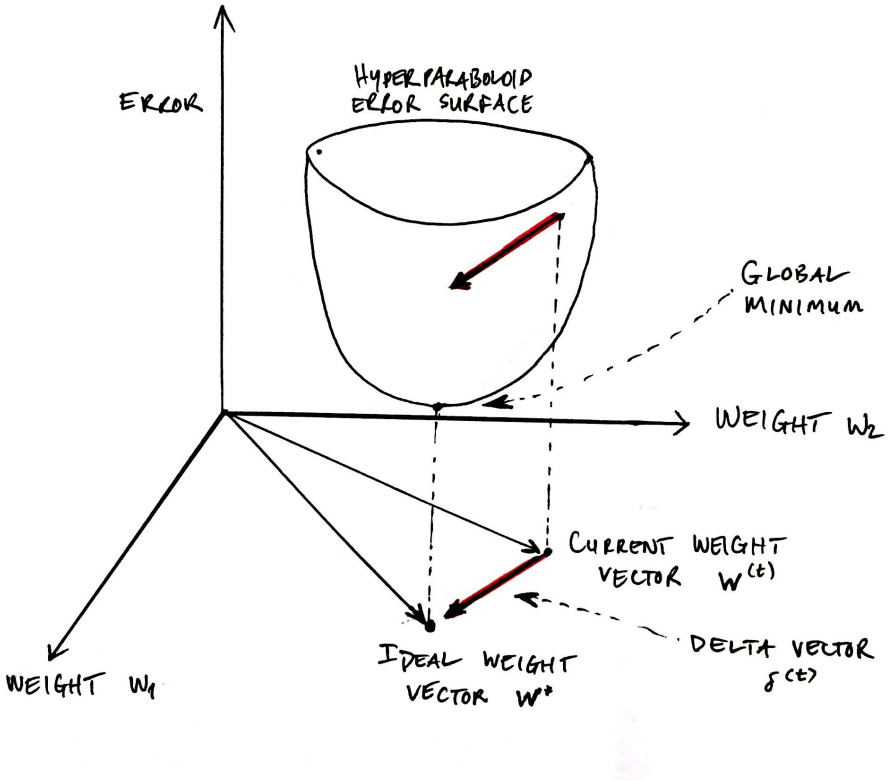


Figure 2.7: Illustration of learning as descending down an error surface by adjusting the model weights.

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \underbrace{\frac{\partial \mathcal{L}}{\partial \theta}}_{\triangleq \delta^{\theta}} (x^{(t)}, y^{(t)}) \tag{2.3.11}$$

where α is called the *learning rate*, and where we are making it clear that the gradient is a function that is evaluated at the specific training example. Note that the gradient of the output with respect to parameter θ will later be defined as δ^{θ} when we discuss the backpropagation algorithm.

We can therefore view learning as optimisation of a loss or error function. This can be visualised for the squared-error as descending down a quadratic error bowl, as shown in Fig. 2.7. At each time step we compute the stochastic estimation of the gradient and then take a small step down the surface of the error bowl by making a small adjustment to the model weights in the direction of the negative gradient (delta).

2.3.3.1 Automatic learning rate selection

In general, setting the learning rate α for neural networks during training can be a black art. Recently, at least two methods were proposed for automatically tuning the learning rate during training, namely Adagrad [23] and Adadelta [24].

We make use of the Adagrad technique. Adagrad adjusts the per-parameter learning rate at time t for parameter θ_i inversely proportional to the squared gradients of θ_i up to time t. I.e. for each parameter θ_i we maintain a separate variable g_i, and adjust the learning rate as follows during training:

$$g_i^{(t)} = g_i^{(t-1)} + \left(\frac{\partial \mathcal{L}}{\partial \theta_i}\right)^2 \tag{2.3.12}$$

$$\alpha_{\theta_i}^{(t)} = \alpha^{t0} / \sqrt{g_i^{(t)}} \tag{2.3.13}$$

where α^{t0} is a global learning rate that is set at the start of training, usually to 1. Notice that this has the effect of decreasing the learning rate for parameters that are frequently updated, and increasing the learning rate for infrequently updated parameters (relative to the other parameters). Computationally, it slows down learning since there is extra bookkeeping work to be done. However, it often tends to speed up the learning process (by bringing the cost function down faster), presumably due to this self-regulating effect.

2.3.3.2 Early stopping with patience

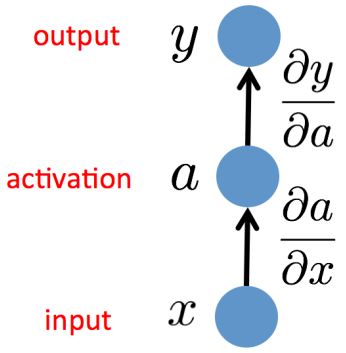
In order to avoid overfitting the training set, we may furthermore employ a standard technique called early stopping [25]. Early stopping is a type of regularisation which prevents the network from finding a solution which will not generalise well, by regularly testing the current state of the network parameters on a held-out validation set, and terminating learning once the performance on the validation set starts to deteriorate, since this is generally a good indication that the network is starting to overfit.

However, neural networks may represent highly nonlinear objective functions with multiple local minima, and what may appear to be overfitting might simply be a plateau in the error landscape. Training with *patience* helps in this case by not stopping training the moment the network's validation error starts to increase, but to continue training for a limited budget of extra time ("patience") with the hope of reaching a better solution, and returning the last best state of the network parameters in the case where such an improvement is not found.

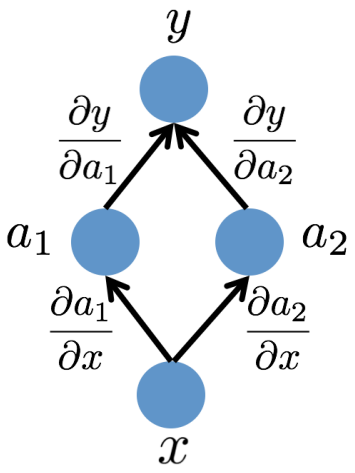
2.3.4 Training: Computational flow graphs

This section considers how one can represent mathematical expressions as graph structures. This is a useful model for thinking about both the inference process (fprop, § 2.3.1) and the training (backpropagation, § 2.3.5) process of neural networks.

Given a mathematical expression defined in terms of computations performed on variables, one can represent it as a directed acyclic graph where each node in the graph represents the result of an intermediate step of the computation, and each edge in the graph



(a) A computational flow graph representing the arbitrary function $y = g(a) = g(f(x))$.



(b) A computational flow graph where the output variable y is a function of more than one variable.

Figure 2.8: A visual illustration of computational flow graphs where nodes represent the results of a computation and edges represent dependencies between computations.

represents a computational dependency. For example, consider the expression $y = x^2$. One could think of this as a graph with an input node x and an output node y, where there exists an edge (dependency) between the two. Likewise, for some arbitrary function $z = g(f(x))$, one could imagine a graph with three nodes, x, a, and y, where a stores the intermediate result $a = f(x)$ and $y = g(a)$.

To highlight the connection with neural networks, one could think of x as the *input*, a as the *activation* and y as the *output* of some neuron (see Fig. 2.8a). But for now, let us consider how this formalism allows us to think about how a change in one variable affects a change in its dependent variables, by considering the derivatives $\frac{\partial y}{\partial a}$ and $\frac{\partial y}{\partial x}$.

Since $y = g(a)$, a change of Δa would result in a change of $\frac{\partial y}{\partial a}\Delta a$ in y. Likewise, a change of Δx would result in a change of $\frac{\partial a}{\partial x}\Delta x$ in a. We can combine these two derivatives with the help of the chain rule of differentiation to derive how a change in x would affect y as follows

$$\Delta y = \frac{\partial y}{\partial a}\frac{\partial a}{\partial x}\Delta x. \tag{2.3.14}$$

If we furthermore introduce more dependents for y, note that the value that y takes on simply becomes the sum over the intermediate paths connecting the dependent variables to y, i.e. $y = \sum_{i=1}^{n} g(a_i) = \sum_{i=1}^{n} g(f_i(x))$. Likewise, the derivative simply decomposes as the sum over all the n intermediate paths in the graph as well (see Fig. 2.8 for an illustration where $n = 2$):

$$\Delta y = \sum_{i=1}^{n} \frac{\partial y}{\partial a_i} \frac{\partial a_i}{\partial x} \Delta x \tag{2.3.15}$$

It should be clear at this point that *making a prediction* in a neural network (inference, or *forward propagation*, § 2.3.1) corresponds to computing the values of all the dependent variables in a flow graph representation of the network, starting from the inputs. Likewise, *training* a neural network (*backpropagation*, § 2.3.5) corresponds to computing the derivatives of dependent variables in a flow graph. This is precisely how the popular Python package `Theano` implements neural networks [26].

2.3.5 Training: Backpropagation derivation

Backpropagation [27,28] (and see [22] for an algorithmic overview) is an efficient algorithm for computing the gradients of the loss function with respect to every model parameter θ_i by proceeding in the reverse direction of the fprop process, and efficiently reusing parts of the gradients along the way.

Backpropagation is used for training all the models presented in this dissertation, so we will discuss the algorithm in detail. We start by considering only two connected units in two consecutive layers of a network, and then generalise the result to a feedforward network of arbitrary depth.

Consider a logistic neuron $y = g(wx + b)$. The computations performed by this neuron corresponds directly to the flow graph illustrated in Fig. 2.8a, where we have shown that, by the chain rule, the derivative of y with respect to the weight w connecting x and a, can be expressed as

$$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial a} \frac{\partial a}{\partial w} \tag{2.3.16}$$

$$= \delta^a \frac{\partial a}{\partial w}. \tag{2.3.17}$$

where we have defined the **error delta** on layer l, $\delta^l \triangleq \frac{\partial y}{\partial a_l}$. In other words, δ^l is the fraction of the total output error for which the node at layer l was responsible, *before the application of its activation function* g. In general, we will use δ^θ to represent the derivative of the output error with respect to layer or parameter θ.

With that in mind, let us now scale up from one neuron and consider how the gradients flow through a series of two connected neurons in two consecutive layers of a feedforward neural network (see Figure 2.9). In particular, let us consider the error delta at layer i:

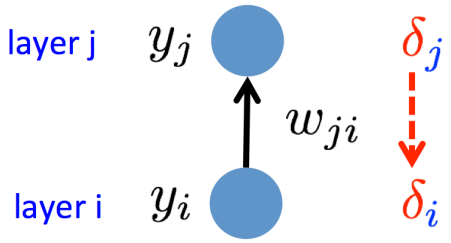


Figure 2.9: Two neurons possibly as part of a larger network. Neuron j takes as input the output of the neuron below $y_j = g(w_{ji}y_i + b_j)$.

$$\delta^i \triangleq \frac{\partial \text{out}}{\partial a_i} \qquad \triangleright \text{ Definition} \qquad (2.3.18)$$

$$= \frac{\partial \text{out}}{\partial a_j}\frac{\partial a_j}{\partial a_i} \qquad \triangleright \text{ Chain rule} \qquad (2.3.19)$$

$$= \delta^j \frac{\partial w_{ji}y_i + b_j}{\partial a_i} \qquad \triangleright \text{ Definition} \qquad (2.3.20)$$

$$= \delta^j \frac{\partial}{\partial a_i}\left(w_{ji}g(a_i)\right) \qquad (2.3.21)$$

$$= \delta^j w_{ji} g'(a_i) \qquad (2.3.22)$$

We are therefore able to derive a **delta recurrence equation** which relates the *total* error contribution (delta) from the output layer that reaches layer i in terms of the *local* error delta on the layer j above. This is what will allow us to come up with an efficient algorithm for computing the derivatives of the output loss with respect to all lower weights.

Notice that in order to compute the gradients for some training sample (x, y), we require the *activations* a_i for all nodes in the network. Therefore, the backprop algorithm proceeds by running `fprop` on the input data x to obtain the node activations a which it stores, and then computes the output loss to obtain the output error delta which it then propagates backwards through the network by the repeated application of the delta rule.

The backprop delta recurrence is intialised at the output layer for $\delta^{\text{out}} = \frac{\partial L}{\partial \text{out}}$, i.e. as the derivative of the loss function with respect to the output layer. For the squared-error loss (§ 2.3.2.1) it is easy to show that $\delta^{\text{out}} = \frac{\partial}{\partial \hat{y}}\frac{1}{2}(y^* - \hat{y})^2 = \pm(y^* - \hat{y})$, or in words: *(true - predicted)*[1]. For the cross-entropy loss (§ 2.3.2.3) it takes a bit of work, but as we will show in § 4.1.3, the output delta is indeed also (true - predicted).

The backpropagation algorithm (see § 2.3.6 for the implementation) is therefore able to compute the gradients on all model weights with one forward pass and one backwards pass through the network, for a runtime of $O(n)$ for n weights. This is a great improvement

[1] The sign depends on the order of the subtraction, but is irrelevant in terms of the squared error.

over the naive computation of the gradients which considers all exponential combinations of paths from input to output nodes, and runs in $O(n^{L-1})$ for L layers.

2.3.6 Training: Backpropagation algorithm

The Python code shown in Listing 2 is a direct implementation of the backwards application of the delta-rule derived in 2.3.18. The function takes as input the activations of the different layers of the network, obtained by running the `fprop()` function presented in § 2.3.1.

Listing 2 Python code implementing the back-propagation algorithm for a feedforward neural network.

```
def backprop(target, acts, ys, weights):
  y = ys[-1]
  delta = (target - y) # intialise delta
  grads = []
  num_layers = len(weights)
  for i in xrange(num_layers - 1, -1):
    Wgrad = dot(delta, ys(i-1).T) # outer product
    bgrad = delta
    grads.append((Wgrad, bgrad))
    # update for next iteration, elem-wise multiplication
    delta =  dot(delta, weights[i].T) * dsigmoid(acts(i-1))
  grads.reverse()
  return grads
```

2.3.7 Unsupervised models

Supervised models learn to predict a label y from an input x. Unsupervised models only receive the inputs x, and attempt to automatically discover the structure of the input space for instance by clustering. In the context of neural networks, two of the most popular methods for unsupervised learning are the probabilistic restricted Boltzmann machine (see for instance Hinton [29]), and the non-probabilistic autoencoder and its variants (see for instance Vincent [30]). We will not focus on the RBM in this dissertation, but the unsupervised principles behind autoencoders have close ties to the unsupervised distributional methods for learning word representations, so we will mention the model in broad strokes here.

An autoencoder learns an *encoding* transformation $h = \sigma(\mathbf{W}x)$ of its input x into a hidden representation h, along with a *decoding* transformation $\tilde{x} = \mathbf{V}h$, such that the *reconstruction error* $\mathcal{L} = ||x - \tilde{x}||_2^2$ between the original x and the reconstruction $\tilde{x}$ is minimised.

In the process, the model learns a new representation for the inputs (in terms of the activations on the hidden layer). One can show that this new representation space spans

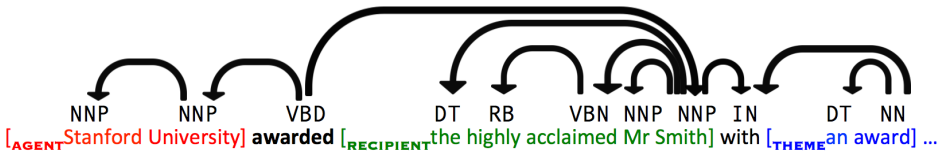


Figure 2.10: An example of the types of linguistic structures that we are interested in predicting in natural language processing.

the main directions of variation in the original data. Due to the squared-error, there is also a close relationship to matrix-decomposition methods like principal components analysis (§ 2.4.2.2). However, in autoencoders the induced transformation is not necessarily orthonormal, as in matrix-decomposition methods.

2.4 Statistical Natural Language Processing

2.4.1 Overview

In natural language processing (NLP) systems the goal is to build a computational system that can map raw input strings of text to output linguistic structures that encode the meaning of the text [5]. For example, as illustrated in Fig. 2.10, given the input text

Stanford awarded the highly acclaimed Mr Smith with an award,

the goal might be to label words with their parts of speech (POS), e.g. nouns `NN*` and verbs `VB*`, or to extract the dependency relationships between words shown as arrows from *head words* to their *dependents*, or to extract the semantic components of the sentence (*who* does *what* to *whom*).

Most modern approaches model this as a machine learning problem where we try to predict a set of *labels* $y \in \mathcal{Y}$ (e.g. a POS tag) that each input $x \in \mathcal{X}$ (e.g. a word) can belong to. To do this, one defines a *model* that can learn how inputs relate to labels. Generally speaking, jointly modelling inputs and labels is referred to as *generative* modelling, since these models can map both inputs to labels and labels to inputs allowing the model to generate or "hallucinate" new data, from where the name. On the other hand, *discriminative* models learn a conditional mapping from inputs to labels that can assign a real-valued score or probability to the different labels y that a given input x might belong to. I.e. it can discriminate between the different labels, from where the name. Given such a trained model, and a new input x, the prediction problem then reduces to solving the "decoding or argmax problem" of finding the best label $\hat{y}$ for a given input x, e.g. $\hat{y} = \arg\max_y P_\theta(y|x)$ for a probabilistic model.

$\mathbf{x} \in \mathcal{X}$	**the**	**cat**	**sits**	**on**	**the**	**mat**
$r(\mathbf{x})$	$r(x_1)$	$r(x_2)$	$r(x_3)$	$r(x_4)$	$r(x_5)$	$r(x_6)$
$\mathbf{y} \in \mathcal{Y}$	`B-NP`	`I-NP`	`B-VP`	`O`	`B-NP`	`I-NP`
NE Tags	**[the cat]$_{\text{NP}}$**		**[sits]$_{\text{VP}}$**	**[on]$_{\text{O}}$**	**[the mat]$_{\text{NP}}$**	

Table 2.1: Example NLP syntactic chunking task for the sentence "the cat sits on the mat". $\mathcal{X}$ represents the words in the input space, $\mathcal{Y}$ represents labels in the output space. $r(\mathbf{x})$ is a feature representation for the input text $\mathbf{x}$ and the bottom row represents the output named entity tags in a more standard form.

In this dissertation we only focus on discriminative models where $P_\theta(y|x)$ is parameterised in terms of a *feature representation* $r(x)$[2] which captures the salient, discriminative aspects of the input domain $\mathcal{X}$ with respect to the labels y. In this class of models, most of the novelty in new approaches relate to how the feature functions $r(x)$ are defined.

As a concrete example, consider the task of syntactic chunking, also called "shallow parsing" [31]: Given an input string, e.g.

"the cat sits on the mat",

the chunking problem consists of labelling segments of a sentence with syntactic constituents such as noun or verb phrases (NPs or VPs). Each word is assigned one unique tag, often encoded using the BIO encoding[3]. We represent the input text as a sequence of words $x_i \in \mathbf{x}$, and each word's corresponding label is represented by $y_i \in \mathbf{y}$ (see Table 2.1). Given a feature-generating function $r(x_i)$ and a set of labelled training pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, the task then reduces to learning a suitable mapping $s_\theta : r(\mathcal{X}) \rightarrow \mathcal{Y}$ by tuning the model parameters θ until the model is successful at predicting the training labels.

The task of coming up with good feature generating functions $r(x)$ is referred to as *feature engineering* and typical feature functions for this task would produce an output vector where individual components of the vector is set to 0 or to 1 depending on the presence of certain binary indicators. Examples for this task include for example "does the word contain a capital letter", "is the word contained in a gazetteer", etc. This approach works well, but it is labour-intensive and generally hard to come up with features that can generalise well across many different tasks. Therefore these feature functions are often only useful for one specific task. This problem is compounded even more when the goal is to come up with features that can capture important aspects related to the task *across different languages*, i.e. cross-lingual prediction.

In this dissertation, our goal is to *automatically and efficiently learn distributed representations of natural language that can generalise across a variety of NLP tasks and*

[2] This is usually denoted as $\phi(x)$, but for consistency in this dissertation we will refer to the feature representation of object x as $r(x)$ to emphasize that it is a function, or simply r_w to refer to result of applying r to x.

[3] E.g. `B-NP` means "begin NP", `I-NP` means "inside NP", and `O` means other/outside.

languages. There is a rich history of distributional methods that can be used for learning useful representations, and in the next sections we will discuss the most well-known methods, namely Brown clustering and latent semantic analysis. However, in the rest of this dissertation, we will approach this problem by building upon advances in the field of neural language models, which will be discussed in detail in Chapter 3.

2.4.2 Representation Learning in NLP

In representation learning for NLP, the goal is to *learn* how to represent the input text in order to be successful at predicting the output labels. This means that during training, we are not not only learning the weights θ of the classifier $P_\theta(y|r(x))$ to produce better predictions, but we are also learning how the inputs are represented to the model, i.e. we are also learning $r(x)$.

If we had unlimited labelled training data (x, y pairs), one could learn the representations end-to-end to improve prediction of the labels. However, due to the cost of obtaining labelled data, unlabelled data is often much more easily obtainable. Therefore the representation learning step is usually performed in an unsupervised setting (using raw text documents).

Distributional representation learning methods in NLP can be seen as learning to encode as a feature vector how words or phrases in one or more languages relate to each other, in order to learn soft equivalence classes which allow one to generalise. E.g. if word w_i in class c_j has certain (latent) properties or we know its annotated label, then one can assume that word w_k in the same class c_j shares similar properties. This allows one to **transfer** the supervised information one has available for w_i to w_j.

2.4.2.1 Brown clustering

Brown clustering was introduced as a technique for clustering words into hard equivalence classes in order to combat data sparsity when training a language model [32]. Learning the conditional language modelling probabilities $P(w_i|w_j)$ requires estimating a table of $O(V^2)$ probabilities from the data for V words in the vocabulary (worst-case, although many of these will be unobserved). When data is limited, some of the rare transitions may not be observed and hence assigned a zero probability under the maximum likelihood inductive principle. However, if words are assigned to C classes, then one can rewrite this probability as

$$P(w_i|w_j) = P(C(w_i)|C(w_j))P(w_i|C(w_i)) \tag{2.4.1}$$

where $C(w_i)$ denotes the class that word w_i belongs to. Note that now we only need to estimate the $O(C^2)$ conditional class-**transition** probabilities and the $O(VC)$ word-**emission**

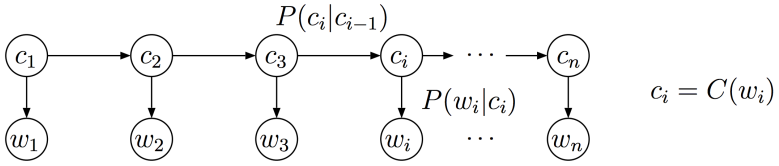


Figure 2.11: A graphical illustration of a class-based, bigram language model (note the similarity to a hidden Markov model) [33].

probabilities. This possibly represents a great reduction if the number of classes C is much less than the number of words V. The algorithm starts by assigning each word to its own class. It then iteratively and hierarchically merges two classes in order to maximise the total log-likelihood assigned by the class-based bigram language model (2.4.1) to the training data. E.g. "saw" and "knife" might be merged into the same class since they often occur in the same contexts. The total runtime of the algorithm is $O(VC^2)$.

Since we do not know the actual word to class assignments and due to its bigram nature, this model can be thought of as a hidden Markov model (see Fig. 2.11). The output of the algorithm is a binary clustering over the vocabulary with the words as the leaf nodes. Each internal node is considered a cluster that contains the words in that subtree. The "features" for each word are the concatenated binary codes consisting of the left (0) /right (1) traversal down the tree, starting from the root.

Brown features have been successfully used in many applications, including named entity recognition (NER) [33–35], probabilistic context-free grammar (PCFG) parsing [36], dependency parsing [37,38], and semantic dependency parsing [39], and was more recently compared to neural word embedding features (what we will be focusing on in this dissertation) by Turian et al. [9].

2.4.2.2 Spectral methods

Another important class of representation learning algorithms make use of so-called spectral techniques, better known as matrix-decomposition techniques, and specifically, singular value decomposition (SVD). Latent semantic analysis (LSA) [1] was the first method to popularise this approach. LSA takes a $(D \times V)$-dimensional **document-term-matrix** of (document, word)-observations, and applies singular value decomposition to yield a reduced $(D \times k)$-dimensional matrix, where each row represents a lower-dimensional vector-representation which characterises the 'concepts' or topics that are present in the document.

Intuitively, this works because SVD finds the directions of maximum variance (change) in the space of words, and then learns a linear projection which projects words that tend to co-occur frequently onto the same components (principal vectors) in the reduced dimensional space. This process tends to capture the topical similarity between words, since

words that co-occur tend to share meaning.

More technically, SVD is an efficient linear algebra technique for decomposing an $(m \times n)$-dimensional matrix A as the product of three lower-rank matrices (i.e. it produces a *low-rank approximation* of the matrix): the left and right singular vectors U and V, and the covariance matrix Σ, i.e.:

$$\underbrace{A}_{(m\times n)} \overset{\text{SVD}}{=} \underbrace{U}_{(m\times k)}\ \underbrace{\Sigma}_{(k\times k)}\ \underbrace{V^\top}_{(k\times n)} \tag{2.4.2}$$

$$= \sum_{i=1}^{k} \underbrace{\sigma^i}_{\text{scalar}}\ \underbrace{\underbrace{u^i}_{(m\times 1)}\ \underbrace{(v^i)^\top}_{(1\times n)}}_{(m\times n)} \tag{2.4.3}$$

When LSA is applied to a (document, term)-matrix, it produces compact representations of the documents (which can be used for information retrieval). However, when it is applied to a (term, document) matrix, it accomplishes the inverse: it produces compact vector representations of *words*, based on how words tend to co-occur in documents.

Furthermore, one is free to change the definition of the context from more global (documents) to more local (windows of surrounding text). Word representations derived in this way are referred to as **eigen-words** [40, 41], since the words are represented by the projections of their contextual co-occurrence patterns onto the reduced set of principal eigenvectors of variation.

Chapter 3

Neural Word Embedding Models

This chapter introduces neural language models (NLMs), the main modelling workhorse employed in this dissertation for representation learning. Throughout, we provide a detailed overview of the different NLM architectures as well as a detailed, tutorial-style overview of all the popular techniques for training these models. We furthermore introduce a novel geometrical framework for understanding the training of word embedding models, particularly from a representation learning point of view. This ties together different training methods and highlights the connections between these models and other distributional methods for learning word representations, such as latent semantic analysis (§ 2.4.2.2).

This chapter is based on a tutorial that I have presented at Carnegie Mellon University in Dec 2012, Microsoft Research (Israel) in Nov 2013, and at University of Copenhagen in Nov 2014[1].

3.1 Introduction

The goal of language modelling is to learn a model that can accurately capture the statistical structure of a natural language, by modelling the language as a sequence of words. If such a model is accurate, it would enable one to make predictions about which words are more likely to occur in certain contexts. In particular, we are interested in predicting what *target word*, that we will denote by w_t, is most likely to follow a given sequence of previous *context words*, that we will denote by h (for 'history'). See for example Fig. 3.5, where the goal is to predict the target word "mat" from the previous context words "the cat sits on the".

The most popular models for this task are called n-gram language models [42]. An n-gram model models the probability distribution over the target word given the $(n-1)$ previous words. Therefore a *uni*-gram model ($n = 1$) represents the unconditional probabilities (0 previous words), a *bi*-gram model ($n = 2$) considers one previous word, etc. n-gram

[1] See www.stephangouws.com/talks.

models are in essence tables of discrete conditional probabilities $P(w_t|h)$ estimated from data by counting the number of times we have observed w_t in the $(n-1)$-length context h.

Despite its unquestionable success and pervasiveness, there are several issues with this approach, most notably that it suffers from the *curse of dimensionality* [43], where the possible number of parameters in this model (conditional probabilities) grows exponentially in the number of words V in the vocabulary and the context-length $n-1$, i.e. as $O(V^{n-1})$. This creates an issue of data-sparsity, where word-context combinations that were never observed in the training set are assigned a probability mass of 0. An active area of research has been to design smoothing techniques (regularisers, § 2.2.2) to redistribute probability mass in these models in order to generalise beyond the observed training data [42].

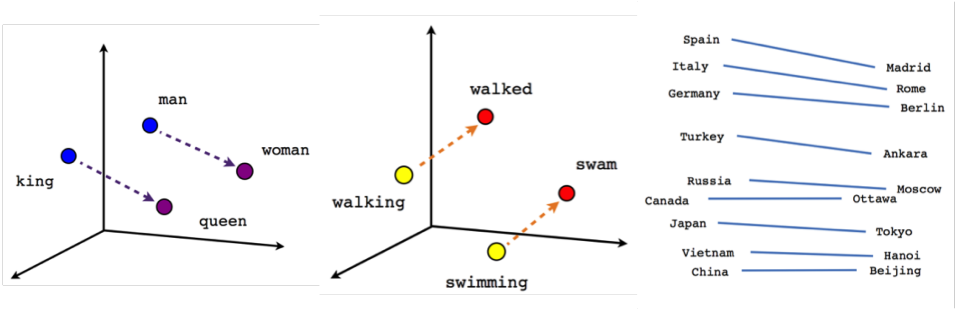
Neural language models (NLMs) are an alternative to n-gram models. An NLM learns a distributed feature representation (a vector, discussed below) for each word in the vocabulary, and then uses a neural network function to map the combined features of the context words to predict the next word in the sequence of training words. A *distributed representation* [44] of a symbol is a tuple or vector where the *meaning* of the symbol is represented by (distributed across) a combination of different features in the vector. In the discrete binary case, it therefore optimally corresponds to a binary encoding and an N-dimensional encoding can represent 2^N different distinct symbols. There is *parameter sharing* in that many symbols can share (be represented by) combinations of the same components of the representation. This stands in stark contrast to a *local representation* of a symbol, also called a one-hot representation (as is used in for instance n-gram models), where only one dimension is active ('on') corresponding to the index of the symbol in an N-dimensional vocabulary. An N-dimensional local representation can therefore represent only N different symbols.

NLMs have two very attractive properties, compared to n-gram models:

1. Firstly, they overcome the curse of dimensionality by not associating an individual parameter with each possible combination of word-context pairs[2], but instead NLMs *share parameters* between inputs. This has the very desirable effect that the number of parameters grows only linearly with the context-length, i.e. as $O(V + (n-1))$, and not exponentially as with n-grams.

2. Secondly, the smooth nature of the functions learned by neural networks introduces an implicit regularising effect, in that similar inputs get mapped to similar outputs. This has the effect that an NLM can naturally generalise to word combinations it has never observed together by leveraging what it has learned from similar words in the training data.

[2] In n-gram models this *is* the conditional probability.



(a) Male-female.

(b) Verb tense.

(c) Country-capital.

Figure 3.1: Visualisations of the different linguistic relationships that neural language models have been found to learn and encode in its embedding space in an unsupervised manner from large collections of text [15].

In an NLM, discrete input word identifiers are mapped into a learned, distributed vector space of meaning by assigning with each word a learned, real-valued feature-vector referred to as the word's *embedding*. The name stems from the fact that the high-dimensional discrete (local) symbols are *embedded* into a low-dimensional vector space. Neural networks tend to map nearby inputs to nearby outputs. Therefore, in order for the network to accurately predict similar words in the same contexts, e.g. *the cat* ***{sits, runs, purrs}***, it will update the representations for similar words to be close (similar) in the induced vector space. We will discuss this concept in much more detail in the rest of this chapter. Different directions in this space correspond to different learned, latent properties that the network uses to represent the properties related to the meanings of words.

It has been shown that NLMs are particularly good at extracting relationships between words purely from their distributional information [15]. Fig. 3.1 shows that the models can learn to encode male-female relationships, syntactic information about words and even seeming *semantic* information such as the `country-capital` relationship ("Berlin" is to "Germany" as what is to "France"?). Since these relationships capture useful general properties about language, they have been found to be very useful as features for a classifier for predicting other aspects of language, like part-of-speech tags [8], named-entities [9], etc.

Since the introduction of the original neural probabilistic language model (NPLM, Section 3.3.1), several other model architectures and training methods have been proposed. For representation learning, we are primarily interested in learning the word embedding features. The rest of this chapter will discuss in detail the different model architectures and training methods that exist for efficiently learning word embeddings. In order to understand how these models relate to one another, it is useful to break up the modelling process into three different stages (see Fig. 3.2):

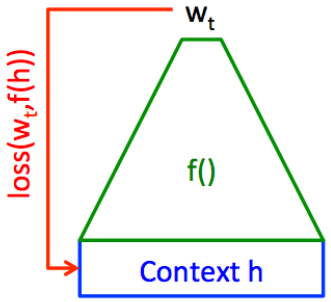


Figure 3.2: A flow-diagram illustrating the three important components of the most popular neural word embedding models: input context words, the network function $f_\theta(h)$ that maps input words to predict output words w_t, and the loss function $\mathcal{L}$ used to train the model.

1. Firstly, how the context words h are selected for a target word w_t (window of previous words, surrounding window of words or unbounded history),

2. secondly, how the network $f_\theta(h)$ maps the context word representations to predict w_t, and finally,

3. what objective or loss function $\mathcal{L}(w_t, f_\theta(h))$ we use to train the model.

In this chapter we review the most important NLM architectures in terms of the above three considerations. In the next sections we will discuss the similarities and differences of NLMs to other distributional methods for learning representations of natural language, in particular latent semantic analysis (LSA, § 2.4.2.2), by viewing all these techniques along the same considerations introduced above. Since we are ultimately interested in learning the word embeddings, we will furthermore analyse the properties and training dynamics of the different loss functions in the case of the most popular word embedding models to obtain a better geometrical understanding of how these models are trained (§ 3.5).

3.2 Distributional Representation Learning Methods

Distributional semantics defines the relationships between linguistic units (words, phrases, sentences) based on how frequently they appear together in large corpora, i.e. based on their distributional properties. The underlying idea is based on the Distributional Hypothesis [45] which can be summarised as *"linguistic items with similar distributions have similar meanings"*. In other words, the Distributional Hypothesis claims that there exists a relationship $f : W \times C \rightarrow M$ between the co-occurrence statistics C and the meanings M of some words W.

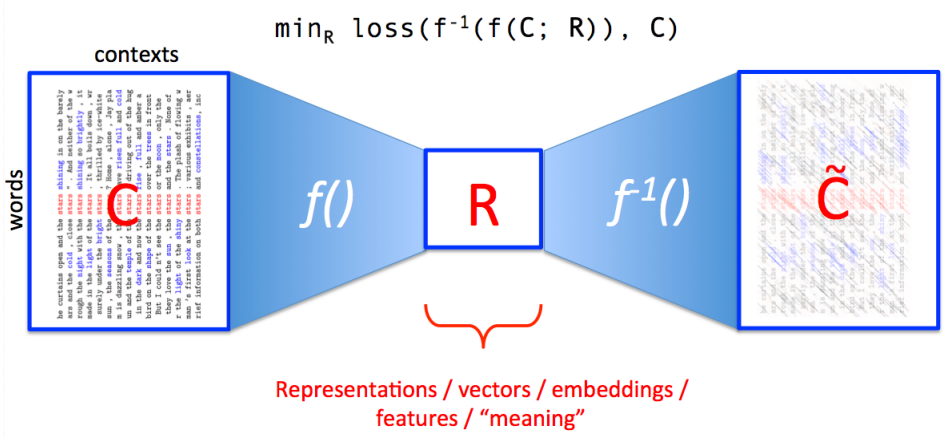


Figure 3.3: Distributional methods for learning semantics (linguistic meaning).

In essence, distributional methods represent an unsupervised family of techniques that *compress* or summarise a word's co-occurrence statistics (how frequently we see some word in the context of other words) into a representation which captures useful properties of the word's meaning (see Fig. 3.3). Viewed in this way, it is useful to think of these techniques as optimising an unsupervised reconstruction objective function (§ 2.3.7) that aims to capture most of the 'interesting' information that is present in the contexts in which a word is observed, by squeezing the high-dimensional contextual information C through an *information bottleneck* [46] into a low-dimensional representation R, from which it has sufficient information left to reconstruct the original. The training objective can be stated as

$$\min_{R} \mathcal{L}(f^{-1}(f(C;R)),C) \tag{3.2.1}$$

In this formulation, a 'good' representation R is one which causes the least corruption during the reconstruction step.

One well-known technique of this type is Latent Semantic Analysis (LSA, § 2.4.2.2 [1]). LSA applies principal components analysis [47] or singular value decomposition to the word-context co-occurrence matrix C to decompose it into three matrices, the left and right singular values $U \in \mathbb{R}^{V \times r}$ and $V^\top \in \mathbb{R}^{r \times n}$ and the diagonal matrix $\Sigma \in \mathbb{R}^{r \times r}$ (with the r eigen-values on the diagonal) which together minimise reconstruction error with respect to the original C (see Fig. 3.4). The singular vectors in U represent the directions of most variation of the contexts in which each word appears (starting with the highest variance component at column 1) and each row-vector $U[i,:]$ represents the projection of the high-dimensional co-occurrence data for each word i onto these principal directions of variation. By reducing the number of components r, one obtains a more compact representation or 'embedding' for each word, at the expense of introducing more error in the reconstruction of C.

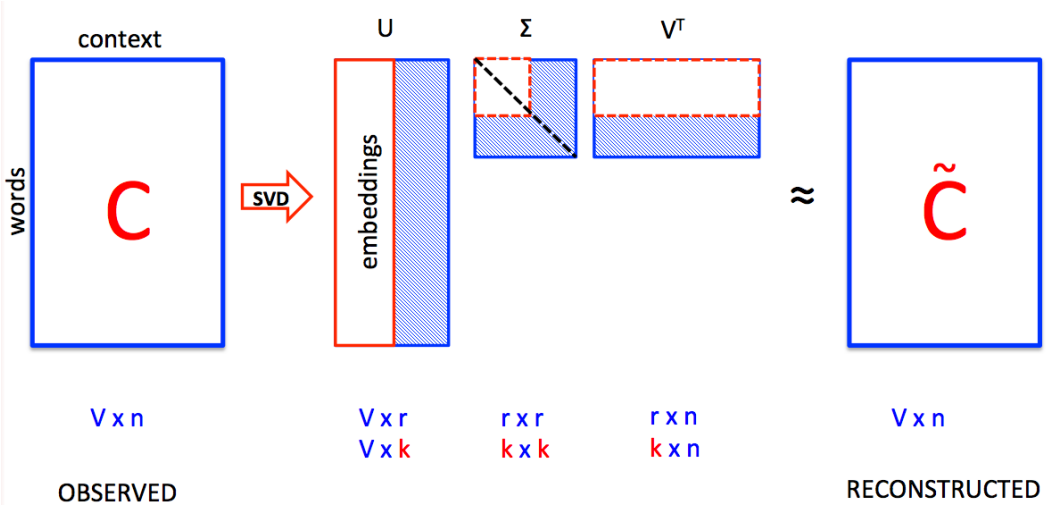


Figure 3.4: Latent semantic analysis [1].

It can be shown that LSA learns an affine function for mapping a word's high-dimensional contextual information into a lower-dimensional space which minimises a squared-error reconstruction cost between the original and the reconstructed data [48]. In terms of 3.2.1, LSA therefore assumes that f is a linear (affine) function parameterised by matrix W, and $\mathcal{L}()$ is the squared-error.

The key insight from this discussion is that *word embeddings* are projections which compress the high-dimensional contextual information in which a word appears into a low-dimensional subspace. To goal is to minimise an expected error function in the low-dimensional space under the training distribution (co-occurrence statistics). This is easily seen for the above case with LSA, where the projection f is affine and the error is the squared-error. *NLMs generalise this by allowing a nonlinear projection (computed by the neural network), an arbitrary error function, and using an iterative, stochastic optimisation scheme.*

In the rest of this chapter we will look in more detail at the various popular loss functions for training NLMs. We will then discuss the similarities between different loss functions when the goal is purely to train high-quality word embeddings.

3.3 A Review of Neural Language Models

In this section we look at the most popular neural language models, namely the original neural probabilistic language model (NPLM), the log-bilinear language model (LBL) which is simple yet effective, hierarchical class-based models which speed up training by clustering the vocabulary, models based on recurrent neural networks which can consider longer context-sizes, and noise-contrastive models for learning word representations which speed up training by using efficient training objectives.

3.3.1 Neural probabilistic language model (NPLM)

The first successful neural language model was introduced by Bengio et al. in a NIPS paper in 2001 [49], and expanded in a subsequent JMLR article [12]. Its most important property was to project each input word into a low-dimensional Euclidian space, and to use a neural network to calculate the conditional language modelling probabilities in this low-dimensional space. The NPLM architecture is visualised in Fig. 3.5.

As with all NLMs, word embeddings correspond to row-vectors that are stacked into a $(V \times K)$-dimensional embeddings matrix, which we will represent with $\mathbf{R}$, and each input word id is *projected* onto its embedding vector. The goal is to update these embedding vectors using end-to-end gradient-based training, therefore we require that the projection step be differentiable. This can be accomplished by representing each word w using a V-length one-hot vector z_w, i.e. a vector with zeros everywhere except at the index position for w. One can then perform the projection step as a (sparse) dot product $r_w = \mathbf{R}^\top z_w$, where r_w denotes the embedding for word w. This projection step is performed for all context words and the resulting embeddings are concatenated to form the activations on the input layer.

The rest of the architecture is a standard feed-forward one-hidden layer neural network with a *tanh*-nonlinearity on the hidden layer and a *softmax* output layer:

$$P_\theta(w_t|h = \{w_1, \cdots, w_n\}) = \frac{\exp(f_\theta(h)^\top q_{w_t} + b_{w_t})}{\sum_j \exp(f_\theta(h)^\top q_j + b_j)} \tag{3.3.1}$$

where b_j is a bias weight which represents the prior unigram probability of observing word w_j, $f_\theta(h)$ represents the network's hidden-layer representation for the context words h, and $q_w = \mathbf{Q}[w,:]$ is the learned *output embedding* for word w, which may or may not be the same as the input embedding $r_w = \mathbf{R}[w,:]$. One reason for this might be to allow the models to learn that the probability of the same word occurring twice $P(w_i|w_i)$ is very low (small dot product), which it cannot do when input and output words use the same vectors [50].

One variant of the model also included linear skip-connections directly from the input layer to the output layer. The authors trained this model by maximising its log-likelihood on a training set using the backpropagation algorithm to compute the updates on the model parameters, but also on the actual word embedding vectors, thereby jointly training model parameters and word representations.

3.3.2 Log-bilinear language model (LBL)

The log-bilinear language model (LBL) was introduced by [51], and made two simplifications to the original NPLM. First, they dropped the nonlinearity on the hidden layer. Second, there are no skip-connections (since the connections from input to output layer are already linear). Although the model was originally motivated and defined as an energy-based model [52] with a bilinear energy function, it is indeed simpler to interpret it as a

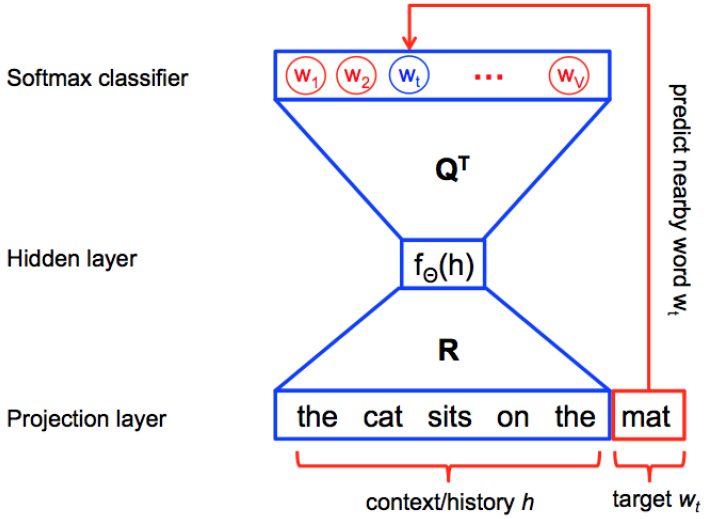


Figure 3.5: Schematic of the original neural probabilistic language model [12], which used a feedforward neural network to predict the probability of the next *target* word from its preceding *context* words. **R** and **Q** are the so-called input and output embeddings. $f_\theta(h)$ maps the input words to the hidden layer representation. Finally, note that the target word is not part of the input, but is in fact the label that the network needs to predict.

simple feed-forward neural network with a linear hidden layer and a softmax output. In this formulation, the activations on the hidden layer constitute the network's predicted embedding of the target word from the context words

$$f_\theta(h = \{w_1, \cdots, w_n\}) = \sum_{i=1}^{n} \mathbf{C}_i r_{w_i} \tag{3.3.2}$$

where $\mathbf{C}_i$ is a learned $(K \times K)$-dimensional matrix for the word at the i-th context-position. The model then predicts the output distribution by comparing the predicted vector $f_\theta(h)$ with the learned embedding vectors of all the words in the vocabulary using the softmax function (3.3.1). Despite its simplicity, this model has been shown to outperform the NPLM in terms of perplexity on several datasets [51]. This is quite an unexpected result given that the model is linear, compared to the nonlinear NPLM. We will look at this in more detail in § 3.4.2.

3.3.3 Hierarchical (class-based) models

Both the NPLM and the LBL models use a softmax output layer to compute the multinomial output distribution over all words. A significant computational problem with the vanilla softmax output function is that it is very expensive to compute both in the forward phase (fprop) and the backwards phase (backprop).

In order to address the high computational cost, [53] introduced the *hierarchical softmax* (HS) function. We will look at the HS in more detail in Chapter 4, but for now we simply note that this approach takes inspiration from work on class-based language models [32]. In short, class-based models make use of the sum-rule of probability (§ 2.1.1) which states that one can decompose the calculation of the probability $P(w|h)$ (3.3.1) in terms of a marginalisation over a new variable c

$$P(w_t|h) = \sum_i P(w_t|h, c_i)P(c_i|h) \tag{3.3.3}$$

where c_i can be interpreted as "class i", and the equation in general as: certain groups of words (classes) appear more frequently in certain contexts, and we can leverage this fact to compute the probability of a particular word to appear in a particular context.

This can result in significant computational savings if we restrict the number of classes that a word can belong to. However it is not trivial to find a good way to partition words into classes, and it also depends if the main goal is to speed up the evaluation of 3.3.3, or improve the perplexity of the model (§ 4.1.5).

In [53] the authors used the WordNet [54] lexical resource to derive the class structure, but this did not work very well. [55] presented and evaluated several different data-driven clustering approaches which produce random, balanced and what they called 'adaptive' class structures. The authors report good results using the adaptive method, which essentially clusters pre-trained embeddings in an offline setting into trees prior to training. In § 4.2 we will develop an efficient algorithm which performs this clustering in an online setting, jointly during training.

Another popular approach is to group words based on their frequency counts. For a one-level class structure (n-ary tree) this amounts to grouping words based on bins of *constant probability mass* [56, 57]. For a multi-level (binary tree) class structure this amounts to constructing a Huffman tree over the vocabulary [58]. We will look at the binary HS in more depth in Chapter 4.

3.3.4 Recurrent models

Language has dependencies which may span across several words. One advantage of the NLMs introduced so far, over n-gram models, is that the number of parameters at the input layer only increases linearly with the context-size. This makes it easier *statistically* to scale to larger context-sizes since less data is required to estimate the parameters, which in turn helps with modelling long-range dependencies. However, feedforward neural networks have no internal memory and a feedforward NLM can therefore only 'see' the $n - 1$ words in the current window that it is considering for predicting the next word.

Instead of increasing the context-size to enable models to view all possible long-range dependencies during prediction, a more elegant approach is to endow the network with

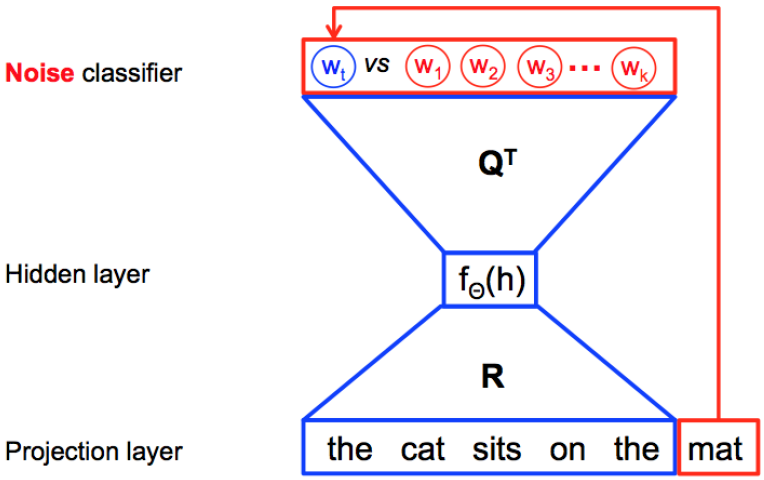


Figure 3.6: Noise-contrastive models attempt to score an observed target word higher than a stochastic sample of k noise words. The scoring function can be non-probabilistic, e.g. Collobert and Weston [8] or probabilistic, e.g. CBOW or skipgram [58, 65].

an internal memory so that it can remember important contextual information over arbitrary time-steps. *Recurrent neural networks* (RNNs) were exclusively designed with this property in mind. An RNN achieves this by using the state of the network at the previous time-step as an additional input for making its prediction at the current time-step. In theory, this allows a network to propagate information across arbitrary time-steps. However, in practice it turns out to be difficult to train RNNs, in part due to the *vanishing and exploding gradient* problem [59]. Some advances have been made over the last few years which improves training [60, 61], but it remains a difficult problem to solve. Despite its successes in character and word-based language modelling [62–64], recurrent models have not proven more useful than feedforward models when the goal is to learn high-quality word features, and therefore we will not focus on recurrent models in this dissertation.

3.3.5 Unnormalised, noise-contrastive NLMs

In this section we look at models where the objective is to be able to successfully predict if a word makes sense in a certain context (i.e. if it is part of the data distribution) or not (i.e. if it is *noise*). We will look at two models, one non-probabilistic and the other probabilistic, but both approaches can be visualised as illustrated in Fig. 3.6.

3.3.5.1 Noise-contrastive rank loss

Up to this point we have considered different probabilistic neural language models. However, in two of the seminal papers that paved the way for neural networks to be used for learning features for NLP tasks, the authors trained a non-probabilistic, noise-contrastive

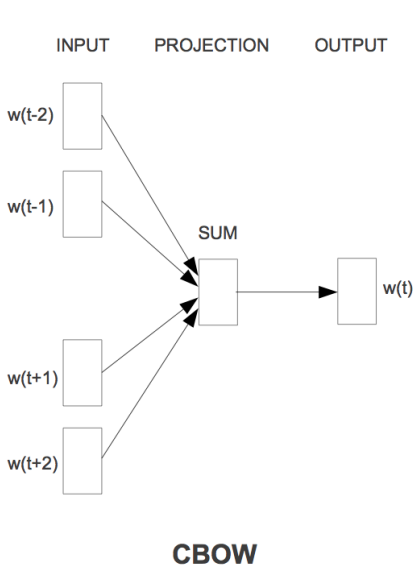


Figure 3.7: CBOW

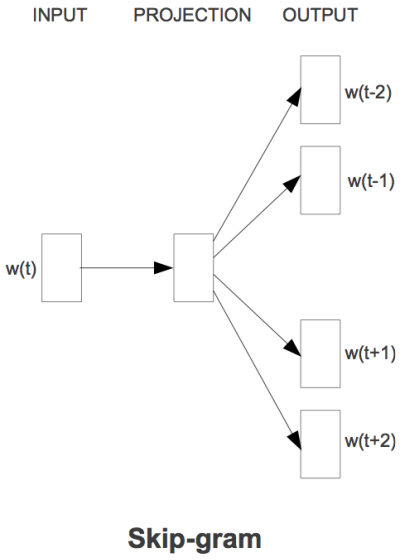


Figure 3.8: Skip-gram.

language model [8, 13]. The objective is to be able to score windows of *valid text* higher than windows of *corrupt text*. Specifically, for each training update, the observed n-gram context h consisting of a sequence of $n-1$ words is used as a positive example. A corrupt, negative example is then created by randomly replacing a word at some fixed position in the observed window with a random word drawn from the vocabulary. The network is then trained to output a real-valued score for the positive example that is higher than the score for the negative example, with some real-valued margin m. This loss will be discussed in more detail in § 3.4.3.4.

Due to its computational efficiency, the authors were able to train this model over all Wikipedia text (±600 million words at the time). They were then able to show that the learned word embeddings can be used for several low- and mid-level NLP tasks, like part-of-speech tagging, word chunking and semantic role labelling. This was the first work to exploit the learned embeddings as features for predicting word-level labels, and in a sense revived the application of neural network models in natural language processing.

3.3.5.2 CBOW and Skip-gram

Mikolov [14, 58] introduced the continuous bag-of-words (CBOW) model (see Fig. 3.7) which further simplifies the LBL (and hence NPLM) in two important ways. First, it completely ignores word-order at the input layer, and computes the context-vector by simply summing over the embeddings of the words in the input window. Second, they use a simplified version of noise-contrastive estimation (§ 3.4.3.3), which they call negative sampling (§ 3.4.3.5), to train the model.

They also introduce the skip-gram model (see Fig. 3.8), which can be seen as the 'inverse' of the CBOW: instead of trying to predict the target word from the context-words, they now try to predict each context-word from the target word. This turns out to be a very good model for feature learning.

Due to the simplicity of the models and efficient training algorithms, they are able to scale training to datasets of more than 1 billion words, with a vocabulary of nearly 700 thousand words, by training on a single desktop machine in only a few hours. They are able to show strong results on the *analogical reasoning dataset* [15], which measures several syntactic and semantic properties captured by the word embedding features. They find that the CBOW model trains faster, as expected, but the skip-gram model learns better features in general. We will offer some conjecture into why this might be the case in § 3.6.

3.4 Discussion of NLMs

In section we analyse the different neural word embedding models in terms of the three considerations introduced in § 3.1 and displayed in Fig. 3.2:

1. how the model defines context,
2. how the model predicts target words from context words, and
3. how the model is trained.

3.4.1 How context h is defined

For *language modelling*, where the goal is to predict the next word in a sequence of words, the notion of context is well-defined as the words *preceding* a given target word in the sentence (looking at future words to predict the next word would be considered cheating). However, for *feature learning* we are free to define context in any way we please, and indeed, different models have different definitions of context. *Window-based methods* define context as a window of words directly preceding [12, 13, 51, 55] or surrounding [8, 14] the target work. *Recurrent models* define context as all words in the sentence preceding the target word [62]. Some researchers have also started experimenting with *syntactically-derived contexts* [66].

In general, the definition of context plays a very important role in the *type* of information that is present in the training data for the models to compress into embeddings. Traditionally in distributional methods it is known that using a wider contextual window around target words causes the models to learn more semantic (topical) information, while using a smaller window tends to preserve more syntactical information. However, as shown by Levy et al. [66] one need not be restricted only to the words in a window around the target word, and indeed there may well be a better definition of context which may allow the models to

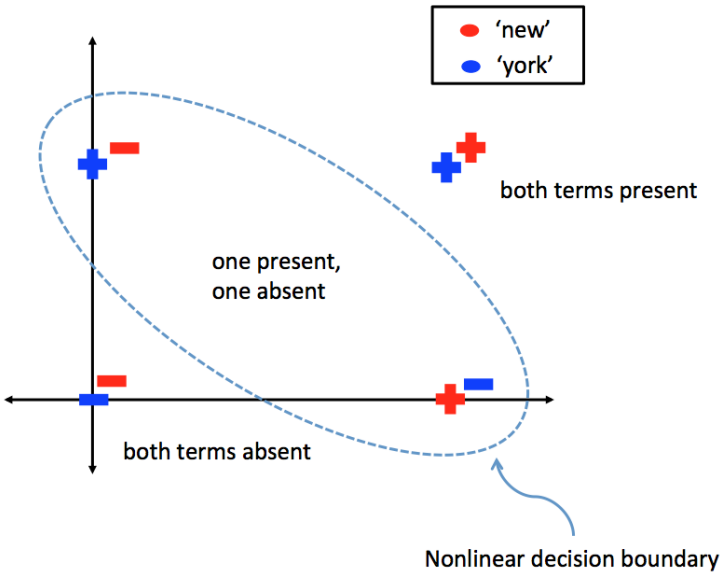


Figure 3.9: Nonlinear neural language models can model when two words are both absent (0,0), both present (1,1), or when one is present and the other is not (0,1) or (1,0). Here we illustrate the idea using the words *new* and *york* which have different meanings based on whether they appear alone or together.

learn features more relevant for the specific task at hand. Specifically, in this dissertation we will introduce the Barista model (Chapter 6) where we train models on *bilingual contexts* to learn cross-lingual features.

3.4.2 The structure of $f_\theta(h)$

The network architecture defines how the model maps the observed context words into a compressed representation $f_\theta(h)$ which it then uses to make a prediction about the target word. This function can be linear (CBOW, skipgram, LBL, HLBL), nonlinear (NPLM, RNN) or even probabilistic (using e.g. RBMs [67]).

The nature of this function has a strong bearing on the *capacity* of the model in terms of its ability to model language. For instance, a linear function can only detect linear relationships between its input words, whereas a nonlinear function can detect nonlinear relationships. Theoretically, this means that a nonlinear model can exploit XOR-like relationships such as 'new' and 'york' is present versus only 'new' is present or only 'york' is present, which a linear model cannot do, as illustrated in Fig. 3.9. However, as was shown for the log-bilinear model (§ 3.3.2), evidently this capability does not give the nonlinear NPLM much of an advantage over the linear LBL in terms of the model perplexity. There may be different reasons for this though (although this is pure conjecture): It might be related to training, since training the nonlinear NPLM is much harder than the linear LBL, and the

results for the NPLM precedes a lot of recent research advances in training deeper, nonlinear models. It might also simply be that in high-dimensional spaces the linear LBL model might still be able to find a suitable representation for representing the data which does not require a nonlinear function.

For feature learning, however, the most popular models are the CBOW and Skip-gram which simply consider the averaged bag of words representation for an input window, i.e. $f_\theta(h) = 1/N \sum_i r_{w_i}$ for the N context words $w_i \in h$.

3.4.3 The loss function $\mathcal{L}(w_t, h)$

So far we have looked at how the model selects context words from the training data and how the network maps context words into a vector representation $f_\theta(h)$. In this section we consider the final crucial component that makes up NLMs, namely how the model is trained. There are many different techniques for training NLMs, and a lot of research has focused on speeding up training by using more scalable training techniques, while retaining the quality of the learned representations.

The first neural probabilistic language model (§ 3.3.1) took 3 weeks to train on 12 million words with a vocabulary size of only 17 thousand words, and by parallelising training over 40 machines [12]. Contemporary neural word embedding models can be trained over 1 billion words (100 times larger) of Wikipedia text, using a vocabulary size of 1 million words (50 times larger) on a single desktop machine (roughly 40 times less computational power, if we ignore the effect of increasing computational capacity) in one or two hours (roughly $(3 \cdot 7 \cdot 24)/1h \approx 500$ time speedup, for a total of roughly $100 \cdot 50 \cdot 40 \cdot 500 = 10^8$ time speedup. Part of this considerable increase in speed is due to Moore's law and simply training on faster machines. However the largest factor is due to simpler architectures (without losing the quality of the induced embeddings) and improved training techniques, which we will examine in this section.

As with all neural networks, NLMs are trained by computing the training delta at the output layer, and then making use of the backpropagation algorithm (see § 2.3.5) to assign credit layer-by-layer backwards in the network to derive the gradients of the loss function with respect to the model parameters θ. Therefore, in the following sections we will show the mathematical derivations of the gradients of each loss function with respect to the output layer, required to initialise the backprop algorithm. Furthermore, since most of the training advances obtain their efficiency by only manipulating a subset of the output weights, we will also show how this would translate into code for computing the gradients with respect to the penultimate layer, which can then be backpropagated in deeper models using the standard backprop algorithm.

3.4.3.1 Maximum likelihood

The original formulation of the NPLM defined the output layer as computing the conditional probability of the target word given the context, $P(w_t|h)$, using the **softmax** function (i.e. as an *energy-based model*, [52]). The softmax computes the posterior probability of w_t given h, $P(w_t|h)$, using *scores* (inversely related to the energy function) computed in terms of the word embeddings:

$$P(w_t|h) = \frac{\exp(s_\theta(w_t, h))}{\sum_{w' \in V} \exp(s_\theta(w', h))} \tag{3.4.1}$$

where $s_\theta(w, h)$ is a score function which measures the "compatibility" between a word w in some context h, both represented as vectors in $\mathbb{R}^K$ for some feature dimension K (typically 50-1000). Scores are exponentiated to ensure that they are positive, and divided by the sum of (context-dependent) scores for the entire vocabulary to ensure that all probabilities sum to one.

In order to train this model, the standard approach is to maximise the log-likelihood of the data under this model by using a gradient-based optimisation method, of which stochastic gradient ascent is almost exclusively used[3]. Therefore, to train some parameter θ of this model, we require the gradients of the log-likelihood with respect to θ:

$$\begin{aligned} \frac{\partial}{\partial \theta} \log P(w_t|h) &= \frac{\partial}{\partial \theta} \left[\cancel{\log \exp}(s_\theta(w_t, h)) - \log \sum_{w' \in V} \exp(s_\theta(w', h)) \right] \\ &= \underbrace{\frac{\partial}{\partial \theta} s_\theta(w_t, h)}_{\text{target word}} - \underbrace{\frac{\partial}{\partial \theta} \log \sum_{w' \in V} \exp(s_\theta(w', h))}_{\text{all words}} \end{aligned} \tag{3.4.2}$$

Computing the gradients of the first term is generally straightforward when the score function is for instance a dot product. For the second, the log that inconveniently precedes the sum (known as a *log-sum-exp*) complicates the calculation of the derivative, but it can be approached using two inverse identities to rewrite the second term into a more convenient form:

[3] Alternatively, one may minimise the *negative log-likelihood* using stochastic gradient *descent*.

$$
\begin{aligned}
&\frac{\partial}{\partial\theta}\log\sum_{w'\in V}\exp(s_\theta(w',h)) \\
&= \frac{1}{\underbrace{\sum_{w'\in V}\exp(s_\theta(w',h))}_{\text{constant } Z_\theta(h)}}\sum_{w'\in V}\underbrace{\frac{\partial}{\partial\theta}\exp(s_\theta(w',h))}_{\nabla P} && \triangleright\ \nabla\log P = 1/P\nabla P \\
&= 1/Z_\theta(h)\sum_{w'\in V}\underbrace{\exp(s_\theta(w',h))}_{P}\underbrace{\frac{\partial}{\partial\theta}\cancel{\log\exp}(s_\theta(w_t,h))}_{\nabla\log P} && \triangleright\ \nabla P = P\nabla\log P \\
&= \sum_{w'\in V}\exp(s_\theta(w',h))/Z_\theta(h)\frac{\partial}{\partial\theta}s_\theta(w',h) \\
&= \sum_{w'\in V}P_\theta(w'|h)\frac{\partial}{\partial\theta}s_\theta(w',h) && (3.4.3)
\end{aligned}
$$

By substituting this into 3.4.2, we get the final expression for the maximum likelihood gradients as

$$
\frac{\partial}{\partial\theta}\log P(w_t|h) = \frac{\partial}{\partial\theta}s_\theta(w_t,h) - \sum_{w'\in V}P_\theta(w'|h)\frac{\partial}{\partial\theta}s_\theta(w',h) \qquad (3.4.4)
$$

Note that the contribution from the target word to the total gradient can be obtained by setting $w' = w_t$ to obtain $(1 - P_\theta(w_t|h))\frac{\partial}{\partial\theta}s_\theta(w_t,h)$. Likewise, the contribution from each non-target word only derives from the second term: $-P_\theta(w'|h)\frac{\partial}{\partial\theta}s_\theta(w',h)$. This is also simply the cross-entropy error with label 1 for the target word and 0 for every other word.

This can be implemented in Python pseudo-code as illustrated in Listing 3.

Listing 3 Maximum likelihood training using the softmax function.

```
def compute_ml_delta(wt, h):
  hidden, output = fprop(h)      # fprop(wt,h) == P(wt | h)
  err = 1. - output[wt]          # err >= 0
  Qgrad = hidden
  delta = err * Q[wt,:]          # compute delta before update
  Update(Q[wt,:], err * Qgrad)   # Q is 'output embeddings'
  for wneg in vocabulary:
    err = 0. - output[wneg]       # err <= 0
    delta += err * Q[wneg,:]
    Update(Q[wneg, :], err * Qgrad)
  return delta
```

3.4.3.2 Stochastic maximum likelihood

In the case of the full softmax, from 3.4.2 and 3.4.3, we can write the second term of the gradient of the log-likelihood as an expectation over the model distribution $P_\theta(w|h)$:

$$\frac{\partial}{\partial\theta}\log P_\theta(w_t|h) = \frac{\partial}{\partial\theta}s_\theta(w_t,h) - \sum_{w'\in V} P_\theta(w'|h)\frac{\partial}{\partial\theta}s_\theta(w',h) \tag{3.4.5}$$

$$= \frac{\partial}{\partial\theta}s_\theta(w_t,h) - \mathop{\mathbb{E}}_{w'\sim P_\theta(w'|h)}\left[\frac{\partial}{\partial\theta}s_\theta(w',h)\right] \tag{3.4.6}$$

In other words, if we can easily sample from the model distribution, then we can approximate this gradient using Monte Carlo integration (see § 2.1.3). Unfortunately, sampling from $P_\theta(w|h)$ requires one to compute the entire forward pass to compute all output activations, which requires $O(V)$ computations and therefore results in no savings at all over just doing regular maximum-likelihood learning. However, one may rewrite an expectation with respect to a distribution P in terms of an expectation with respect to Q,

$$\begin{aligned}\mathop{\mathbb{E}}_{x\sim P(x)}[f(x)] &= \sum_x P(x)f(x)\\ &= \sum_x Q(x)(P(x)/Q(x))f(x)\\ &= \mathop{\mathbb{E}}_{x\sim Q(x)}[\gamma(x)f(x)]\end{aligned} \tag{3.4.7}$$

where $\gamma(x) = P(x)/Q(x)$ are known as the 'importance weights'. This means that in order to approximate an average according to a distribution P, one can sample according to a distribution Q and reweight by P. This is useful in the case where it is hard to *sample* from P but where *evaluating* $P(x)$ is cheap. Unfortunately, that is also not the case, since evaluating P in our case is expensive. It turns out there is another trick one could employ in the case where computing the normalised $P(x) = \tilde{p}(x)/Z$ is hard, but computing the unnormalised $\tilde{p}(x)$ is easy:

$$\mathop{\mathbb{E}}_{x\sim P(x)}[f(x)] \approx 1/\Gamma \sum_{x_i\sim Q(x)}^{i=k} w(x_i)f(x_i) \tag{3.4.8}$$

where now, $\gamma(x) = \tilde{p}(x)/Q(x)$ and $\Gamma = \sum_{i=1}^{k}\gamma(x_i)$. This is a *biased estimator* of 3.4.7, but one can show that it converges to the true value as $k\to\infty$. By substituting this expression into 3.4.5, we can obtain a *Biased Importance Sampling Estimator* of the gradient of the log-likelihood [56, 68]:

$$\frac{\partial}{\partial\theta}\log P_\theta(w_t|h) = \frac{\partial}{\partial\theta}s_\theta(w_t,h) - \mathop{\mathbb{E}}_{w'\sim P_\theta(w'|h)}\frac{\partial}{\partial\theta}s_\theta(w',h) \tag{3.4.9}$$
$$\approx \frac{\partial}{\partial\theta}s_\theta(w_t,h) - 1/\Gamma\sum_{w_i\sim Q(w)}^{k}\gamma(w_i)\frac{\partial}{\partial\theta}s_\theta(w_i,h)$$

for some *proposal distribution* $Q(w)$. What this means in practice is that to obtain the gradient at time step t, given a context-target training pair, one first computes the (analytical) first term. One then samples k words from the proposal distribution Q, and computes the weights $\gamma(w_i) = s_\theta(w_i,h)/Q(w_i)$ to weight the contribution of the gradients computed over the sample, and finally sum up all the weights to normalise the weights to sum to 1. This would yield the gradient for computing one update to the model parameters, as shown by the Python pseudocode in Listing 4.

Listing 4 Stochastic maximum likelihood training using Biased Importance Sampling.

```
def compute_bis_delta(wt,h):
  hidden = get_hidden(h)  # compute f_θ(h)
  Z = score(wt, h)        # normalisation constant
  neg_words = []
  for i in xrange(k):
    wneg = sample_from_proposal()
    weight = score(wneg,h)   # unnormalised
    Z += weight
    neg_words.append((wneg, weight))
  err = 1.0 - score(wt, h)/Z
  Qgrad = hidden
  delta = err * Q[wt,:]
  Update(Q[wt,:], err * Qgrad)
  for (wneg, weight) in neg_words:
    err = 0. - weight / Z
    delta += err * Q[wneg,:]
    Update(Q[wneg,:], err * Qgrad)
  return delta
```

Ideally, the proposal distribution Q should minimize

$$Q^* = \arg\min_Q \mathop{\mathbb{E}}_{P(x)}[P(x)/Q(x)] = \int_X \frac{P(x)^2}{Q(x)}dx. \tag{3.4.10}$$

This would yield the minimum-variance estimator. However, this integral is typically hard to solve in practice and a good substitute is to use the empirical unigram distribution [56].

Unfortunately, this gradient estimator suffers from high variance during training [56]. Noise-contrastive estimation (discussed in § 3.4.3.3) starts out as a very different formulation, but ends up doing a very similar thing to BIS, without the variance issues.

3.4.3.3 Noise-contrastive estimation (NCE)

Noise-contrastive estimation (NCE) was originally proposed by Gutmann and Hyvärinen as an alternative to maximum-likelihood learning, and can be understood as "learning by comparison" [69,70]. Some of these ideas can also be related to work in the whole-sentence language modeling literature [71]. In NCE, the objective is to frame learning as a model's ability to discriminate (or compare or classify) between the actual data and some known noise distribution. The idea being, if a model can successfully tell if a sample comes from the data distribution or from some noise distribution it must necessarily have learned properties of the data in the form of a statistical model. If the noise distribution is sufficiently non-trivial, then the properties that the model must have learned about the data will become progressively more fine-grained.

Although NCE was originally introduced for learning marginal density models $p(x)$, it was subsequently successfully applied as a particularly efficient method for training conditional, discrete neural probabilistic language models that model $P(w|h)$ by Mnih [65]. The paper by Gutmann and Hyvärinen and the paper by Mnih and Teh use different notation, and it is not immediately obvious how their formulations are related, so in this section we will start from the intuition, which leads to the original formulation by Gutmann & Hyvärinnen [69], and then do the full derivation to arrive at the formulation used by Mnih [65]. We start by writing down the original noise-contrastive estimation objective, which posits that a good model should be able to classify data from noise, in other words we want to maximise

$$\mathcal{L}_{\text{NCE}}(\theta) = \mathop{\mathbb{E}}_{w^+,h\sim P_d}\left[\log(\sigma(\Delta s_\theta(w^+,h)))\right] + \mathop{\mathbb{E}}_{w^-\sim P_n}\log(1-\sigma(\Delta s_\theta(w^-,h))), \tag{3.4.11}$$

where we can see that we are doing logistic regression on a (delta) score computed from the observed (positive) words drawn from the data distribution $P_d(w)$, versus scores computed from noise (negative) words w^- coming from a noise distribution $P_n(w)$. The delta score function is defined as the log-odds of the sample under the model distribution and the noise distribution, i.e.

$$\begin{aligned}\Delta s_\theta(w,h) &= \log\frac{P_\theta(w|h)}{P_n(w)} && (3.4.12)\\ &= \log P_\theta(w|h) - \log P_n(w) && (3.4.13)\\ &= \log\tilde{p}_\theta(w|h) + c_h - \log P_n(w) && (3.4.14)\end{aligned}$$

where we have written the score in terms of the unnormalised distribution $\tilde{p}_\theta(w|h)$ and its context-dependent normalisation constant $c_h = \log Z_\theta(h)$. Note, however, that computing

c_h requires evaluating the full expensive output layer, so *one of the key tricks* of applying NCE to train NLMs, is that in [65] the authors set $Z_\theta(h) = 1$, i.e. $c_h = 0$, and report that it does not negatively affect the training in any noticeable way. We therefore follow suit and drop the constant in the following derivation.

One can now also see that an alternative interpretation to the objective is that we are trying to *maximise* the log-odds assigned by the model to data coming from the data distribution, and maximise inverse log-odds (therefore *minimise* log-odds) assigned by the model to data from the noise distribution.

At this point, by treating the two classifiers as independent logistic regression models, we have all that we need to compute the gradients of the cross-entropy with respect to the model parameters, and hence to train a model using NCE. This is indeed the simplest way to implement NCE in practice, and what we will use in Listing 5. However, to understand how this objective function is related to maximum likelihood, we need to derive the gradients. We will therefore do the full derivation to fill in the missing details presumably due to space constraints in their paper, to arrive at the NCE equations presented by Mnih and Teh [65].

We start by noting that given the sigmoid function $\sigma(z) = 1/(1 + e^{-z})$, we have that $1 - \sigma(z) = \sigma(-z)$, so we can therefore write the objective as

$$\mathcal{L}_{\text{NCE}}(\theta) = \mathop{\mathbb{E}}_{w^+,h\sim P_d}\left[\log(\sigma(\Delta s_\theta(w^+,h)))\right] + \mathop{\mathbb{E}}_{w^-\sim P_n}\log\sigma(-\Delta s_\theta(w^-,h))) \tag{3.4.15}$$

We assume, by design, that noise samples are k times more frequent than data samples since we will be sampling k times more noise than data. Therefore the delta scores are $\Delta s_\theta(w,h) = \log\frac{\tilde{p}_\theta(w|h)}{kP_n(w)}$ and we will also multiply the second expectation with k below. From this point on we will also write the *model distribution* $\tilde{p}_\theta(w|h)$ as $P^h_\theta(w)$ to save space. Using the definition of the sigmoid function shown above, we can rewrite the sigmoid-delta-scores for the positive term in 3.4.15 as

$$\sigma(\Delta s_\theta(w^+,h)) = \frac{1}{1+\exp^{-\Delta s_\theta(w^+,h)}} \tag{3.4.16}$$

$$= \frac{1}{1 + kP_n(w^+)/P^h_\theta(w^+)} \tag{3.4.17}$$

$$= \frac{P^h_\theta(w^+)}{P^h_\theta(w^+) + kP_n(w^+)} \tag{3.4.18}$$

and likewise for the negative term

$$\sigma(-\Delta s_\theta(w^-,h)) = \frac{kP_n(w^-)}{P^h_\theta(w^-) + kP_n(w^-)} \tag{3.4.19}$$

Now we can substitute this into 3.4.15, to obtain

$$\mathcal{L}_{\text{NCE}}(\theta) = \mathop{\mathbb{E}}_{w^+,h\sim P_d}\left[\log\left(\frac{P^h_\theta(w^+)}{P^h_\theta(w^+) + kP_n(w^+)}\right)\right] + k\mathop{\mathbb{E}}_{w^-\sim P_n}\left[\log\left(\frac{kP_n(w^-)}{P^h_\theta(w^-) + kP_n(w^-)}\right)\right]. \tag{3.4.20}$$

This is the objective as it is presented in Mnih [65]. We train the model by maximising the log-likelihoods under both distributions. To do this, we are interested in deriving the equations for the gradients with respect to a model parameter θ, therefore

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial \theta} = \frac{\partial}{\partial \theta} \mathop{\mathbb{E}}_{w^+, h \sim P_d} \left[\log P_\theta^h(w^+) - \log \left[P_\theta^h(w^+) + \underbrace{kP_n(w^+)}_{\text{constant}} \right] \right] + \frac{\partial}{\partial \theta} k \mathop{\mathbb{E}}_{w^- \sim P_n} \left[\underbrace{\log kP_n(w^-)}_{\text{constant}} - \log \left[P_\theta^h(w^-) + \underbrace{kP_n(w^-)}_{\text{constant}} \right] \right] \tag{3.4.21}$$

Due to the linearity of the expectation, we can take the derivatives inside the expectations. Next, we can use the fact that $\frac{\partial \log f(x)}{\partial x} = 1/f(x) \frac{\partial f(x)}{\partial x}$, and noting that the derivatives with respect to the constant noise probabilities go to zero:

$$= \mathop{\mathbb{E}}_{w^+, h \sim P_d} \left[\frac{\partial}{\partial \theta} \log P_\theta^h(w^+) - \frac{1}{P_\theta^h(w^+) + kP_n(w^+)} \left[\underbrace{\frac{\partial}{\partial \theta} P_\theta^h(w^+)}_{\nabla P} + 0 \right] \right] + k \mathop{\mathbb{E}}_{w^- \sim P_n} \left[0 - \frac{1}{P_\theta^h(w^-) + kP_n(w^-)} \left[\underbrace{\frac{\partial}{\partial \theta} P_\theta^h(w^-)}_{\nabla P} + 0 \right] \right] \tag{3.4.22}$$

We can then use the fact that $\nabla P = P \nabla \log P$ to rewrite this as

$$= \mathop{\mathbb{E}}_{w^+, h \sim P_d} \left[\frac{\partial}{\partial \theta} \log P_\theta^h(w^+) - \frac{P_\theta^h(w^+)}{P_\theta^h(w^+) + kP_n(w^+)} \frac{\partial}{\partial \theta} \log P_\theta^h(w^+) \right] - k \mathop{\mathbb{E}}_{w^- \sim P_n} \left[\frac{P_\theta^h(w^-)}{P_\theta^h(w^-) + kP_n(w^-)} \left[\frac{\partial}{\partial \theta} \log P_\theta^h(w^-) \right] \right] \tag{3.4.23}$$

By collecting the common $\frac{\partial}{\partial \theta} \log P_\theta^h(w^+)$ terms in the first expectation

$$= \mathop{\mathbb{E}}_{w^+, h \sim P_d} \left[\left(1 - \frac{P_\theta^h(w^+)}{P_\theta^h(w^+) + kP_n(w^+)} \right) \frac{\partial}{\partial \theta} \log P_\theta^h(w^+) \right] - k \mathop{\mathbb{E}}_{w^- \sim P_n} \left[\frac{P_\theta^h(w^-)}{P_\theta^h(w^-) + kP_n(w^-)} \left[\frac{\partial}{\partial \theta} \log P_\theta^h(w^-) \right] \right] \tag{3.4.24}$$

and then rewriting $1 - \frac{P}{P+N}$ as $\frac{N}{P+N}$ we arrive at the final formulation

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial \theta} = \mathop{\mathbb{E}}_{w^+,h\sim P_d}\left[\underbrace{\frac{kP_n(w^+)}{P_\theta^h(w^+)+kP_n(w^+)}}_{\alpha(w^+)}\frac{\partial}{\partial\theta}\log P_\theta^h(w^+)\right] - k\mathop{\mathbb{E}}_{w^-\sim P_n}\left[\underbrace{\frac{P_\theta^h(w^-)}{P_\theta^h(w^-)+kP_n(w^-)}}_{\beta(w^-)}\left[\frac{\partial}{\partial\theta}\log P_\theta^h(w^-)\right]\right] \tag{3.4.25}$$

This is the expression for the gradient as it is presented by Mnih [65]. To simplify the expression, we can write this more compactly by defining the weights on the positive and negative gradients to be α and β respectively

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial \theta} = \mathop{\mathbb{E}}_{w^+,h\sim P_d}\left[\alpha(w^+)\frac{\partial}{\partial\theta}\log P_\theta^h(w^+)\right] - k\mathop{\mathbb{E}}_{w^-,h\sim P_n}\left[\beta(w^-)\frac{\partial}{\partial\theta}\log P_\theta^h(w^-)\right] \tag{3.4.26}$$

In practice, at each training step, given an observed w^+, h, we sample k negative words w^- from the noise distribution, and then compute the gradient as

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial \theta} = \alpha(w^+)\frac{\partial}{\partial\theta}\log P_\theta^h(w^+) - k\sum_{w^-\sim P_n}\beta(w^-)\frac{\partial}{\partial\theta}\log P_\theta^h(w^-). \tag{3.4.27}$$

The speedup over maximum-likelihood results since we only need to compute these terms for the $(k+1)$ words instead of for all V words. Furthermore, by explicitly writing the expectations in 3.4.25 as weighted sums and collecting like terms, we can also write this expression as follows:

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial \theta} = \sum_w \alpha(w)\Big(P_d(w) - P_\theta^h(w)\Big)\frac{\partial}{\partial\theta}\log P_\theta^h(w), \tag{3.4.28}$$

which, as $k \to \infty$ and hence $\alpha(w) \to 1$, *becomes the maximum-likelihood gradient* (3.4.4) since the second term drops away:

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial \theta} \to \sum_w \Big(P_d(w) - P_\theta^h(w)\Big)\frac{\partial}{\partial\theta}\log P_\theta^h(w) \tag{3.4.29}$$

$$= \underbrace{\mathop{\mathbb{E}}_{P_d}\left[\frac{\partial}{\partial\theta}\log P_\theta(w)\right]}_{\text{ML gradient}} - \underbrace{\mathop{\mathbb{E}}_{P_\theta^h}\left[\frac{\partial}{\partial\theta}\log P_\theta^h(w)\right]}_{0} \tag{3.4.30}$$

We therefore arrive at the result that a completely different objective (classifying data from noise), which turns out to be much easier to optimise (as we will discuss below), leads

to performing the same updates on the model parameters (in the limit) as the original maximum-likelihood objective.

However, to implement NCE, we find that it is simpler to understand it simply as doing logistic regression on the delta-scores as defined by 3.4.11. This can be implemented as the Python pseudo-code illustrated in Listing 5. Note that in this formulation, `err` is equal to α for positive words, and β for negative words.

Listing 5 Training with noise-contrastive estimation, implemented as doing logistic regression on the delta scores $\log(P_{\text{model}}/P_{\text{noise}})$.

```
def compute_nce_delta(wt,h):
  hidden = get_hidden(h)
  Qgrad = hidden
  delta = zeros(hidden_dim)     # zero vector
  for i in xrange(k + 1):
    if i == 0:
      label = 1                 # target word
      w = wt
    else:
      label = 0                 # noise word
      w = sample(Pnoise_pdf)
    pn_w = Pnoise_pdf[w]        # numerical probability
    predict = score(w, hidden, pn_w)  # compute sigm(log P_m(w) - log P_n(w))
    err = (label - predict)     # cross-entropy error
    delta += err * Q[w,:]       # backprop
    Update(Q[w,:], err * Qgrad)     # classifier weights
  return delta
```

Due to the similarity of the expression for the NCE gradients to 3.4.9, one might wonder about the relative merits of each method. In practice, the $w(w_i)$ weights in the importance sampling objective can have a high variance, and this is related to how "different" the model distribution P and the proposal distribution Q are. When we sample a value with a small probability under Q but a large probability under P, the weight $w = P(w)/Q(w)$ can become very large. In the opposite case it can become very small. This high variance makes learning very unstable using BIS (since it can propose big, noisy adjustments to the parameters). One way to overcome this is to sample more and more words, but this makes the method more expensive. On the other hand, notice that the NCE weights α and β as we defined them, are bounded between 0 and 1. This tends to make learning much more stable, and indeed enables training to progress (albeit slower) even when using a sample size of $k = 1$.

3.4.3.4 Noise-contrastive rank-loss

In the non-probabilistic ranking language model proposed by Collobert & Weston [13], the objective is to use a single hidden-layer neural network $s_\theta(x)$ to score valid, observed sequences of words x, higher than a corrupted sequence $\tilde{x}$ with some margin m, i.e. the goal is to minimise

$$\mathcal{L}_{\text{rank}} = \sum_{x \in \mathcal{D}} \sum_{w' \in \mathcal{V}} \max\{0, m - s_\theta(x) + s_\theta^{w'}(\tilde{x})\} \tag{3.4.31}$$

Here we denote by $s_\theta^{w'}(\tilde{x})$ the real-valued score output by the neural network for the corrupt sequence formed from x by replacing a fixed word w (e.g. middle or last) in the sequence by word w' (with $w' \sim P_n(w)$ and $w' \neq w$).

This is not very efficient for the same reasons that the vanilla softmax is not efficient, since this objective considers all possible words at each training step. The authors therefore proposed a *stochastic rank loss* where at each update they sample one negative word $w^- \sim P_n(w)$ with $P_n(w)$ the unigram distribution, and minimise

$$\mathcal{L}_{CW} = \mathop{\mathbb{E}}_{w^- \sim P_n(w)} \left[\sum_{x \in \mathcal{D}} \max\{0, m - s_\theta(x) + s_\theta^{w^-}(\tilde{x})\} \right] \tag{3.4.32}$$

This formulation scales extremely well, and indeed allowed the authors to train one of the largest scale neural language models (at the time), over all the text of Wikipedia (600 million words at the time) on standard commodity hardware.

To understand how this objective function works, consider the updates it proposes during training by deriving the gradients of the loss with respect to the model parameters θ. Notice that 3.4.32 is just the standard hinge loss (§ 2.3.2.2):

$$g(x) = \max\{0, m - f(x)\}, \tag{3.4.33}$$

with $f(x) = s_\theta(x) - s_\theta^{w^-}(\tilde{x})$, i.e. $\mathcal{L}_{CW} = g(f(x))$. We can therefore use the chain rule of differentiation to derive

$$\frac{\partial}{\partial \theta} \mathcal{L}_{CW} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial \theta} \tag{3.4.34}$$

$$= \begin{cases} \frac{\partial}{\partial \theta}(m - f(x)) & \text{if } (m - f(x)) > 0, \\ 0 & \text{otherwise} \end{cases} \tag{3.4.35}$$

$$= \begin{cases} \frac{\partial}{\partial \theta} s_\theta^{w^-}(\tilde{x}) - \frac{\partial}{\partial \theta} s_\theta(x) & \text{if } s_\theta(x) - s_\theta^{w^-}(\tilde{x}) < m, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4.36}$$

Therefore we see that training proceeds (i.e. gradients are non-zero) while the scoring network scores the valid windows, containing the observed target words, less than a margin of m higher than windows containing the corrupt targets.

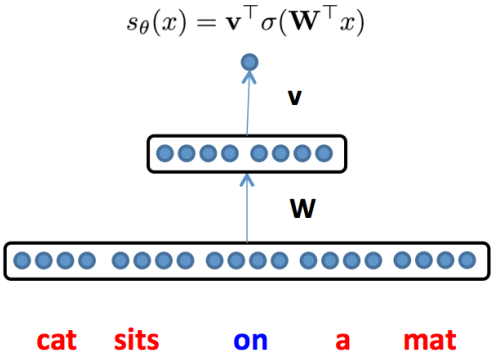


Figure 3.10: An illustration of the scoring network used by Collobert & Weston [8] to score observed windows of text against corrupt windows of text (by replacing the blue target word with a sampled noise word).

3.4.3.5 Noise-contrastive cross-entropy ("Negative sampling")

Mnih and Teh [65] introduced noise-contrastive estimation (NCE) as an efficient way of training NLMs (see § 3.4.3.3). NCE can be shown to approximately maximise the log-probability of the softmax (as we have shown by showing that the NCE gradient tends to the softmax gradient as the number of noise samples k is increased). While this is a useful property for training *language models*, this is not a requirement if the goal is simply to learn high-quality word embedding features. Mikolov [14] introduced a simplified version of NCE called **negative sampling** which can be considered a probabilistic version of the noise-contrastive rank loss discussed in § 3.4.3.4.

In negative sampling, the goal is for the model to assign a higher probability to observed samples of (w_t, h) training pairs, compared to draws from a noise distribution $P_n(w)$, i.e. we want to maximise $P_{\text{model}}(\text{data}) + (1 - P_{\text{model}}(\text{noise}))$. Note that both distributions are the same (model) distribution. If a model succeeds in maximising this objective, it means that it is simultaneously assigning high probability mass to the observed data and low mass to the noise data, which means that it must necessarily have learned interesting aspects about the data distribution.

Specifically, the model distributions are implemented as logistic regression classifiers, and the goal is to maximise

$$\mathcal{L}_{\text{NEG}}(\theta) = \mathop{\mathbb{E}}_{w_t,h\sim P_{\text{data}}} \left[\log \sigma(\Delta s_\theta(w_t,h))\right] + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w^-\sim P_n} \left[1 - \sigma(\Delta s_\theta(w^-,h))\right] \tag{3.4.37}$$

$$= \mathop{\mathbb{E}}_{w_t,h\sim P_{\text{data}}} \left[\log \sigma(\Delta s_\theta(w_t,h))\right] + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w^-\sim P_n} \left[\sigma(-\Delta s_\theta(w^-,h))\right] \tag{3.4.38}$$

$$= \mathop{\mathbb{E}}_{w_t,h\sim P_{\text{data}}} \left[\log \sigma(q(w_t)^\top f_\theta(h)))\right] + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{w^-\sim P_n} \left[\sigma(-q(w^-)^\top f_\theta(h)))\right]] \tag{3.4.39}$$

where we are writing the objective in terms of the $\Delta s_\theta(w_t,h)$ score to show the similarity with NCE, although now it is defined as the dot product of the output embeddings $\mathbf{Q}$ with the context vector $f_\theta(h)$.

This objective is clearly very related to the NCE objective (3.4.11), with the only difference in how the scoring function $\Delta s_\theta(w,h)$ is defined. In NCE, we make use of the *numerical* noise-probabilities, whereas this is dropped in negative sampling.

3.5 A Geometrical Perspective on Word Embedding Models

In this dissertation we are primarily interested in NLMs as unsupervised distributional models for learning features about words. In order to understand how different NLMs relate to each other in terms of the features that they learn about words, it is useful to consider how they are trained. In this section, we will highlight the commonalities between the different popular training techniques by geometrically considering their training dynamics. This analysis applies directly to the most popular log-linear bag-of-words `word2vec`-type models that we are most interested in for their feature-learning capabilities, but also acts as useful geometrical intuition about the training dynamics of nonlinear NLMs.

We start by noting that word embedding vectors trace out a geometrical surface (informally, a "manifold") in the learned embedding vector-space. For all popular loss functions that we have discussed in this chapter and that are trained using SGD on a per-example basis, one may interpret the loss function as assigning a cost to the current geometrical structure of the manifold with respect to a new observed training pair.

Furthermore, all loss functions that we have introduced contrast the score of the observed word to scores that the model assigns to a selection of unobserved (contrastive) words. If we let positive words $w^+ \sim \mathcal{P}$ correspond to the distribution of observed target words, and negative words $w^- \sim \mathcal{N}$ to the distribution of contrastive words that were *not* observed, then we can write the gradient of the loss at the output (the output delta) as the contributions resulting from the positive and negative words as

$$\delta^{\text{out}} = \mathop{\mathbb{E}}_{w^+\sim\mathcal{P}} \left[\delta^{\text{out}}(w^+)\right] + \mathop{\mathbb{E}}_{w^-\sim\mathcal{N}} \left[\delta^{\text{out}}(w^-)\right], \tag{3.5.1}$$

where we represent with $\delta^{\text{out}}(w^{+/-})$ the (scalar) contribution resulting from word $w^{+/-}$ to the derivative of the loss with respect to the output[4]. The different training methods that we looked at particularly differ in how they compute the output deltas, and how they select the negative words $\mathcal{N}$. However, for a simple BOW `word2vec`-type model, the gradients on the learned word representations $r_{w_h} = \mathbf{R}_{[w_h,:]}, \forall w_h \in h$ can be decomposed (after application of the delta-rule) as

$$\nabla_{r_{w_h}} \mathcal{L}(w_t, h) = \mathop{\mathbb{E}}_{w^+ \sim \mathcal{P}} \left[\underbrace{\delta^{\text{out}}(w^+)}_{>0} \mathbf{q}_{w^+} \right] + \mathop{\mathbb{E}}_{w^- \sim \mathcal{N}} \left[\underbrace{\delta^{\text{out}}(w^-)}_{<0} \mathbf{q}_{w^-} \right]. \tag{3.5.2}$$

Notice that these two terms represent directions along which we will update the input word representations r_h that we are trying to learn. Since the delta weights at the outputs are positive scalar values for positive words and negative scalar values for negative words, this implies that context words are *pulled* towards the positive words, and *pushed* away from the negative words.

Geometrically, this interaction can be visualised as points in a vector-space connected by pulling and pushing forces that act on the word vectors as shown in Fig. 3.11. The goal of the optimisation procedure during training is to move the word embedding point-vectors around in the negative direction of the gradient of the loss in order to find an equilibrium geometrical configuration (the embedding manifold) that minimises the expected cost of the loss function under the training distribution $\mathbb{E}_{P(w_t,h)}[\mathcal{L}(w_t, h)]$. In the process, the gradient tends to zero (i.e. training converges) as the representations for the target-words $w^+ \sim \mathcal{P}$ are balanced with and become more similar to the representations of their observed context-words $w_h \in h$, and vice versa, and the negative words $w^- \sim \mathcal{N}$ keep this process from collapsing to trivial solutions (such as assigning the same vector to all words). Therefore, during training, the initial spherical manifold spanned by the word embeddings[5], as illustrated in Fig. 3.11, will be contorted according to the frequency of observing the context words w_h with the positive and negative words, resulting in a rich geometrical encoding of the distributional statistics of the training data.

For nonlinear models, such as the NPLM, this geometrical interpretation should be viewed more as an intuition, since due to the nonlinearities the push-pull forces do not correspond to Euclidian distances anymore, but act along curves in a nonlinearly contorted space.

[4] For all loss functions we looked at, the functional form of the deltas on positive and negative words are the same but of opposite signs. For instance, one can show that the log-likelihood of the softmax $P_\theta(w_t|h) = \exp(s_\theta(w_t, h)) / \sum_j \exp(s_\theta(w_j, h))$ results after a bit of algebra in $\delta^{\text{out}}(w^{+/-}) = \pm \frac{\partial}{\partial \theta} s_\theta(w^{+/-}, h)$, etc.

[5] When we initialise word embeddings by sampling from an isotropical Gaussian distribution.

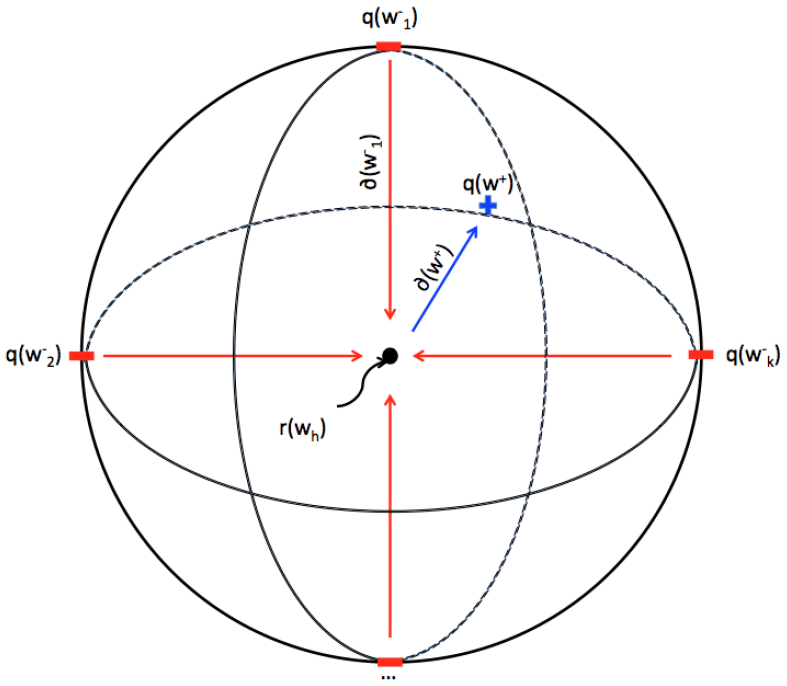


Figure 3.11: The gradients on the word embeddings when training shallow word embedding models can be interpreted as pushing and pulling forces acting on the input word embedding point-vectors r_h and on the output word embeddings q_w (shown here as $r(h)/q(w)$ for readability). The average force magnitudes $\delta(w)$ over a training set are proportional to the frequency of observing the word-pairs together during training. The goal of training is to balance these forces, in order to reach an equilibrium geometrical configuration, representing a rich geometrical encoding of the distributional statistics in the training data.

3.6 Why Do Similar Words Have Similar Embeddings?

We are now in a position to better understand the training dynamics of neural language models, and particularly how these dynamics produce informative embedding features for words. To start, consider the training data displayed in Fig. 3.12 showing hypothetical sentences from a training set that share the same context words. This data illustrates the Distributional Hypothesis, which states that words that are related occur in similar contexts (see § 3.2).

For the popular training objectives, we have seen that the gradients of the objective functions can be interpreted as exerting push-pull forces on the word embedding point-vectors. Conversely, all popular objective functions can be shown to be proportional to the distance between the target word embeddings and the context-vector $f_\theta(h)$ computed by the network, i.e. $\mathcal{L}(w_t, h) \propto \text{dist}(r_{w_t}, f_\theta(h))$. In order to minimize this function over the

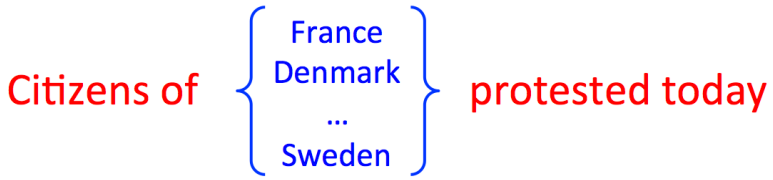


Figure 3.12: An example of several training fragments from a training corpus that share the same context words (in red) with different target words (in blue curves).

training data, we therefore require that

$$\begin{aligned}\mathcal{L} &\propto \sum_{(w_t,h)\in\mathcal{D}} \text{dist}(r_{w_t}, f_\theta(h)) \\ &\propto \sum_h \sum_{w_t|h} \text{dist}(r_{w_t}, f_\theta(h))\end{aligned}$$

is minimised, where we have simply rewritten the sum over the training data to be the sum over the data, ordered by context. Therefore, *for the same context* h, we want to minimize the distance of all word vectors r_{w_t} that frequently co-occur with that context, to the same fixed point $\mathbb{E}_{w_t,h\sim P_d}[f_\theta(h)]$. By the distributional hypothesis, these words are all related, hence related words cluster together.

An alternative interpretation is that the dynamics of the learning process force distributionally similar words into the same regions of the embedding space, so that the *same* mapping $f_\theta(h)$ maps a fixed context h close to or assigns a high score to all target words w_t observed in the training distribution. I.e. in the above example, in order for the network to assign a high score to *France, Denmark, ...* given the same fixed context, it necessarily implies that the learned vector representations for q_{France} and q_{Denmark} etc. need to be close. Through this process of nudging word embeddings a little bit towards each word's embedding that it co-occurs with, the vectors end up representing a rich amount of distributional information *geometrically* in terms of the distances between the word embeddings in the embedded space.

This might also explain why the (inverted) skipgram architecture is empirically found to learn better features than other models *when we train over more data* [58]. In context-to-target models (e.g. CBOW), the updates for each context word is obtained by *smoothing over* all context words (by computing the delta as the difference between the target word and the smoothed context vector). This discards a lot of distributional information. However, in the skip-gram, each target word is explicitly updated towards each context-word vector.

The smoothing might help when we are training with small datasets, but it also reduces the information in the training data by acting as a regulariser. However, as the amount of data increases there is an implicit smoothing effect, and skipgram-type models can evidently

benefit from the richer information that exists by considering each (w_t, w_h) training-pair individually (where w_h is a context-word).

3.7 Conclusion

This chapter provided an in-depth overview of all popular neural network language models and the methods for training them. NLMs were shown to compress high-dimensional co-occurrence observations of words and the contexts they appear in, into a low-dimensional space where distance between the compressed representations (embeddings) is proportional to the frequency of observing these input and output units together in the training data. How context-words are selected has a big impact on the *types* of co-occurence relationships that the model encodes. The loss function has a big impact on the speed with which a model can be trained. This chapter provided a detailed look at these different components.

Furthermore, the training dynamics of word embedding models were discussed in terms of a novel geometrical framework which enables one to understand these models as learning useful features about words by balancing pushing and pulling forces acting on the word point-vectors during training, in order to match the observed distributional statistics of natural language in the training data.

Chapter 4

Improved Learning with Hierarchical Output Layers

In the standard neural language model (NLM), the goal is to predict the next word w given its context h according to the softmax (Gibbs) distribution (see § 3.3.1)

$$P(w|h) = \exp(s_\theta(w,h))/Z_\theta(h). \tag{4.0.1}$$

Training *and* testing scales badly due to the standard output softmax layer requiring a context-dependent normalisation constant $Z_\theta(h) = \sum_{w' \in V} \exp s_\theta(w', h)$, sometimes called the *partition function*, which depends on the current model parameters $\theta^{(t)}$ (and hence cannot simply be computed once during training and stored), and therefore needs to sum over the entire vocabulary during each training step and each testing step.

This chapter provides an in-depth discussion of a more scalable version of the softmax, called the **hierarchical softmax** [53]. Based on the theoretical insight gained, we propose two methods for improving training of NLMs using the hierarchical softmax.

The hierarchical softmax borrows an idea from class-based language models [42] (§ 2.4.2.1), and partitions words into classes. Instead of summing over the entire vocababulary at each training step, we can then rewrite $P(w|h)$ as a sum over the much smaller individual classes. Here an important consideration is how to partition the vocabulary into a class structure, and for that the main contribution of the first part of this chapter is that we propose and evaluate an efficient $O(N \log N)$ algorithm called SimTree (§ 4.2), which constucts a tree structure in order to *maximise the expected model likelihood* (to a local maximum). The algorithm is fast to run in practice and our evaluations show that the model learns useful semantic clusterings, and that the tree structure leads to a large improvement in the perplexity that the model achieves in a language modelling task.

In the second part of this chapter, we hypothesise that training gradients in a tree-structured output layer get progressively more noisy towards the root. Based on this hypothesis we propose a simple technique for improved feature-learning called **subsampled**

hierarchical softmax (§ 4.3). The idea is to discard nodes in the tree during training with a probability inversely proportional to their depth in the tree (i.e. deeper nodes are more likely to be retained). Through experimental analysis we verify the hypothesis and show that this technique both speeds up training and leads to higher quality word embeddings.

The work in the first part of this chapter has been submitted to the *IEEE Transactions on Neural Networks and Learning Systems*.

4.1 Hierarchical Softmax (HS)

The hierarchical version of the softmax was introduced by Morin and Bengio [53]. In that work, the authors proposed to structure the tree according to the WordNet lexical resource, however it was not found to work very well. Mnih and Hinton [55] evaluated several data-driven clustering techniques for learning the tree structure, and showed that the structure of the tree has a big impact on the model quality. However, their approach involved disjointly pretraining word embeddings, running a clustering algorithm on the pretrained embeddings, and then retraining the model. In follow-up work Mnih and Teh [72] learn a tree level-by-level using a heuristic lookahead, however they still use a three-step bootstrapping process as opposed to fully online as in our case. Several other non-neural network based studies have also looked at using tree-based classifiers with perhaps the most similar to our work being the recently proposed LOM-tree method [73]. This was developed roughly in parallel to our work, but appears to be a very competitive alternative to our method. In future work we will compare the two methods head-to-head.

In this section, we introduce an online algorithm for jointly training the model parameters and learning the tree structure. We first present a detailed novel theoretical analysis of the most popular type of HS which uses a binary tree. Based on the analysis, we develop a clustering criterion for optimising tree structure for model likelihood, and integrate this within the **SimTree** algorithm. Finally, we present qualitatitve and quantitative evaluations of the proposed algorithm.

4.1.1 Coarse-to-fine elimination of words for speed savings

Due to the sum in the partition function, the softmax (4.0.1) can be seen as computing the probability of a target word by comparing it to (normalising by) the probability mass assigned to all potential alternatives. The softmax function over V words, therefore, *indiscriminately* considers all V words as possible next words when evaluating $P(w|h)$. Yet, language has sequential structure (which is exactly what we are trying to model), and certain words are certainly more likely in certain contexts, while others are less likely or even almost certainly improbable. One could obtain significant computational savings by only considering groups of the *most likely* words in certain contexts.

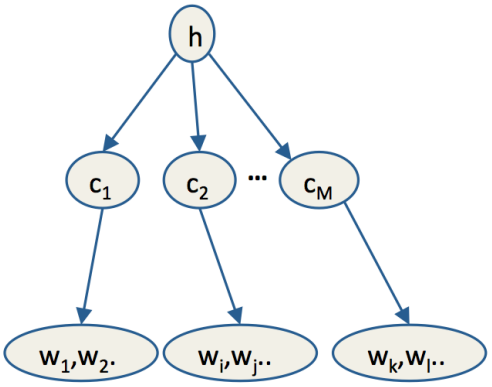


Figure 4.1: Graphical model illustrating the probabilistic dependencies between context h, class c_i and target word w_i variables when we represent $P(w_t|h)$ as a marginalisation over latent class variables c_i in a class-based language model.

More generally, in order to compute the probability $P(w|h)$, in the general case one needs to normalise over all $w \in V$ alternatives. However, one can make use of the general sum rule of probability (§ 2.1.1) to write

$$P(w|h) = \sum_i P(w, c_i|h) = \sum_i P(w|c_i, h)P(c_i|h). \tag{4.1.1}$$

In other words, we write the conditional distribution in terms of a marginalisation over a latent (unobserved) class variable c_i which depends on the context h.

If $P(w|c_i, h) > 0$ for all i (i.e. each word can potentially belong to all classes), then we do not save any computations (since the marginalisation still sums over all V words, albeit one class at a time). However, if we know or decide that words can belong to only K classes, and if the M *total* classes are roughly balanced with V/M words per class, then we can disregard all classes c_j for which $P(c_j|h) = 0$, which reduces the calculations to $O(KM)$ for roughly $O(VK/M)$ *speedup* over computing the full softmax. The "sweet spot" occurs when words belong only to $K = 1$ class and total classes $M = \sqrt{V}$ which yields $O(\sqrt{V})$ speedup. Visually, this can be interpreted as structuring the vocabulary into a two-layer tree, and calculating $P(w|h)$ by traversing a path from the root to class node $c_i \in c(h)$, summing (normalising) over only the subset of words in each class c_i (see Fig. 4.1). By furthermore grouping classes into meta-classes and introducing depth (visually one could think of this in the form of a multi-layer tree) one could push down the computational cost to $O(\log_b V)$ for b classes at each level of a balanced binary tree (see Fig. 4.2).

The hierarchical softmax exploits this property by *breaking up the calculation of the full softmax objective $P(w|h)$ into a tree structure to speed up its evaluation*. Additionally, by structuring the calculations into a tree structure we can also aim to make the prediction problem somewhat easier by ordering the computations along a path that progressively con-

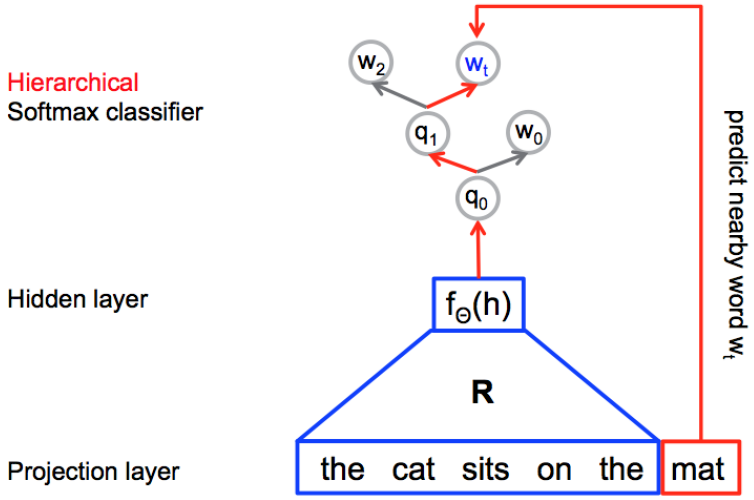


Figure 4.2: Illustration of the hierarchical softmax output layer. Note that the layer can be visualised as an inverted tree structure rooted at the penultimate layer, where the goal is to predict the intermediate left or right branching steps on the path from the root to a target word w_t.

siders more (semantically) fine-grained groups of words in a coarse-to-fine (easy to hard) hierarchy of predictions.

In this work, we consider the most popular case, where the tree is binary, i.e. $b = 2$, with $V - 1$ nodes (classes) defined over V words. For a balanced binary tree, this implies that at each branching step down the tree we are eliminating on average half of the (remaining) words that the model needs to consider, which yields a computational time logarithmic in the number of words in the vocabulary on average, or an *exponential speed-up* compared to evaluating the full softmax objective. For example, a vocabulary of $V = 10^4$ words can be processed using only 14 evaluations, $V = 10^5$ requires 17 and for $V = 10^6$ only 20.

4.1.2 The HS objective

The hierarchical softmax defines the language modelling objective

$$P(\text{next word } = w | \text{ previous words } = h)$$

in terms of a sequence of predictions over groups of words. Each prediction considers a smaller set of words, and models the probability that w is contained in that class of words. As was mentioned in the previous section, it is useful to think of this process as traversing a tree, where the root node considers all words, and each child node considers some finer-grained partitioning of the vocabulary.

In the case of a binary tree, each node can be seen as a binary classifier that predicts whether the target word is contained in its left (label=0) or right (label=1) subtree.

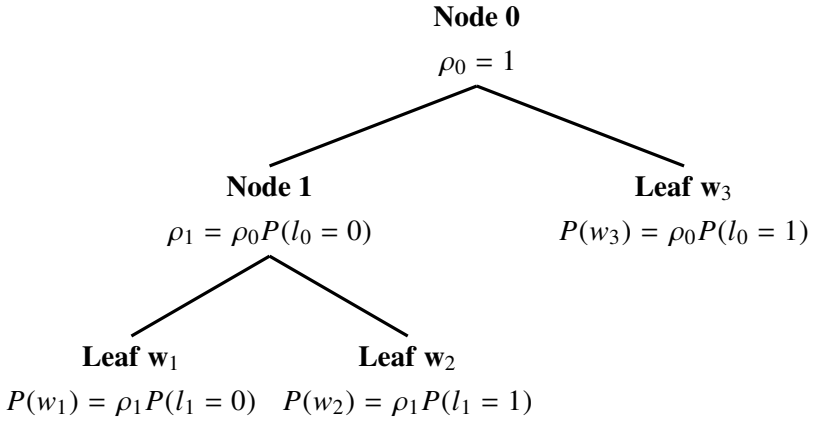
Each word w (leaf node in the tree) can therefore be represented by a unique bit vector $L(w) = \{l_1, l_2, \cdots, l_{|L(w)|}\}$ representing the concatenated sequence of left-right (0/1) branching decisions required to reach w from the root node.

Given these definitions, we define the binary **Hierarchical Softmax objective** as follows

$$P(w|h) = \prod_{i=1}^{|L(w)|-1} P(l_i|h). \tag{4.1.2}$$

where $i = 1$ is the root node. In this formulation, $P(l_i|h)$ represents a branching decision made at the i-th level of the tree about whether w is contained in the left ($P(l_i = 0|h)$) or right ($P(l_i = 1|h)$) subtree. This choice is arbitrary, as exchanging labels merely implies mirroring the tree structure.

Since $P(l_i = 0|h) + P(l_i = 1|h) = 1$, it is easy to show that $P(w|h)$ defines a properly normalised probability distribution over all words using the following informal argument by structural induction over the tree, and referring to the example tree below:



Consider the base case where we have a tree with only one root node and two leaf nodes. Since branching decisions at each node must sum to one, it is easy to see that the probabilities at the leaf nodes sum to one (by definition). By applying this logic recursively, dividing at each node i in the tree the ***remaining probability mass*** ρ_i received from its parent node j according to the branching probabilities $P(l_j)$, it is relatively straightforward to show that indeed the sum of all the residual leaf node probabilities must sum to one, i.e. $\sum_{w_i \in V} p(w_i|h) = 1$ and hence 4.1.2 defines a properly normalised probability distribution over all words.

4.1.3 Training the hierarchical softmax

For the binary HS, the most common choice is to parametrise the *right-branching probability* as:

$$P(l_i = 1|h) = \sigma(q_i^\top r_h + b_i) \tag{4.1.3}$$

where $\sigma(z) = 1/(1+e^{-z})$ is the logistic function and $r_h = f_\theta(h)$ is the network representation for the context words h. This can be interpreted as performing logistic regression on the left/right-branching label, using a learned feature vector q_i[1] and bias weight b_i for an internal node i of the tree. Note that the left-branching probability $P(l_i = 0|h) = 1-P(l_i = 1|h)$. Since $P(l_i = 0|h)+P(l_i = 1|h) = 1$, we can use the fact that $1-\sigma(z) = 1-1/(1+\exp(-z)) = \sigma(-z)$ to rewrite both probabilities compactly as

$$P(l_i|h) = \sigma(\pm[q_i^\top r_h + b_i]) \tag{4.1.4}$$

where we write a positive sign for right-branching labels ($P(l_i = 1|h)$), and a minus sign for left-branching labels.

Node embeddings q_i and **word embeddings** r_j are then jointly updated during training to maximise total model likelihood over a training set (given a tree structure). However, it is important to keep in mind that the tree structure and model parameters influence each other in terms of the model likelihood we are trying to maximise (and hence the probabilities we are trying to compute). We will consider this interaction in more detail in the following sections, but for now we will derive the training update equations assuming a given tree structure defined by $L(w)$.

As with all neural networks, training consists of two phases (see Fig. 4.3): Given a word-context training pair (w_t, h) ,

- all activations are forward-propagated through the network and through the relevant nodes on the path $L(w_t)$ to the observed target word w_t (**fprop**), then
- the error gradients are computed at the output and the deltas are back-propagated along the same path, updating parameters as we go (**bprop**)).

We treat each node on the path as a logistic regression classifier, and compute its cross-entropy error with respect to the (given) correct path $L(w_t)$ (i.e. predictions on a path reduce to a sequence of independent binary classifications).

Specifically, we train the model by maximising the log-likelihood $\mathcal{L}$ (not to be confused with the path $L(w)$) on a dataset using stochastic gradient ascent, where at training step t,

[1] Due to the dot product, q_i is necessarily of the same dimensionality as the word embeddings, r_j. Although it is possible to define an arbitrary scoring function other than the dot product, this is the only one used in the literature.

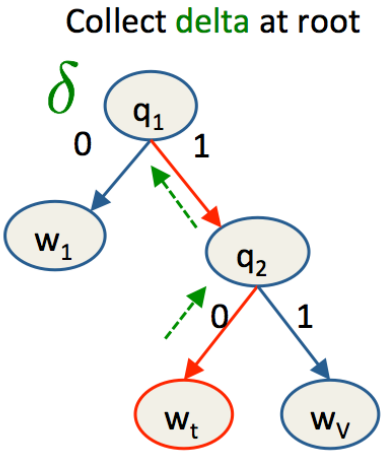


Figure 4.3: The binary hierarchical softmax predicts $P(w_t|h)$ as the product of the individual classifier probabilities on the path $L(w_t)$. The training delta on the hidden layer is computed by summing the individual deltas backwards along the same path (shown in dashed green).

given an observed word w_t in context h, from 4.1.2:

$$\mathcal{L}(w_t, h) = \log P(w_t|h) \tag{4.1.5}$$

$$= \log \prod_{i=1}^{L(w_t)-1} P(l_i|h) \tag{4.1.6}$$

$$= \sum_{i=1}^{L(w_t)-1} \log P(l_i|h) \tag{4.1.7}$$

$$= \sum_{i=1}^{L(w_t)-1} \log \sigma(\pm z_i) \tag{4.1.8}$$

where we have defined $z_i = q_i^\top r_h + b_i$ for the rest of this discussion.

In order to do gradient-based training, we take the partial derivative of $\mathcal{L}$ for each classifier on the path $i \in L(w_t)$ with respect to the output layer z, where we see that the gradient *decomposes over the path* as $\frac{\partial \mathcal{L}}{\partial z} \triangleq \delta^{\text{out}} = \sum_{i=1}^{L(w_t)-1} \delta^{\text{out}}(z_i)$, where the contribution from each classifier on the path can be expressed as

$$\begin{aligned}
\delta^{\text{out}}(z_i) &\triangleq \frac{\partial \mathcal{L}(w_t, h)}{\partial z_i} \\
&= \frac{\partial}{\partial z_i} \log \sigma(\pm z_i) && \triangleright\ \frac{\partial}{\partial z} \log f(z) = \frac{1}{f(z)} f'(z) \\
&= \frac{\cancel{\sigma(\pm z_i)}}{\cancel{\sigma(\pm z_i)}} (1 - \sigma(\pm z_i)) \frac{\partial}{\partial z_i} (\pm z_i) && \triangleright\ \frac{\partial}{\partial z} \sigma(f(z)) = \sigma(f(z))(1 - \sigma(f(z))) \frac{\partial f(z)}{\partial z} \\
&= (1 - \sigma(\pm z_i)) \frac{\partial}{\partial z_i} (\pm z_i)
\end{aligned} \tag{4.1.9}$$

Notice that the gradient is defined for $\pm z_i$. Therefore, for positive (right-branching) labels it becomes $(1 - \sigma(z_i)))(+1)$ and for negative (left branching) labels recall that $\sigma(-z_i) = 1 - \sigma(z_i)$, and therefore the gradient becomes $(1 - (1 - \sigma(z_i)))(-1) = 0 - \sigma(z_i)$, which is just the *cross-entropy error* (§ 2.3.2.3).

For the node parameters q_i, we can substitute this error into the backprop delta-rule to obtain the deltas on the classifier weights q_i as

$$\delta^{q_i} = \underbrace{(1 - \sigma(\pm z_i))}_{\text{scalar error}} \cdot \underbrace{r_h}_{\text{direction}} \qquad \triangleright\ z_i = q_i^\top r_h + b_i$$

where again $r_h = f_\theta(h)$ are the activations on the penultimate layer.

Since we pass error gradients up the tree (Fig. 4.3), we see that during stochastic gradient descent training each node i on the path to the target word w_t will receive the following updates (η is the learning rate):

$$q_i^{(T)} \leftarrow q_i^{(T-1)} + \eta \delta^{q_i}(w_t, h) \tag{4.1.10}$$

and over all training cases $w_t, h \in \mathcal{D}$, we can summarise the delta for a node i as the contributions resulting from all its children nodes $w_t \in child(i)$ that appeared as targets during training

$$\delta^{q_i} = \sum_{w_t, h \in \mathcal{D}} (1 - \sigma(\pm z_i)) \cdot r_h, \qquad s.t.\ \ w_t \in child(i) \tag{4.1.11}$$

where

In order for the HS to assign a high likelihood to an observed word w_t according to 4.1.2, we require that the individual $P(l_i|h)$ classifiers along the given path $L(w_t)$ assign a high probability to each branching decision, to ensure that the product of these probabilities will also be high. Since each $P(l_i|h) \propto q_i^\top r_h$, to ensure a high likelihood we require that the dot product of the node feature vectors q_i with r_h be large (positive or negative). This is indeed shown by q_i's update equation (4.1.10), which indicates that q_i's are updated in the directions of the context-vectors in which their children leaf nodes (target words) appear, i.e. to be more similar to the context vectors, in turn yielding a larger dot product.

Without loss of generality, let us consider a bigram model $P(w_t|w_j)$, i.e. where $r_h = f_\theta(h) = r_j$. Now, it is easy to see that a large dot product $q_i^\top r_j$ requires that the q_i on the path to w_t be "close" to the embedding for the context word r_j. In order to ensure that for some context word w_j, $P(w_t|w_j)$ is high, we therefore require q_i in the path $L(w_t)$ to have a large dot product with all context-words with which w_t is likely to co-occur in the training set (i.e. q_i will be updated to be similar to their embeddings).

Therefore: q_i converges to the weighted centroid (average) of the context-vectors in which its children leaf or target-words appear. The weights in this weighted average are defined by the frequencies of observing each child word (leaf node) during training.

Our main insight here is that, given a binary HS tree structure, in order to maximise likelihood under a training distribution $P(w,h)$, the optimal values that each node feature vector q_i can obtain after training can be stated in terms of a difference of expectations, that we will call the **Hierarchical Softmax Equilibrium** condition:

$$q_i \overset{\text{optimal}}{=} \mathop{\mathbb{E}}_{w_t,h\sim P(w,h)} \left[\mathbb{1}[w_t \in \textbf{right}\text{-child}(i)] \cdot r_h - \mathbb{1}[w_t \in \textbf{left}\text{-child}(i)] \cdot r_h\right]. \quad (4.1.13)$$

This equation follows from the updates defined in 4.1.10 in order to *minimise* predicted scores (dot products) for words in a node's left branch and *maximise* scores for words in its right branch. The expectation arises since each respective update is made by sampling a (w_t, h) training pair from the training distribution.

Intuitively, this means that in order to *best* predict large positive scores (right branching decisions) for words in its right branch, and large negative scores (left branching decisions) for words in its left branch, q_i should *compromise between* (geometrically: move towards and away from) the observed context-vectors of the two branches, proportionally to how frequently words in each branch is observed in the training data.

4.1.5 Optimal tree structure: Speed vs likelihood

A balanced b-ary tree provides on average an $O(\log_b V)$ path length $|L(w)|$ per word, which, assuming uniform word frequency, means the expected empirical cost of traversing the tree will be $\sum_i^V P(w_i)|L(w_i)| = 1/V \sum_i^V O(\log_b V) = O(\log_b V)$.

To speed up evaluation, one wants to minimise the expected empirical cost of traversing the tree. Since word frequency is highly non-uniform, one may achieve this by structuring the tree such that more frequent words have a shorter path length. In information theory this principle of encoding more frequent symbols using a shorter code is called Huffman Coding [74], and the tree corresponding to this principle is called a Huffman tree. Indeed, it is the **speed-optimal** way of structuring the tree (i.e. it minimises expected empirical traversal cost, $\mathbb{E}_{P(w,h)}\left[|L(w)|\right]$), but for reasons we will discuss below, this is not the **likelihood-optimal** way of structuring the tree. The likelihood-optimal tree structure aims to maximise *expected model likelihood* $\mathcal{L}$ with respect to $L(w)$ (where model parameters θ are consid-

ered fixed), i.e $\max_{L(w)} \mathcal{L}(\theta, L(w))$ for

$$\begin{aligned}\mathcal{L}(\theta, L(w)) &= \mathop{\mathbb{E}}_{(w_t,h)\sim P_{\text{empirical}}(w,h)} [P_{HS}(w_t|h)] \\ &= \mathop{\mathbb{E}}_{(w_t,h)\sim P_{\text{empirical}}(w,h)} \left[\prod_{i=1}^{L(w_t)-1} P(l_i|h) \right] \\ &= \sum_{(w_t,h)\sim P_{\text{empirical}}(w,h)} P(w_t,h) \prod_{i=1}^{L(w_t)-1} P(l_i|h)\end{aligned} \tag{4.1.14}$$

Let us now consider how the tree structure impacts the ability of the model to achieve high conditional likelihoods (and hence low perplexities). Note that the q_i parameters are shared between words. This creates a dependency between tree structure $L(w)$ and node parameters q_i with respect to expected model likelihood, as can be seen from 4.1.14.

Intuitively, to construct a likelihood-optimal tree, we want words which are roughly equally likely in the same contexts to be in the same branch of the tree, since this makes it easier for the classifiers at each node to predict the next words. This can be seen since, during prediction, each branching decision eliminates one subset of the vocabulary, and ideally we want to be able to eliminate the least likely words at each step down the tree (i.e. move towards sub-groups of higher conditional likelihood). This implies that words should cluster in the tree in order to balance their empirical conditional likelihood in the data. The structure of the optimal tree should therefore be strongly influenced by the empirical distribution $P(w,h)$. We will refer to this as the **likelihood-optimal principle** for constructing an HS tree.

Equally important, as we have shown, during training q_i converges to the expected centroid of its children's context vectors, but as 4.1.10 and 4.1.13 show, with a non-optimal tree, for a *common context* h, q_i will be updated *towards* r_h for observed w_t, h in its right branch, but at the same time *away from* r_h for observed w_t, h appearing in its left branch. This is clearly sub-optimal and will negatively impact the convergence of the q_i parameters during training.

Therefore, in order for the HS equilibrium (4.1.13) to converge, it needs to work in conjunction with a tree structure conforming to the likelihood-optimal principle. However, since tree structure $L(w)$ is unknown, this suggests a *alternating-optimisation*-type algorithm for learning a likelihood-optimal tree structure jointly with the word and node embeddings, which we present and evaluate in the next section.

4.2 SimTree: An iterative, $O(N \log N)$ algorithm for learning a likelihood-optimal tree structure during training

As the HS Equilibrium (4.1.13) suggests, in the likelihood-optimal setting, we seek a tree structure that recursively partitions the vocabulary according to $P(w|h)$ such that words that

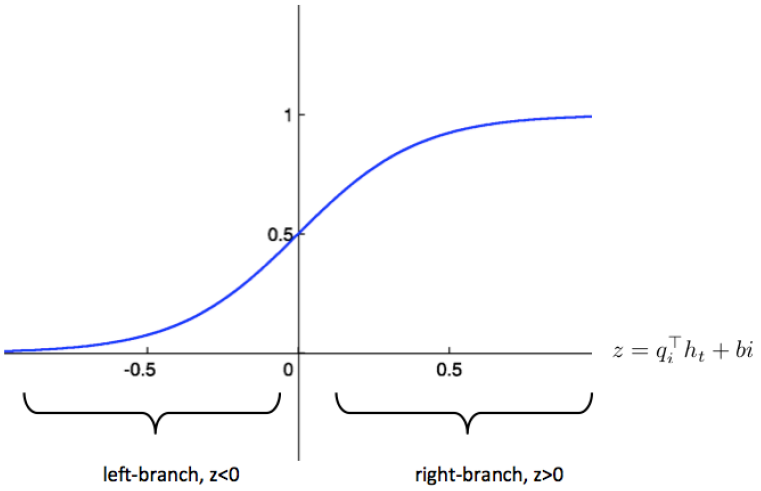


Figure 4.4: For a binary HS using logistic regression classifiers, the decision boundary for predicting left- or right-branching labels lies at $q^\top r_h + b_i = 0$.

are conditionally more likely given some h should appear in a node's right branch (this increases the individual $P(l_i|h)$ and hence the total product of probabilities for $P(w|h)$), and likewise words that are conditionally less likely should appear in its left branch. Furthermore, in the binary formulation, 4.1.4, note that the decision boundary of the left/right decision for a target-word context pair w_t, h under the sigmoid function lies at $z = q_i^\top r_h + b_i = 0$. See Fig. 4.4.

Therefore, for a current estimate of each node's feature vector q_i, we can define a recursive **likelihood-optimal clustering criterion** based on the sign of z:

$$\forall q_i \text{ we want } \begin{cases} q_i^\top \tilde{h}_j + b_i \leq 0 & \forall\ w_j \in \text{left-child}(i) \\ q_i^\top \tilde{h}_k + b_i > 0 & \forall\ w_k \in \text{right-child}(i). \end{cases} \tag{4.2.1}$$

where $\tilde{h}_j = \mathbb{E}_{w_j, h_j \sim P(w,h)}[h_j]$ is the empirical average context vector in which word w_j is observed (to make the distinction clear). We recursively divide up the words that a root node is responsible for predicting based on the sign of the score that the *current* node parameters would have assigned these words. We therefore want to cluster words such that the signs of their predicted scores are balanced between the branches, as illustrated in Fig. 4.5. This is an iterative procedure, however we will show that one iteration can be performed in $O(N \log N)$ average time for N nodes by taking inspiration from the top-down `quicksort` algorithm [75].

The computational problem clearly lies in reliably estimating $\tilde{h}$, since considering all the (exponentially many) possible contexts makes this an intractible task in general. However, we can make two approximations:

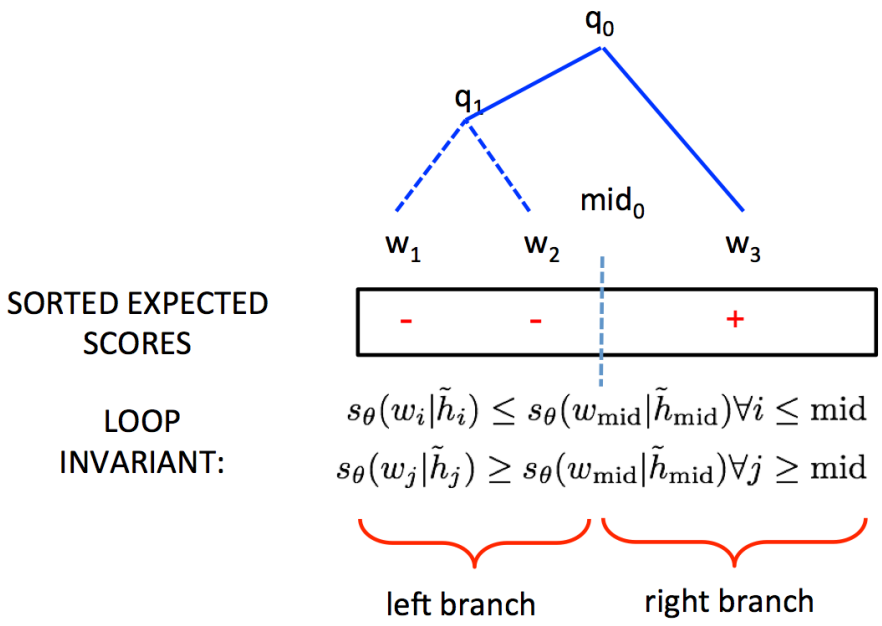


Figure 4.5: An illustration of how the SimTree algorithm partitions words to balance the expected conditional likelihoods in the two subtrees. The red signs refer to the sign of the expected score of word w_i with the feature vector for node q_0, indicating that the result of the clustering is that words in each branch have the same sign.

- As an approximation, we can view the current state of the target-word embeddings during training as summarising the information of its context-vectors, since each output embedding converges to the centroid of its context-vectors (4.1.10), and by clustering with q_i instead of $\tilde{h}_i$ should provide a proxy for clustering the vocabulary according to $P(w|h)$. This would roughly correspond to an online version of the offline algorithm proposed by [55].

- As a better approximation, but at the expense of using extra space proportional to $O(VK)$ for K-dimensional embeddings, we can store a running average of the context-vectors observed for each target word and recluster based on this. Given a $(V \times K)$ matrix $\mathbf{H}$ initialised to 0's after each reclustering, for each observed (w_t, h) pair at time step T during training, we update

$$\mathbf{H}_{[w_t,:]} = (T\mathbf{H}_{[w_t,:]} + f_\theta(h))/(1+T), \tag{4.2.2}$$

  where $f_\theta(h)$ is the activations on the hidden layer for the input context words in the context h[2]. This ensures at each step T that $\mathbf{H}_{[w_t,:]} = \mathbb{E}_{(w_t,h)\sim P_d(w,h)}[f_\theta(h)]$. However, in practice, since the network parameters are updated during training, an expo-

[2]Note that for a stream of T digits $\{s_t\}_{t=0}^{T}$, computing the arithmetic average at time T, $a_T = 1/T \sum_{i=0}^{T} s_i$ naively requires $O(T)$ storage. However, a_T can be computed *incrementally* using only $O(1)$ extra storage as $a_T = (Ta_{T-1} + s_T)/(T+1)$.

nential average works better since it favours a more recent state of the parameters, i.e

$$\mathbf{H}_{[w_t,:]} = (0.9\mathbf{H}_{[w_t,:]} + 0.1 f_\theta(h)). \tag{4.2.3}$$

Furthermore, when we change the tree structure during training, we also change the target words that a tree node is responsible for predicting. This means that the old q's may no longer be useful, but will receive big gradients as training resumes[3] to move towards the new expected centroid of the context-vectors of the new subset of words for which it is responsible.

After changing tree structure, a reasonable approach would be to initialise new q's to be the centroids of their new children embeddings (corresponding to a k-means type maximisation step). However, a better approach would be to collect counts during training of $P(w,h)$ and after reclustering, use these to weight the contribution from each target word's observed context-vectors. If we keep and update $\mathbf{H}$, then this simply corresponds to:

$$q_i = \frac{1}{Z}\left[\sum_{j \in \text{right-child(i)}} n_j \tilde{h}_j - \sum_{k \in \text{left-child(i)}} n_k \tilde{h}_k\right], \tag{4.2.4}$$

where $Z = \sum_i n_i$ is the normalisation constant required to make the subsets of weights in the two branches sum to 1. In other words, initialise q_i according to an approximation of the HS Equilibrium (4.1.13). This is an approximation since we are using the *empirical* data distribution observed over T time steps to approximate the true data distribution.

The algorithm for SimTree, shown in Algorithm 1, incorporates these ideas within an alternating-maximisation framework to construct an approximate likelihood-optimal tree by alternating between an training-step (training model parameters given a fixed tree structure, and a reclustering step (reclustering the tree according to 4.2.1) to change tree structure to maximise expected likelihood given the current parameters. Node parameters are then re-initialised based on 4.2.4 before training is resumed.

Listing 6 Quicksort algorithm.

```
def quicksort(arr, i, j):
  if i < j:
    mid = partition(arr, i, j)  # inplace
    quicksort(arr, i, mid - 1)
    quicksort(arr, mid, j)
```

An efficient top-down algorithm is employed similar to quicksort for the partitioning step. Recall that quicksort is a divide-and-conquer inplace sorting algorithm with a simple recursive definition, shown in Listing 6. Note that `partition(arr, i, j)` is an efficient

[3] Since it will be making bad predictions causing the *error* scalar weights to be high in 4.1.9.

$O(N)$ subroutine which returns the index `mid`, which maintains the **loop invariant** such that for all $i < mid$, `arr[i]` $<$ `arr[mid]`, and for all $j \geq mid$, `arr[j]` $\geq$ `arr[mid]`. We exploit this idea, but add two new ideas:

1. during each recursive call we construct and link in a new branch of the total tree that looks at a subset of the vocabulary, starting from the root node which partitions all words; and

2. we replace the condition in the loop invariant to *balance the signs* of the array entries (scores) for words to the left and right of `mid`. See Listing 7.

Listing 7 An implementation of the modified partition step which balances the signs of scores to the left and right of mid.

```
def partition(scores, left, right):
  if left >= right: return right
  else:
    i = left;
    j = right;
    while (i < j):
      while (i < j) and scores[i] < 0: i += 1
      while (j > i) and scores[j] >= 0: j -= 1
      if (i < j): SwapVocab(i - 1, j + 1)
   if (i == left or j == right):  # balanced subtrees when scores are tied
     mid = left + (right - left + 1)/2
   else:
     mid = i
   return mid
```

Putting it all together, SIMTREE works as follows:

LEARN PARAMETERS:: Clear $\mathbf{H}$. Keep tree structure fixed (starting from random) and train word and node embeddings on a subset of the data for T steps. Update $\mathbf{H}$ for each observed context vector according to 4.2.3, and keep counts n_{w_t} for the number of times w_t has been observed.

RESTRUCTURE TREE:: Keep parameters fixed and restructure the tree using 4.2.1 as clustering criterion:

1. The algorithm performs a top-down clustering of the vocabulary, starting at the root node (`parentId`=0) and considering the entire vocabulary (`left`=0, `right`=$V-1$).

2. All nodes are initialised randomly. Let i be the current `parentId`, starting at the root node $i = 0$, and incremented as new nodes are recursively created in the tree. Given

the current estimate of q_i, it then computes a (`right` - `left`)-length array (`scores`) of the z-values estimated for each target word, i.e. the dot product of each target word's expected context-vectors (in $\mathbf{H}$) with q_i, plus the bias.

3. `scores` is passed to PartitionSimTree() which computes the partitioning boundary `mid` by shuffling the vocabulary and their respective embeddings such that for all `left` $\leq i <$ `mid`, and all `mid` $\leq j <$ `right`, `scores`$[i]$ $<$ `scores[mid]` and `scores`$[j] \geq$ `scores[mid]`.

4. Next, q_i is re-initialised according to the approximate HS Equilibrium (4.2.4).

5. The algorithm then recursively continues down each subtree, linking each new subtree root into the current parent node.

The high-level algorithm is given in Algorithm 1, and the details of the clustering subroutine is given in Listing 8. The output is a V-length array of pointers to nodes for each word containing its path down the tree, and its bit vector of codes $L(w)$ which become the training labels for the node classifiers during training the HS.

Algorithm 1 SimTree training algorithm for jointly learning word and node embeddings and the tree structure for the HS.

function SimTree
  while Not converged **do**
    Learn θ: ▷ Fix tree, learn model parameters
    for $\{(w_t, h)\}_{t=1}^{T}$ **do**
      $\theta^{(t)} \leftarrow \theta^{(t-1)} + \frac{\partial \mathcal{L}(w_t,h)}{\partial \theta}$ ▷ Train on subset of dataset.
      n_{w_t} += 1
      $\mathbf{H}_{[w_t,:]} \leftarrow (0.1 r_h + 0.9 \mathbf{H}_{[w_t,:]})$ ▷ Running exponential average.
    end for
    Recluster $L(w)$: ▷ Fix parameters, learn tree
    CLUSTER_TREE()
    Reset n_{w_t} and $\mathbf{H}$
  end while
end function

4.2.1 Qualitative experiments

4.2.1.1 Reduced dataset

As a qualitative tool for inspecting the types of clusterings that the SimTree algorithm can learn, we created a toy dataset from the 12 million word AP News dataset [68] by removing

Listing 8 Top-down algorithm for constructing a binary HS tree in data structure `V` by recursively partitioning the vocabulary according to a word's score with its parent node, computed using `compute_scores()`.

```
def cluster_tree(node, parent, path, code, pos, left, right):
  if node == None:
    node = Node()
  if left >= right:
    V[right] = node
    node.id = right
    node.code = code[:]
    node.path = path + [right]
  else:
    compute_scores(scores, left, right, parent)
    mid = partition(scores, left, right)
    initialise_q(parent, left, mid, right)
    node.id = parent
    path[pos] = parent
    code[pos] = 0       # traverse left
    node.left = cluster_tree(node.left, parent, path, code, pos+1, left, mid-1)
    code[pos] = 1       # traverse right
    node.right = cluster_tree(node.right, parent, path, code, pos+1, mid, right)
  return node
```

all words that appear fewer than 2500 times in the dataset. This yields a vocabulary of only 489 words (the original vocabulary is 17,964).

We ran the SimTree algorithm for one training epoch on the toy data by filtering out training data that do not contain at least one word from the reduced vocabulary, with all other words replaced by the unknown (UNK) symbol. The SimTree algorithm was invoked once every 1/8-th of the total training word. Four invocations were found to be enough to reach convergence and running for more iterations did not change much. The learning rate was set to 0.1, and linearly decreased in the number of training cases.

4.2.1.2 Trees learned

As a qualitative evaluation of the types of clusterings that SimTree can learn, one can visualise the trees to see which words the model clusters closer together. The resulting tree over the 489 words is still somewhat unwieldy for display purposes, so we filtered out the paths for a carefully chosen selection of words, namely proper names *washington & new_york*, counting words *two, three, five, million, billion*, modals *could, would, must, will*, and days of the week *wednesday, friday, monday*. The reduced trees were then visualised using `graphviz` into tree diagrams. Internal nodes represent the internal identifier of the node (0

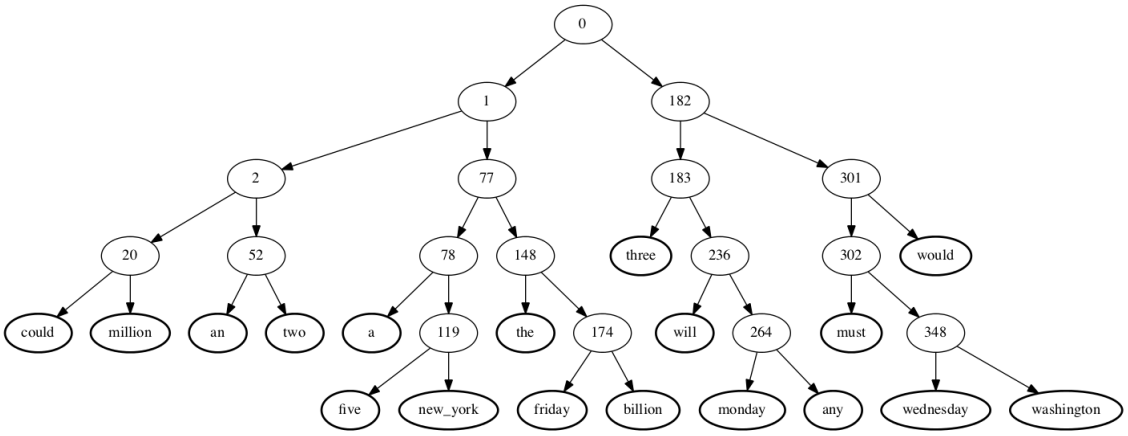


Figure 4.6: A random tree structure.

Cluster	Code prefix	Words
1	000000	a, ", the, of, #n
2	0000011	to, said, on, for, in, about, COLON, PERIOD
3	000010101	will, would, could, one, three, #an, at_least, also, after
4	00001011010010	president, election, military, association, political, saying, make, trying
5	000010110100110	countries, china, american, are, both, only

Table 4.1: Word clusters learned by training the SimTree algorithm on the reduced vocabulary AP News dataset.

through $V-2$). Note that nodes on the paths to the visualised words which are not relevant are not shown.

For comparison, we generated a random tree structure for the same words (Fig. 4.6). As expected, there is no structure to the clustering and the tree is roughly balanced. For comparison, we also generated a Huffman tree structure, based on the unigram probabilities of the words $P(w)$ (Fig. 4.7). The most distinctive property of a Huffman tree is that more frequent words cluster near the root. The visualisation for the SimTree is displayed in Fig. 4.8. The tree does not optimise tree structure for word frequency, and we see that the tree appears more balanced than the Huffman tree, with related words from the reduced set of words mostly clustering together as we would expect.

4.2.1.3 Codes learned

Another way of inspecting the learned clusters is to cluster the binary codes learned for words by prefix. A few of these clusters are displayed in Table 4.1. It is interesting to see that the model learns to cluster determiners together (cluster 1), prepositions and punctuation symbols (cluster 2), modals and counting words (cluster 3), words related to politics (cluster 4) and words related to geography (cluster 5).

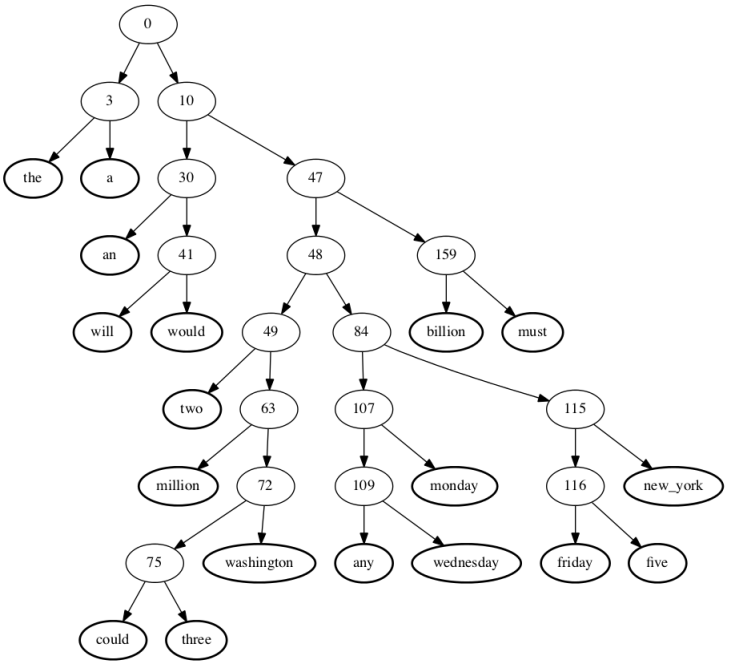


Figure 4.7: A Huffman (Shannon-Fano) tree structure.

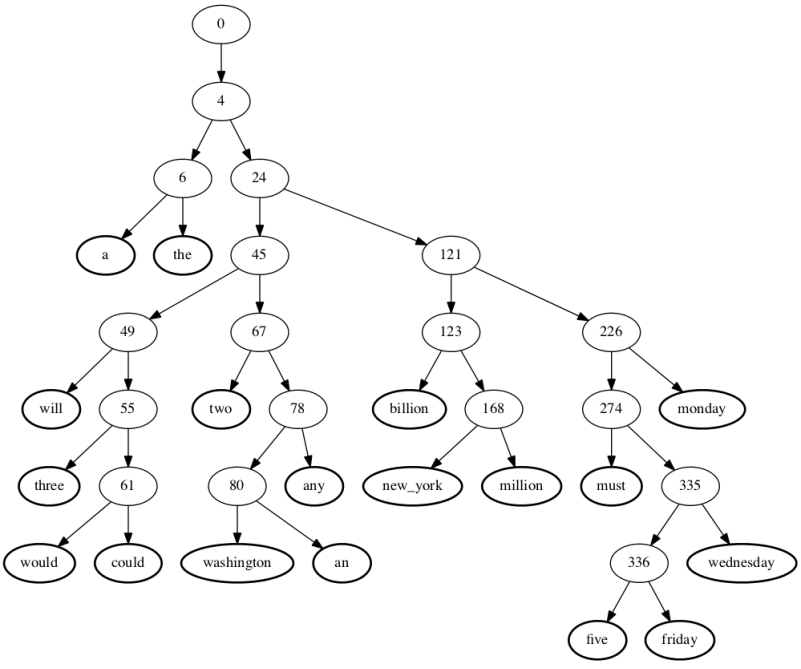


Figure 4.8: The tree structure learned by the SimTree algorithm.

4.2.2 Language modelling experiments

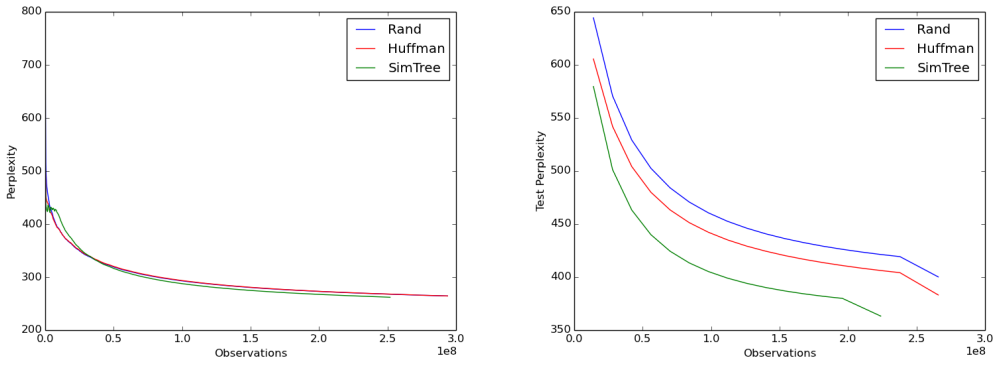
SimTree is motivated for improving model likelihood and hence perplexity (§ 4.2). In this section we compare the training and test perplexities obtained in a language modelling task by using SimTree compared to using a Huffman tree structure and a random tree baseline.

Note that the goal of this section is not to train a state of the art language model. Instead, the goal is to evaluate, all things being equal, whether using the proposed SimTree algorithm at the output layer leads to an improved model in terms of the perplexity of the model on the training and testing data. For this we trained a bag-of-words language model (i.e. a CBOW model architecture), and compute and report the model perplexity on the training and test set. Perplexity is defined as the inverse geometric average of the log-likelihoods that a model assigns to a dataset D, i.e. $\text{ppx} = b^{-\frac{1}{|D|}\sum_{i \in D} \log_b P(w_i|h_i)}$. Therefore, a model that assigns a higher probability to the observed data has a lower perplexity.

For these experiments we trained over the full AP News dataset using the full vocabulary. The AP News dataset is split into a fixed training set of 12 million words and fixed 1 million words for validation and testing respectively. The total vocabulary size is $17,964$. Input and output embeddings of 40 dimensions were used and trained using an initial learning rate of 0.1, linearly discounted with training. The initial number of training epochs is set to 3. Validation and test perplexity is computed once per epoch. If performance on validation increases, the total number of training epochs is increased by one (i.e. we train using *early stopping with patience*, § 2.3.3.2). Using this strategy, most algorithms finished training around $18 - 20$ epochs over the training data. The SimTree algorithm was used to recluster the tree every $1/8$-th of the total words in one full iteration over the data, for a total of 4 times (i.e. reclustering is performed during the first half of the first iteration over the data). One reclustering step over the $17k$ vocabulary is almost instantaneous, so one could certainly recluster multiple times, however we found multiple reclustering to offer no extra improvement in the final likelihood.

The language modelling results are displayed in Fig. 4.9. There are several interesting observations. First, we notice that the *training perplexities* do not seem to differ very much with the different tree structures as all three reach a final training perplexity of around 260. However, the *test perplexities* show a marked difference, with the random tree achieving a final test perplexity of 400.22, the Huffman tree 383.19 and the SimTree 363.27. This represents a reduction of 36.95 in perplexity over the random tree (9.2%), and 19.92 over the Huffman tree (5.1%).

Moreover, the SimTree algorithm shows a much faster convergence rate during training, with the SimTree algorithm reaching its best test perplexity after 251 million examples vs 294 million for the Huffman and Random tree, i.e. for a reduction of 14.6% in training examples. Note that we presented all training data in exactly the same sequence to all models during all runs.



(a) Training perplexity.

(b) Test perplexity.

Figure 4.9: Perplexities (lower is better) obtained on the full AP News dataset using a CBOW model architecture trained using a random tree (top curve), a Huffman tree (middle curve) and using the proposed SimTree algorithm (bottom curve).

4.3 Subsampled Hierarchical Softmax

In the previous section we looked at a technique for improving the model perplexity by restructuring the tree in an online data-driven manner. However, for *feature learning*, we are interested in learning high-quality word vectors as quickly as possible. For this, SimTree is not a good idea as it does not optimise tree structure for speed and can be a lot slower to train than for instance using a Huffman tree structure. Therefore, in this section we will look at a simple technique for both speeding up and improving the quality of the learned word vectors trained using the Huffman tree.

4.3.1 Motivation and definition

As shown in § 4.1.3, for each (w_t, h) training-pair, the gradient-update for the binary hierarchical softmax decomposes over individual classifier predictions along the path $L(w_t)$. Furthermore, the contribution to the gradient from each classifier is equal to its embedding vector q_i scaled by the scalar cross-entropy error of that classifier.

The higher a node appears in the tree, the more words it is responsible for predicting and therefore the more gradients it contributes to, with the root involved in predicting the first step in the path for all words. Furthermore, the higher a node is in the tree, the more "general" and less fine-grained the distinctions it needs to classify between the two groups of words it is responsible for.

We can therefore view the total *information* in the gradient that reaches the hidden layer as a sum of progressively more general (noisy) information as we move up the tree, with an emphasis on the contribution of the higher nodes since they appear more frequently in all word paths. The basic idea for the **subsampled hierarchical softmax** is to discard gradient

information inversely proportional to a node's depth in the tree in order to throw away more of the noise in the upper gradients, and retain more of the finer-grained information in the lower gradients. Specifically, whereas the standard gradient of the HS on the hidden layer can be written as $\sum_{i=1}^{L(w_t)-1} \underbrace{\frac{\partial}{\partial\theta} \log P(l_i|h)}_{\delta^\theta(i)}$ (see 4.1.5), we now modify this as follows:

$$\frac{\partial \mathcal{L}_{\text{SSHS}}}{\partial\theta} \triangleq \sum_{i=1}^{L(w_t)-1} [[r \sim U < P_{ss}(i)]]\delta^\theta(i), \tag{4.3.1}$$

where r is sampled uniformly at random, $P_{ss}(i)$ is the probability of sub-sampling (discarding) the gradient contribution from the node at depth i on the path $L(w_t)$, and $[[x]]$ is defined as 1 if x is true and 0 if x is false.

Note that this technique is related to the technique of subsampling at the word level, as used by [14]. However, it is motivated for a different reason, and as we will show in the experiments, it works complementary to that technique.

4.3.2 Subsampling distribution

The definition of the subsampling distribution $P_{ss}(i)$ is clearly important. For this we experimented with two options. We used the empirical distribution updated dynamically during training as $P_{ss}(i) = n_i/N$ where n_i is the number of times node i has been accessed up to that point during training, and N is the total number of nodes that have been accessed so far during training. In other words, this distribution is normalised over the entire tree. However, we found better results using a simple, fixed, exponentially-decaying distribution in the path length. The distribution is defined in terms of the maximum path length L in the tree: $P_{ss}(i) = \beta^{i/L}/Z$, where Z is the normaliser for this distribution. This is both simpler to implement and was found to give better results. We experimented with different values for β and found $\beta = 0.005$ to work well.

4.3.3 Effect of the tree structure

Higher nodes in the tree have to learn to classify between larger groups of words. If these groups are selected to match the distributional statistics of the data, then the classification problem would be easy. However, if these groups of words are selected at random, then by definition predicting class membership becomes harder, which would cause the error deltas to have higher variance and the updates to the word embeddings to be noisier.

We therefore hypothesise that this technique will have a bigger effect on random tree structures than on data-driven tree structures, and we will evaluate this hypothesis experimentally in the next section.

4.3.4 Experiments

We trained the CBOW model architecture with a context-size of 10 words with and without the subsampled hierarchical softmax using the above exponentially-decaying distribution with 100-dimensional embeddings on the full English Wikipedia (obtained from [76], 1.7 billion words as of Nov 2014). For evaluating the quality of the learned embeddings we used the standard analogical reasoning dataset and evaluation tool from the `word2vec` package [14]. The results are displayed in Table 4.2.

First, the results show that the proposed subsampling method provides a speedup during training of 14% for a random tree and 17% for a Huffman tree. This is to be expected, since we are not performing any computations for the dropped nodes.

Second, for the Huffman tree, the results show that using only HS subsampling leads to a substantial 27% increase in accuracy over not using HS subsampling, but not as much as using only subsampling at the word level (51%).

Third, when combining HS subsampling with subsampling at the word level, we get a significant improvement of 65.69% over the baseline with no subsampling, and 9.7% relative improvement over the state-of-the-art baseline that uses the hierarchical softmax with subsampling only at the word level. This improvement in accuracy comes in spite of a reduction in training time of 17%.

Finally, we observe that the technique has a larger net effect for a random tree (34.4%) than for a Huffman tree (27%). A Huffman tree is constructed according to the empirical distribution of the words $P(w)$. This validates the hypothesis that the proposed technique will have a larger effect for random trees versus for trees constructed in a data-driven manner.

Interestingly, this technique does not help when training with the skip-gram architecture. However, using this technique improves the accuracy using the CBOW architecture to be comparable to the accuracy obtained with the skipgram architecture (the best skipgram result with the HS is 42.2), despite the fact that training the skipgram architecture is roughly $O(|h|)$ *more expensive* (for $|h|$ the number of context words) than training the CBOW.

4.4 Conclusion

This chapter provided an in-depth look at hierarchical output layers, and in particular the binary hierarchical softmax. Through detailed analytical analysis we discussed the computational speedups that methods like Huffman trees can give. We also looked at how the tree structure impacts the ability of the model to model the data well. For this we proposed a novel, fast, iterative EM-type algorithm for jointly learning tree structure with the model parameters. We presented experimental evidence that SimTree can learn interesting semantic

Tree	Word SS	HS SS	Time (m)	Total Acc	Syntactic	Semantic
Random	No	No	32	9.92	11.15	8.91
Random	No	**Yes**	25 (-22%)	13.33 (+34.4%)	15.97	11.16
Random	Yes	No	14	23.94 (+141.33%)	35.36	14.51
Random	Yes	**Yes**	12 (-14%)	**26.43 (+166.43%)**	40.11	15.15
Huffman	No	No	22	24.95	16.21	32.16
Huffman	No	**Yes**	17 (-23%)	31.68 (+27%)	22.53	39.23
Huffman	Yes	No	12	37.70 (+51.10)	35.98	39.13
Huffman	Yes	**Yes**	10 (-17%)	**41.34 (+65.69%)**	42.02	40.74

Table 4.2: Results on the analogical reasoning dataset of [15] using 100-dimensional embeddings induced using the CBOW architecture and trained with the hierarchical softmax with and without word-level subsampling (**Word SS**) and the proposed subsampled hierarchical softmax technique (**HS SS**). *(+x%)* indicates x percent improvement over the baseline of not using any subsampling technique.

clusterings, and also yield significant improvements in the model perplexity on a language modelling task.

This chapter also looked at a simple technique for speeding up training and improving the accuracy of learning word embeddings using the hierarchical softmax by stochastically discarding contributions to the gradient from nodes higher up in the tree. This was shown to lead to a roughly 10% improvement in the quality of the learned representations while at the same time reducing training time by 17%.

Chapter 5

BilBOWA: Bilingual Bag-of-Words without Alignments

This chapter introduces BilBOWA (*Bilingual Bag-of-Words without Alignments*), a simple and computationally-efficient model for learning bilingual distributed representations of words which can scale to large monolingual datasets and does not require word-aligned parallel training data. Instead it trains directly on monolingual data and extracts a bilingual signal from a smaller set of raw-text sentence-aligned data. This is achieved using a novel sampled bag-of-words cross-lingual objective, which is used to regularise two noise-contrastive language models for efficient cross-lingual feature learning. We show that bilingual embeddings learned using the proposed model outperform state-of-the-art methods on a cross-lingual document classification task as well as a lexical translation task.

The code can be found at https://github.com/gouwsmeister/bilbowa. A preliminary version of the work in this chapter was presented at the NIPS 2014 Deep Learning Workshop and an expanded version of the work was presented at the International Conference of Machine Learning (ICML 2015).

5.1 Introduction

Raw text data is freely available in many languages, yet labeled data – e.g. text marked up with parts-of-speech or named-entities – is expensive and mostly available for English. Although several techniques exist that can learn to map hand-crafted features from one domain to another [77–79], it is in general non-trivial to come up with good features which generalise well across tasks, and even harder across different languages. It is therefore very desirable to have unsupervised techniques which can learn useful syntactic and semantic features that are invariant to the tasks or languages that we are interested in.

As we have seen so far during this dissertation, unsupervised *distributed* representations of words capture important syntactic and semantic information about languages and

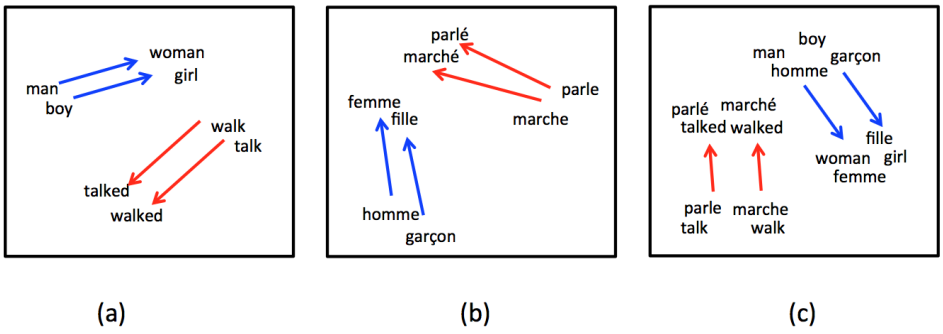


Figure 5.1: Monolingual embeddings (a & b) have been shown to capture syntactic and semantic features such as noun gender (shown in blue) and verb tense (shown in red). The goal of crosslingual embeddings shown in (c) is to capture these relationships across two or more languages.

these techniques have been succesfully applied to a wide range of tasks [8, 9], across many different languages [76]. Traditionally, inducing these representations involved training a neural network language model (§ 3.3.1) which was slow to train. However, contemporary word embedding models are much faster in comparison, and can scale to train on billions of words per day on a single desktop machine [58, 65, 80]. In all these models, words are represented by learned, real-valued feature vectors referred to as *word embeddings* and trained from large amounts of raw text. These models have the property that similar embedding vectors are learned for similar words during training. Additionally, the vectors capture rich linguistic relationships such as *male-female* relationships or verb tenses, as illustrated in Figure 5.1 (a) and (b). These two properties improve generalisation when the embedding vectors are used as features on word- and sentence-level prediction tasks.

A related goal is to induce distributed representations *over different language-pairs* in order to serve as an effective way of learning linguistic regularities which generalise across languages, in that words with similar distributional syntactic and semantic properties in both languages are represented using similar vectorial representations (i.e. embed nearby in the embedded space, as shown in Figure 5.1 (c)). This is especially useful for transferring limited label information from high-resource to low-resource languages, and has been demonstrated to be effective for document classification [16], outperforming a strong machine-translation baseline; as well as named-entity recognition and machine translation [81, 82].

Since these techniques are fundamentally data-driven techniques, the quality of the learned representations improves as the size of the training data increases [58, 80]. However, as we will discuss in more detail in §5.2, there are two significant drawbacks associated with current bilingual embedding methods: they are either very slow to train or they can only exploit parallel training data. The former limits the large-scale application of these techniques,

while the latter severely limits the amount of available training data, and furthermore introduces a big domain bias into the learning process, since parallel data is typically only easily available for certain narrow domains (such as parliamentary discussions).

This chapter introduces **BilBOWA** (*Bilingual Bag-of-Words without Word Alignments*), a simple, scalable architecture for inducing bilingual word embeddings with a trivial extension to multilingual embeddings. The model is able to leverage essentially unlimited amounts of monolingual raw text. It furthermore does not require any word-level alignments, but instead extracts a bilingual signal directly from a limited sample of sentence-aligned, raw-text parallel data (e.g. Europarl) which it uses to align embeddings as they are learned over monolingual training data. Our contributions are the following:

- We introduce a novel, computationally-efficient *sampled cross-lingual objective* ("BilBOWA-loss") which is employed to align monolingual embeddings as they are being trained in an online setting. The monolingual models can scale to large-scale training sets, thereby avoiding training bias, and the BilBOWA-loss only considers sampled bag-of-words sentence-aligned data at each training step, which scales extremely well and also avoids the need for estimating word-alignments (§5.3.2);
- we experimentally evaluate the induced cross-lingual embeddings on a document-classification (§5.5.2) and lexical translation task (§5.5.3), where the method outperforms current state-of-the-art methods, with training time reduced to minutes or hours compared to several days for prior approaches;
- finally, we make available our efficient C-implementation to hopefully stimulate further research on cross-lingual distributed feature learning[1].

5.2 Learning Cross-lingual Word Embeddings

Monolingual word embedding algorithms [58, 80] learn useful features about words from raw text (e.g. Fig 5.1 (a) & (b)). These algorithms are trained over large datasets to be able to predict words from the contexts in which they appear. Their working can intuitively be understood as mapping each word to a learned vector in an embedded space, and updating these vectors in an attempt to simultaneously minimise the distance from a word's vector to the vectors of the words with which it frequently co-occurs (§ 3.6). The result of this optimisation process yields a rich geometrical encoding of the distributional properties of natural language, where words with similar distributional properties cluster together. Due to their general nature, these features work well for several NLP prediction tasks [8, 9].

In the cross-lingual setup, the goal is to learn features which generalise well *across different tasks and different languages*. The goal is therefore to learn features (embeddings) for

[1] https://github.com/gouwsmeister/bilbowa

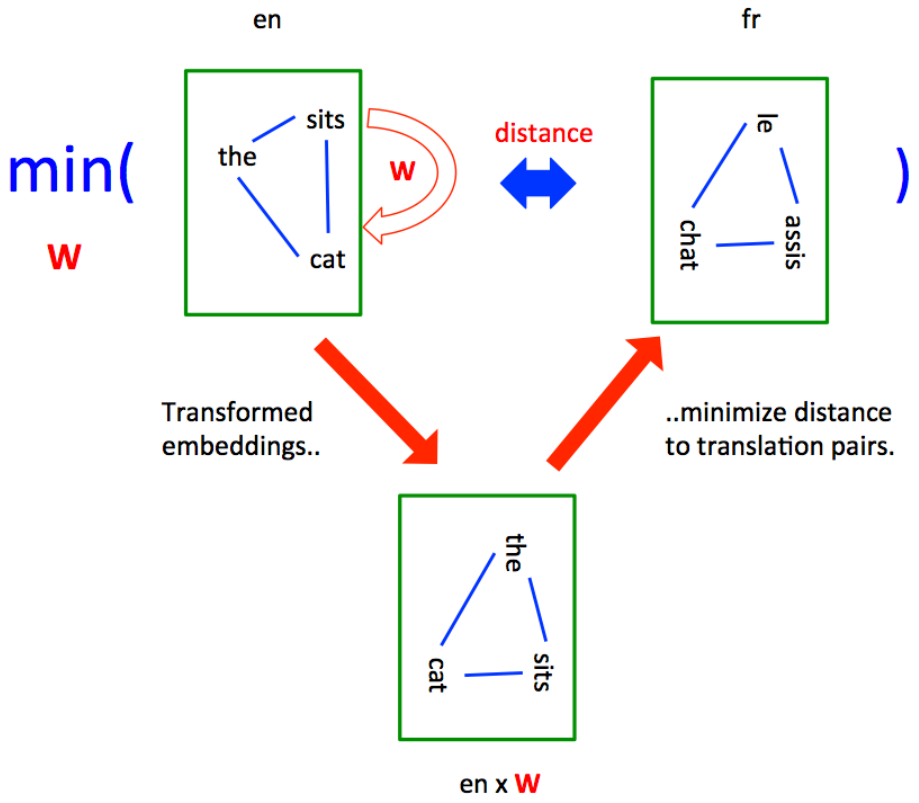


Figure 5.2: An illustration of an *offline* alignment method for learning bilingual word embeddings. Embeddings are trained separately on two languages and afterwards a projection is learned that projects the embeddings of English words onto the embeddings of their French translation-pairs.

each word such that *similar words in each language* are assigned similar embeddings (the **monolingual objectives**), but additionally we also want *similar words across languages* to have similar representations (the **cross-lingual objective**). The latter property allows one to use the learned embeddings as features for training a discriminative classifier to predict labels in one language where we have labelled data (e.g. topics, parts-of-speech, or named-entities), and then directly transfer the classifier to a language for which we do not have much labelled data. From an optimisation perspective, there are several approaches to how one can optimise these two objectives (our classification):

Offline Alignment: The simplest approach is to optimise each monolingual objective separately (i.e. train embeddings on each language separately using any of the several available off-the-shelve toolkits), and then enforce the cross-lingual constraints as a separate, disjoint, 'alignment' step. The alignment step consists of learning a transformation for projecting the embeddings of words onto the embeddings of their translation pairs, obtained from a dictionary, as illustrated in Fig. 5.2. This was shown to be a viable approach by [82]

who learned a linear projection from one embedding space to the other. It was extended by [83], who simultaneously projected source and target language embeddings into a joint space using canonical correlation analysis. The advantage of this approach is that it is very fast to learn the embedding alignments. The main drawback of this approach is that it is not clear that a single transformation (whether linear or nonlinear) can capture the relationships between all words in the source and target languages, and our improved results on the translation task seem to point to the contrary (§5.5.3). Furthermore, an accurate dictionary is required for the language-pair and the method considers only one translation per word, which ignores the rich multi-sense polysemy of natural languages.

Parallel-only: Alternatively, one may leverage purely sentence-aligned parallel data and train a model to learn similar representations for the aligned sentences. This is the approach followed by the Bilingual Compositional Vector Models (BiCVM [84]) and the Bilingual Auto-Encoder (BAE, [85]). The advantage of this approach is that it can be fast when using an efficient noise-contrastive training criterion like that of the BiCVM. The main drawbacks of this method are that it can only train on *limited parallel data*, which is expensive to obtain and not necessarily written in the same style or register as the domain where the features might be applied (i.e. there is a strong *domain bias*).

Jointly-trained Models: Another approach is to *jointly optimise* the monolingual objectives $\mathcal{L}(\cdot)$, with the cross-lingual objective enforced as a **cross-lingual regulariser** (see Figure 5.3 for a schematic). To do this, we define a cross-lingual regularisation term $\Omega(\cdot)$, and use it to constrain monolingual models as they are jointly being trained over the context h and target word w_t training pairs in the dataset $\mathcal{D}$, e.g.

$$\mathcal{L}_{\text{JOINT}} = \min_{\theta^e,\theta^f} \sum_{l\in\{e,f\}} \sum_{w_t,h\in\mathcal{D}_l} \underbrace{\mathcal{L}_l(w_t,h;\theta^l)}_{\text{feature learning}} + \lambda \underbrace{\Omega(\theta^e,\theta^f)}_{\text{alignment}}. \tag{5.2.1}$$

This formulation captures the intuition that we want to learn representations which model their individual languages well (the first term) while the $\Omega(\cdot)$ regulariser encourages representations to be similar for words that are related across the languages. Conceptually, this regulariser consists of minimizing a distance function between the vector representations $\mathbf{r}_i$ learned for words w_i in the two domains, weighted by how similar they are, i.e.

$$\Omega(\mathbf{R}^e,\mathbf{R}^f) = \sum_{w_i\in V^e} \sum_{w_j\in V^f} \text{sim}(w_i,w_j)\cdot \text{distance}(\mathbf{r}_i^e,\mathbf{r}_j^f). \tag{5.2.2}$$

where we use $\mathbf{R}$ to denote learned embedding representations, and $\mathbf{r}_i$ to denote the embedding learned for word w_i. In other words, when this weighted sum (and hence its contribution to the total objective) is low, one can be sure that words across languages that are similar (i.e. high $\text{sim}(w_i,w_j)$) will be embedded nearby each other.

This approach was shown to be useful by [16]. Crucially, the advantages of this formulation are that it *enables one to train on any available monolingual data*, which is both more

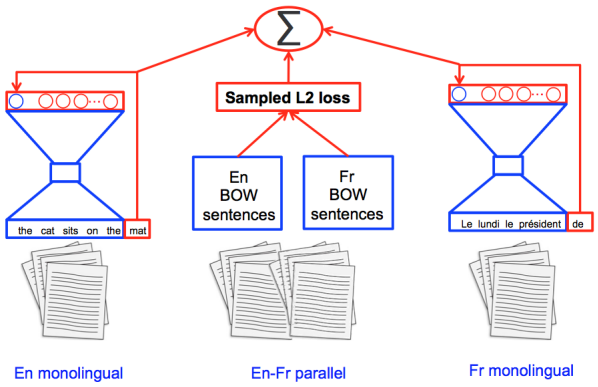


Figure 5.3: Schematic of the proposed BilBOWA model architecture for inducing bilingual word embeddings. Two monolingual skipgram models are jointly trained while enforcing a sampled L_2-loss which aligns the embeddings such that translation-pairs are assigned similar embeddings in the two languages.

abundant and less biased than the parallel-only approach, since one can train on data which resembles the data one will be applying the learned features to. The disadvantage is that the original model of Klementiev et al. is extremely slow to train. The training complexity stems both from how the authors implement their monolingual and cross-lingual objectives. For the monolingual objective, they train a standard neural language model for which the complexity of the output softmax layer grows with the output vocabulary size. Therefore, in order to evaluate the model the authors had to reduce the output vocabulary to only the 3000 most frequent words. The second reason for the slow training time is that the cross-lingual objective considers the interactions between all pairs of words between the source and target vocabulary *at each training step*, which scales as the product of the two vocabularies. In this work, we address these two issues individually.

5.3 The BilBOWA Model

As discussed in §5.2, the primary challenges with existing bilingual embedding models are their *computational complexity* (due to an expensive softmax or an expensive regularisation term, or both), but most importantly, the strong *domain bias* that is introduced by models that train only on parallel data such as Europarl.

The BilBOWA model is designed to overcome these issues in order to enable computationally-efficient cross-lingual distributed feature learning over large amounts of monolingual text. A schematic overview of the model is shown in Figure 5.3. The two main aspects (discussed in the following sections) are

1. Firstly, similar to [81], we leverage advances in monolingual feature learning algorithms by replacing the softmax objective with a more efficient noise-contrastive ob-

jective (§5.3.1), allowing monolingual training updates to scale independently of the vocabulary size.

2. Secondly, we introduce a novel computationally-efficient cross-lingual loss which only considers sampled, bag-of-words sentence-aligned data for the cross-lingual objective (§5.3.2). This avoids the need for estimating word alignments, but moreover, the computation of the regularisation term reduces to only the words in the observed sample (compared to considering the $O(V^2)$ worst-case possible interactions at each training step in the naive case).

5.3.1 Learning Monolingual Features: The $\mathcal{L}$ term

Since we do not care about language modelling, but more about feature learning, an alternative to the softmax is to use a **noise-contrastive approach** to score valid, observed combinations of words against randomly sampled, unlikely combinations of words (see § 3.3.5). This idea was introduced by Collobert and Weston [8] where they optimised a margin between the observed score and the noise scores. In their formulation, scores were computed on *sequences* of words, but in [82] this idea was taken one step further and successfully applied to *bag-of-word* representations of contexts in their continuous bag-of-words (CBOW) and skipgram models trained using the negative sampling training objective (a simplified version of noise-contrastive estimation [65], refer to § 3.4.3.5). Any of these objectives would yield comparable speedup and could be used in our architecture. In this work we opted for the skipgram model trained using negative sampling since it has been shown to learn high-quality monolingual features.

5.3.2 Learning Cross-lingual Features: The BilBOWA-loss (Ω term)

Besides learning how words in one language relate to each other, it is equally important for the representations to capture how words between the two languages relate to each other, which we enforce using the Ω term in equation 5.2.1. In the general bilingual setting, word similarities can be expressed as a matrix $\mathbf{A}$ where a_{ij} encodes the translation "score" of word i in one language with word j in the other. In the rest of our discussion we will refer specifically to English and French, and denote all English-specific parameters using e superscript, and all French-specific parameters with f superscript.

If the K-dimensional word embedding row-vectors $\mathbf{r}_i$ are stacked to form a (V, K)-dimensional matrix $\mathbf{R}$, then we can express what we will refer to as the **exact cross-lingual objective** as follows:

$$\Omega_{\mathbf{A}}(\mathbf{R}^e, \mathbf{R}^f) = \sum_i \sum_j a_{ij} ||\mathbf{r}_i^e - \mathbf{r}_j^f||^2 \tag{5.3.1}$$

$$= (\mathbf{R}^e - \mathbf{R}^f)^\top \mathbf{A} (\mathbf{R}^e - \mathbf{R}^f). \tag{5.3.2}$$

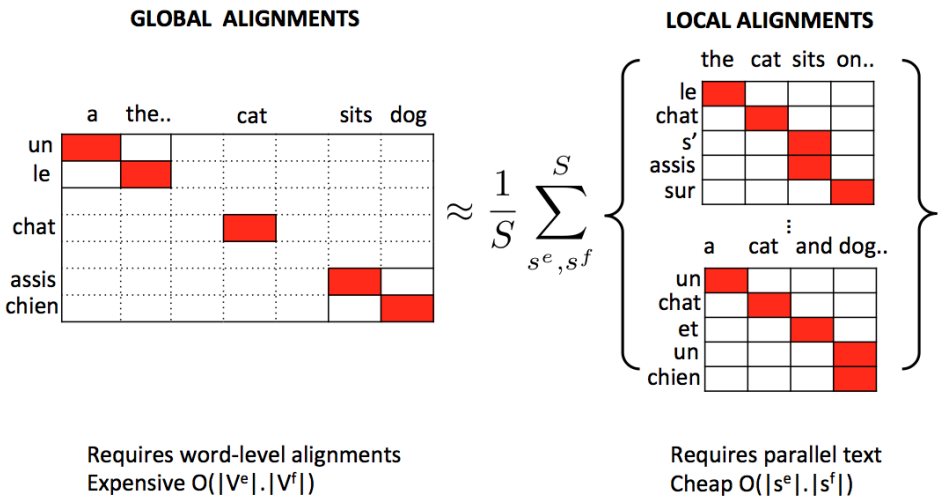


Figure 5.4: Using global word-alignments for aligning cross-lingual embeddings (equation 5.3.1) is costly and scales as the product of the two vocabulary sizes. In contrast, the BilBOWA-loss (equation 5.3.5) approximates the global loss by averaging over implicit local co-occurrence statistics in a limited sample of parallel sentence-pairs.

where subscript $\mathbf{A}$ indicates that the alignments are fixed (given). $\mathbf{A}$ captures the relationships between all V^e words in English with respect to all V^f words in French, and is indeed also the source of the two main challenges in this formulation, namely:

1. how to derive or learn which words to pair as translation pairs (i.e. deriving/learning $\mathbf{A}$);

2. how to *efficiently* evaluate $\Omega(\cdot)$ during training, since naively evaluating it scales as the product of the two vocabulary sizes $O(|V^e| \cdot |V^f|)$ *at each training step*.

The cross-lingual objective therefore penalises the Euclidian distance between words in the two embedding spaces ($\mathbf{R}^e$ and $\mathbf{R}^f$) proportional to their alignment frequency. Previous work approached this step by performing a word-alignment step prior to training to learn the alignment matrix $\mathbf{A}$. However, performing word alignment requires running Giza++ [86] or FastAlign [87] software and training HMM word-alignment models. This is both computationally costly and also noisy. We would like to learn the translation correspondences without utilizing word alignments. In order to do that, we directly exploit the parallel training data. The main contribution of this work is to approximate the costly $\Omega(\cdot)$ term, defined in equation 5.3.1 in terms of the *global* word-*alignment* statistics, using cheaply-obtained *local* word *co-occurrence* statistics obtained from raw-text parallel sentence-pairs (i.e. without running a word-alignment step). The main concept is illustrated schematically in Figure 5.4, and discussed in more detail below.

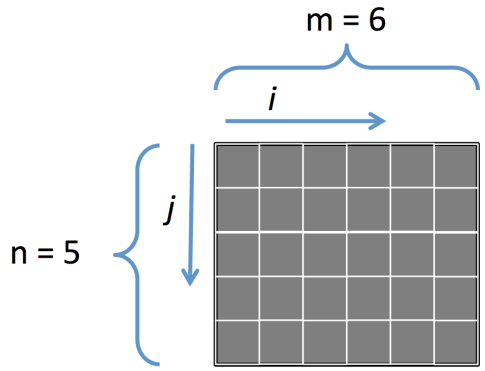


Figure 5.5: A uniform word alignment model assigns equal probability to each word in the English side (left to right) being aligned with each word in the French side (down), visualised here as a uniformly tinted alignment distribution over all word-pairs.

As a first step, notice that since the alignment weights can be normalised to sum to one, we can interpret the alignment weights as a distribution and write equation 5.3.1 as an expectation over the distribution of English and French word-alignment probabilities $a_{ij} = P(w_i^e, w_j^f)$,

$$\Omega_{\mathbf{A}}(\mathbf{R}^e, \mathbf{R}^f) = \mathop{\mathbb{E}}_{(i,j)\sim P(w^e,w^f)} \left[||\mathbf{r}_i^e - \mathbf{r}_j^f||^2 \right] \tag{5.3.3}$$

Since the parallel data is paired at the sentence level, we know that translation pairs for the *en* sentence occur in the *fr* sentence, but we do not know where. We therefore need a word-alignment model. A naive assumption is to assume that each observed *en* word can potentially be aligned with each observed *fr* word (i.e. to assume a **uniform word-alignment model** as illustrated in Fig. 5.5) for each word *in the observed sentence pairs*. Under this assumption, one can then approximate equation 5.3.3 by sampling S m-length English and n-length French sentence-pairs (s^e, s^f) from the parallel training data:

$$\Omega_{\mathbf{A}}(\mathbf{R}^e, \mathbf{R}^f) \approx \frac{1}{S} \sum_{(s^e,s^f)\in S} \frac{1}{mn} \sum_{w_i \in s^e} \sum_{w_j \in s^f} ||\mathbf{r}_i^e - \mathbf{r}_j^f||^2 \tag{5.3.4}$$

It is important to note that the lengths of the sampled English and French parallel sentences m and n need not be equal, and more importantly $m \ll |V^e|$ and $n \ll |V^f|$. Furthermore, notice that under a uniform alignment model, at each training step, each word in the sampled English sentence s^e will be updated towards all words in the French sentence s^f. We can precompute this by simply updating each English word towards the mean French bag-of-words sentence-vector. Specifically, we implement equation 5.3.4 by sampling only one parallel sentence-pair per training step (i.e. $S = 1$), and at each training step t we optimise the following **sampled, approximate cross-lingual objective**:

$$\Omega_{\mathbf{A}}^{(t)}(\mathbf{R}^e, \mathbf{R}^f) \triangleq ||\frac{1}{m} \sum_{w_i \in s^e}^{m} \mathbf{r}_i^e - \frac{1}{n} \sum_{w_j \in s^f}^{n} \mathbf{r}_j^f||^2 \tag{5.3.5}$$

where s^* denotes the English or French sampled sentence-pair drawn from the parallel corpus. In other words, *the BilBOWA-loss minimises a sampled L_2-loss between the mean bag-of-words sentence-vectors of the parallel corpus*. On its own, this objective is degenerate since all embeddings would converge to the trivial solution (by collapsing all embeddings to the same value), but coupled as a regulariser with the monolingual losses, we find that it works very well in practice. By sampling training sentences from the parallel document distribution, this objective efficiently approximates equation 5.3.1 (the more two words are observed together in a parallel sentence-pair, the stronger the embeddings for the two words will be pushed together, i.e. proportional to a_{ij}), without having to actually compute the word alignment weights a_{ij}.

5.3.3 Parallel subsampling for better results

Equation 5.3.5 is an approximation of equation 5.3.1. As illustrated in Figure 5.4, we are really interested in estimating the global word-alignment statistics for a word-pair, i.e. a_{ij}. However, by sampling words at the sentence-level, the local alignment statistics are skewed by the words' *unigram frequencies of occurrence* in a given sentence (i.e. regardless of alignment). Since language has a very strong Zipfian distribution we therefore find in practice that equation 5.3.5 *over-regularises the frequent words*. A simple solution to this is to subsample (discard) words from the parallel sentences proportional to their unigram probabilities of occurrence. In other words, we discard each word from the parallel sentences with a probability that depends on its unigram frequency of occurrence. This heavily discards frequent words and effectively flattens the unigram distribution to a uniform distribution. This idea is closely related to the monolingual subsampling employed in the `word2vec`-type models, although it is motivated for a different reason in the cross-lingual setting.

In practice we find this useful for learning finer-grained cross-lingual embeddings for the frequent words. To better illustrate the effect this has on training, we *jointly* visualised comparable. the top-25 most frequent words in English and German using the t-SNE algorithm [88]. This is illustrated in Figure 5.6. We show in red the embeddings trained without subsampling and in blue the embeddings for the same words trained using parallel subsampling. As the visualisation shows, without subsampling frequent words are over-regularised and cluster near the origin. This effect is largely reduced by the proposed subsampling scheme.

5.4 Implementation and Training Details

We implemented our model in C by building on the popular open-source `word2vec` toolkit[2]. The implementation launches a monolingual skipgram model as a separate thread for each

[2] https://code.google.com/p/word2vec/

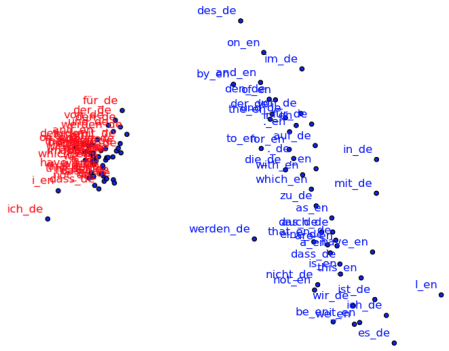


Figure 5.6: A joint t-SNE visualisation of the *same* 25 most frequent English and German words trained *without* (red, left) and *with* parallel subsampling (blue, right), illustrating the effect that occurs without parallel subsampling, as frequent words are over-regularised towards the origin.

language, as well as a cross-lingual thread. All threads access the shared embedding parameters asynchronously. For training the model, we make use of online asynchronous stochastic gradient descent (ASGD), where at time step t, parameter θ is updated as

$$\theta^{(t)} = \theta^{(t-1)} - \eta \frac{\partial \mathcal{L}_{\text{JOINT}}}{\partial \theta} \tag{5.4.1}$$

Our initial implementation synchronised updates between threads, but we found that simply clipping individual updates to $[-0.1, 0.1]$ per thread was sufficient to ensure training stability and considerably improved training speed. This is consistent with a recent trend towards asynchronous SGD (e.g.˜ []hogwild2011). For monolingual training data, we use the freely available, pre-tokenised Wikipedia datasets [76]. For cross-lingual training we use the freely-available Europarl v7 corpus [89]. Unlike the approach of [16] however, we do not need to perform a word-alignment step first. Instead our implementation trains directly on the raw parallel text files obtained after applying the standard preprocessing scripts that come with the data to tokenise, recase and remove all empty sentence-pairs. Embedding matrices were initialised by drawing from a zero mean, unit-variance gaussian distribution. The skipgram negative sampling objectives (a simplified form of noise-contrastive estimation, § 3.4.3.5) require us to sample k noise words per training pair from the unigram $P(w)$ *en* and *fr* distributions, and we set $k = 15$ which has been shown to give good results.

Doing each training update therefore occurs asynchronously across the threads. Monolingual threads each select a context-target (h, w_t)-pair for each language and sample k noise words according to their unigram noise distributions. The cross-lingual thread samples a random pair of parallel sentences from the parallel data. Finally, each thread makes an update to all parameters asynchronously according to equation 5.4.1, for which gradients are easy to compute due to the square-loss of the BilBOWA-loss and the log-linear nature of the skipgram models. The learning rate was set to 0.1, with linear decay.

5.5 Experiments

In this section we present experiments which evaluate the utility of the induced representations. We evaluate the embeddings in a cross-lingual document classification task which tests *semantic transfer* of information across languages, as well as a word-level translation task which tests fine-grained *lexical transfer*.

5.5.1 Qualitative results

Two popular ways of inspecting the learned embeddings is by looking at the nearest neighbours of words in the embedded space, and by projecting the embeddings down to 2-dimensions and plotting them. In this section, we will use these two methods to look at the relationships our model learns versus another purely bag-of-words method.

One appealing aspect of our model is that we can change the context-size of the monolingual models to learn more syntactic or more semantic features. This makes the representations leared using our model applicable for both topical prediction tasks, as we will demonstrate in the cross-lingual document topic task we will evaluate on later, as well as for fine-grained word-level prediction tasks, as we will show later with the lexical translation task. To show this, we induced embeddings on the English-German Europarl data used both as monolingual and bilingual data, in the way used by the sentence-level bag-of-words approaches of Chandar et al. [85] and Hermann et al. [84]. We then generated the nearest neighbours for the words *January, president* and *said*, which are shown in Table 5.1 along with the nearest neighbours reported by Chandar et al., taken from their paper. We have furthermore highlighted the English neighbours which we consider (subjectively) to be less similar to the English head words. As this analysis shows, at least for the examples that are reported for their method, the nearest neighbours generated by our method are less "topically-related" and show finer-grained semantic relatedness. Nonetheless, as we will see in the quantitative evaluations, this actually seems to be an advantage for the sentence-level bag-of-words methods in the cross-lingual document classification task.

5.5.2 Cross-lingual Document Classification

We use an *exact replication* of the cross-lingual document classification (**CLDC**) setup introduced by Klementiev et al. [16] to evaluate their cross-lingual embeddings. The CLDC task setup is as follows: The goal is to classify documents in a target language using only labelled documents in a source language. Specifically, we train an averaged perceptron classifier on the labelled training data in the source language and then attempt to apply the classifier as-is to the target data (known as "direct transfer"). Documents are represented as the tf-idf-weighted sum of the embedding vectors of the words that appear in the documents.

january		**president**		**said**	
january	januar	president	präsident	said	gesagt
march	märz	*i*	präsidentin	told	sagte
october	oktober	*mr*	präsidenten	say	sehr
july	juli	presidents	herr	believe	heute
december	dezember	*thank*	ich	saying	sagen
1999	jahres	president-in-office	ratspräsident	*wish*	heutigen
june	juni	*report*	danken	*shall*	letzte
month	1999	*voted*	danken	*again*	hier
year	jahr	*colleagues*	bericht	*agree*	sagten
september	jahresende	*ladies*	kollegen	*very*	will
January	Januar	president	Führing	said	gesagt
July	Juni	politician	gewählte	say	sagte
June	1	election	Mitglied	repeat	sagten
December	Juli	leader	libyschen	*just*	bemerkt
March	Jahres	parliament	nominiert	saying	vorhin
May	Dezember	*Libyan*	durfte	heard	soeben
April	31	headed	slowenischen	*here*	worauf
November	März	representative	besaß	*earlier*	angedeutet
October	30	re-election	Präsidenten	explained	sagen
February	April	king	Wahl	did	wiederhole

Table 5.1: Nearest neighbours for the purely bag-of-words autoencoder approach of Chandar et al. [85] shown above the double lines, versus our approach shown below the double lines, to the head words shown in bold at the top. In the nearest neighbours, we have italicised the words which are semantically less related to the query words. Notice that the bag-of-words approach generates more topical clusters, whereas our approach generates more syntactic clusters.

Similar to Klementiev, we induce cross-lingual embeddings for the English-German language pair, and use the induced representations to classify a subset of the English and German sections of the Reuters RCV1/RCV2 multilingual corpora [90] as pertaining to one of four categories: `CCAT` (Corporate/Industrial), `ECAT` (Economics), `GCAT` (Government/Social), and `MCAT` (Markets).

For the classification experiments, 15,000 documents (for each language) were randomly selected from the RCV1/2 corpus, with one third (5,000) used as the test set and the remainder divided into training sets of sizes between 100 and 10,000, and a separate, held-out validation set of 1,000 documents used during the development of our models. Since our setup exactly mirrors Klementiev et al, we use the same baselines, namely: the *majority* class baseline, *glossed* (replacing words in the target document by their most frequently

Method	*en* → *de*	*de* → *en*	**Training Time (min)**
Majority Baseline	46.8	46.8	-
Glossed Baseline	65.1	68.6	-
MT Baseline	68.1	67.4	-
Klementiev et al.	77.6	71.1	14,400
Bilingual Auto-encoders (BAEs)	**91.8**	72.8	4,800
BiCVM	83.7	71.4	15
BilBOWA (this work)	86.5	**75**	**6**

Table 5.2: Classification accuracy and training times for the proposed BilBOWA method compared to Klementiev et al. [16], Bilingual Auto-encoders [85], and the BiCVM model [84], on an exact replica of the Reuters cross-lingual document classification task. These methods were all used to induce 40-dimensional embeddings using the same training data. Baseline results are from Klementiev.

aligned words in the source language), and a stronger *MT* baseline (translating target documents into the source language using an SMT system).

Results are summarised in Table 5.2. In order to make all results comparable, results for all methods reported here were obtained using the same embedding dimensionality of 40 and the same training data. We use the first 500K lines of the English-German Europarl data *both* as monolingual and parallel training data. We use a vocabulary size of 46,678 for English and 47,903 for German. Since our method is motivated as a faster version of the model proposed by Klementiev *et al.*, we note that we significantly improve upon their results, while training in 6 minutes versus the original 10 days (14,400 minutes). This yields a total factor 2,400 speedup. This demonstrates that the BilBOWA-loss (equation 5.3.5) is both a computationally-efficient and an accurate approximation of the full cross-lingual objective implemented by Klementiev (equation 5.3.1).

Next, we compare our method to the current state-of-the-art bilingual embedding methods. The current state-of-the-art on this task is 91.8 (en2de) and 72.8 (de2en) reported using the Bilingual Auto-encoder (BAE) model by [85]. Hermann et al. [84] report 83.7 and 71.4 with the BiCVM model. As shown, our model outperforms the BiCVM on both tasks, and outperforms BAEs on German to English to yield a current state-of-the-art result on that task of 75%. The runtime of our method also compares very favorably to other methods. Note that even though the BiCVM method should in principle be as fast or faster than our method, its reported training time here is slightly higher since it was trained for more iterations over the data.

Method	En→Sp P@1	Sp→En P@1	En→Sp P@5	Sp→En P@5
Edit Distance	13	18	24	27
Word Co-occurrence	30	19	20	30
Mikolov et al., 2013	33	35	**51**	52
BilBOWA (This work)	**39** (+6)	**44** (+9)	**51**	**55** (+3)

Table 5.3: Results for the translation task measured as word translation accuracy (out of 100, higher is better) evaluated on the top-1 and top-5 words as ranked by the method. Cross-lingual embeddings are induced and distance in the embedded space are used to select word translation pairs. $+x$ indicates improvement in absolute precision over the previous state-of-the-art on this task [82].

5.5.3 WMT11 Word Translation

We also evaluated the induced cross-lingual embeddings on the word translation task used by Mikolov et al. [82] using the publicly-available WMT11 data[3]. In this cross-lingual word-translation task[4], the authors extracted the 6,000 most frequent words from the WMT11 English-Spanish data, and then used the online Google Translate service to derive dictionaries by translating these source words into the target language (individually for English and Spanish). Since their method requires translation-pairs for training, they used the first 5,000 most frequent words to learn the "translation matrix", and then evaluated their method on the remaining 1,000 words used as a test set. To translate a source word, one finds its k nearest neighbours in the target language embedding space, and then evaluate the translation precision $P@k$ as the fraction of target translations that are within the top-k words returned using the specific method. Our method does not require translation-pairs for training, so we simply test on the same 1,000 test-pairs.

We use as baselines the same two methods described in [82]. *Edit Distance* ranks words based on their edit-distance. *Word Co-occurrence* is based on distributional similarity: For each word w, one first constructs a word co-occurrence vector which counts the words with which w co-occurs within a 10-word window in the corpus. The word-count vectors are then mapped from the source to the target language using the dictionary. Finally, for each test word, the word with the most similar vector in the target language is selected as its translation.

The results on the English-Spanish translation tasks are summarised in Table 5.3. We induced 40-dimensional embeddings using the English and Spanish Wikipedias and Europarl as parallel data. Our model improves on both the baselines and on *Mikolov et al.*'s method on both tasks and gives a noticeable improvement in accuracy for the $P@1$ prediction. For

[3] http://www.statmt.org/wmt11/

[4] For other work on word-selection tasks, see for instance the SemEval 2013 Tasks 10 and 11 and SemEval 2014 Task 10.

the English to Spanish translation, we improve *absolute word translation accuracy* by 6 percent. For the Spanish to English task, we improve absolute word translation accuracy by 9 percent. This indicates that our model is able to learn fine-grained translation equivalences from the monolingual data by using only the raw-text, sentence-aligned parallel data, despite the lack of word-level alignments or training dictionaries.

5.6 Related work

Klementiev et al. [16] was one of the first papers to explore the idea of cross-lingual feature learning using neural language models. The crux of their approach is to model this as a *multi-task prediction problem*, where predicting a word in each language is considered one task, and then they introduce interactions between the different tasks based on the alignment-frequency of words obtained from bilingual, word-aligned data. They achieve this by training a separate NLM for each language, and then adding a regularisation term which penalises large distances between the embeddings of translation pairs, as shown in 5.3.1. As we have mentioned in this chapter, this approach suffers severe computational problems due to the expensive softmax denominator and due to the regularisation term considering the interactions between *all words* in the two languages. Although intuitively, the translation pairs that a word correspond to is only a fraction of the total vocabulary, in practice these interaction statistics are obtained from noisy word alignments and hence evaluating this term is still expensive ($O(V_1 V_2)$ worst-case).

In a publication that appeared roughly at the same time, we also considered jointly trained NLMs for learning similar features for translation pairs [17]. In that work, we analysed the embeddings learned by jointly training two NLMs with constraints on some "pivot pairs" (translation pairs). We studied the mean distance between embeddings for all known translation pairs, given an L_2-loss (5.3.1) applied to only a subset of the translation pairs. The surprising result was that embeddings started to converge to similar representations when constraining even only 10% of the translation pairs, without affecting the negative log-likelihood of the individual models on a held-out validation set. This suggested that constraining a few *pivot pairs* during training might be enough to learn rich cross-lingual features.

Zou et al. [81] learn English-Chinese bilingual word embeddings for machine translation and cross-lingual prediction. They propose to replace the expensive softmax of the Klementiev et al. model, with a more efficient margin-loss. They propose a "Translation Equivalence" objective which is equivalent to balancing an English-Chinese version of 5.3.1 with a Chinese-English version. They show an improvement of half a BLEU point in a translation task, but marginal improvement in a named-entity recognition task. However, despite the faster language modelling objective, their approach still scales badly due

to the expensive cross-lingual objective. They report training times of roughly 17 days.

Hermann et al. [84] learn multilingual semantic representations by training purely on sentence-aligned parallel data. They compute a vector representation for each sentence (bag-of-words unigrams and bigrams) and then optimise a noise-contrastive margin between observed sentence pairs and noise sentence pairs. They show good results on cross-lingual document classification and across a number of language pairs. However this approach suffers firstly from limited available parallel data, and second from the biased nature of the parallel data that is available.

Chandar et al. [85] propose a bilingual auto-encoder model which attempts to reconstruct English bag-of-words sentence vectors from their French sentence vectors, and vice versa. They apply the learned representations to the CLDC task and achieve state-of-the-art results. Their approach suffers from the same limited training data as Hermann et al., but furthermore their purely sentence-level bag-of-words approach precludes the application of the learned features for any word-level tagging task.

Kočisky et al. [91] integrate a log-bilinear language model into the FastAlign [87] reparametrised version of the IBM version 2 model. Their model aims to jointly learn a word aligment model (as part of FastAlign) and word embeddings (as part of the LBL). Their model is primarily a word alignment model parametrised in terms of the learned distributed representations. As a side-evaluation, they apply their model to the CLDC task where they show strong results. However, training a full word-alignment model might be overkill if the goal is simply to obtain the cross-lingual embeddings.

5.7 Discussion

The BilBOWA model as introduced in this chapter utilises a sampled L_2 bag of words cross-lingual loss. From a scientific point of view, we showed that this loss efficiently approximates the expensive full crosslingual loss, translating into a significant speedup during training. From an engineering point of view, we showed that the asynchronous implementation can speed up training, and that parallel subsampling improves the accuracy of the learned features. For the former, we found that getting asynchronous training to work required clipping the updates, especially as the dimensionality of the embeddings gets larger. This had a big impact on the model training speed, and no noticeable impact on the quality of the learned embeddings. Parallel subsampling speeds up training and at the same time makes it more accurate for the frequent words. It therefore turns out to be really important both in the monolingual and parallel setting. We have motivated the reason for the crosslingual setting as helping to uncover a better approximation of the global alignment statistics from the observed local, sentence-level co-occurrences.

Despite the speedup, from a user perspective, the model is still much slower to use than

offline methods like translation matrix [82] or multilingual CCA [83, 92]. However, as we show on the translation task, BilBOWA can learn much finer-grained cross-lingual relationships than these methods. BilBOWA is also slightly slower than purely parallel methods like BiCVM and BAEs, but it has the advantage that it can train over much larger monolingual datasets. If the goal is to learn high-quality general purpose bilingual embeddings, it should always be beneficial to leverage more training data, and hence a hybrid model like BilBOWA might be a better choice than a parallel-only technique.

5.8 Conclusion

In this chapter we introduced BilBOWA, a computationally-efficient model for inducing bilingual distributed word representations directly from raw text without requiring word-alignments or dictionaries. Instead, the model directly utilises a limited amount of parallel data to learn bilingual word representations. BilBOWA combines advances in training monolingual word embeddings with a particularly efficient novel sampled cross-lingual objective. The result is that the required computations per training step scales only with the number of words in the sentences, thereby enabling efficient large-scale cross-lingual training. We evaluated BilBOWA on English-German cross-lingual document classification and achieved state-of-the-art results while reducing training time to only a few minutes, rather than several days as for most previous approaches. We also evaluated on an English-Spanish word-translation task where we improve upon the previous state of the art in relative word translation error rate.

Chapter 6

BARISTA: Simple, Task-specific Bilingual Embeddings

This chapter introduces BARISTA (*"Bilingual Adaptive Reshuffling with Individual Stochastic Alternatives"*), a simple and computationally efficient training technique for learning task-specific bilingual word embeddings using monolingual word-embedding algorithms. The method incorporates additional semantic information in the form of how words map into equivalence classes during the embedding process.

In this chapter we make use of cross-lingual *part-of-speech classes* extracted from the crowd-sourced Wiktionary, *translation classes* obtained using the online Google Translate system, as well as *semantic supersense classes* extracted from the WordNet lexical resource. The model has the advantage that it is orthogonal to the choice of embedding algorithm, does not require parallel data, and can be adapted to specific tasks by redefining the equivalence classes. We analyze the learned embeddings and show that the model incorporates the information during training by trading off learning the standard syntactic linguistic regularities that word embedding models have become known for learning, with learning to cluster according to the semantic classes. We demonstrate the utility of the approach by applying these learned embedding features to the tasks of cross-lingual part-of-speech tagging over seven language pairs, as well as cross-lingual supersense tagging.

All code can be found at https://github.com/gouwsmeister/barista. The work in this chapter was presented at the North American Chapter of the Association for Computational Linguistics (NAACL 2015).

6.1 Introduction

Standard neural language models are unsupervised models that train purely on raw text by learning word features that enable the model to predict the next word in a sequence of words. In the process, the model learns to cluster words into soft equivalence classes (words

that show similar statistical behaviour), in that words that cluster together in the embedding space are words that have a high conditional likelihood of appearing in the same contexts in the training data. While there is still some controversy whether such methods are superior to older methods, there is no doubt that continuous word representations can potentially solve some of the data sparsity problems inherent in NLP, due to its high-dimensional discrete prediction problems.

However, for some tasks we might already know the equivalence classes relevant to the prediction task at hand, and would like the models to make use of this additional information during training, to hopefully learn richer, task-specific representations. For example, it is easy to extract generalized parts of speech (POS) classes [93] from crowd-sourced annotations on Wiktionary[1] [94]. If one knows which words in English maps to which POS classes, and likewise, which words in French maps to which POS classes, one might like to leverage this additional information during training to learn richer, POS-specific representations. Likewise, albeit at a much finer level of granularity, if one knows which words in two languages are translation pairs (translation equivalence classes), one might like to incorporate this side information to learn fine-grained bilingual representations.

It has furthermore been shown that weakly supervised embeddings algorithms can lead to large improvements for tasks like sentiment analysis [95]. This notion of weak supervision guiding the learning of word representations is relevant to the work presented in this chapter, and the Barista model is designed to efficiently utilize additional semantic information about how words in its training vocabulary map into equivalence classes. The model uses a very simple way of reshuffling the training data to learn rich, **semi-supervised, task-specific bilingual embeddings** over large amounts of raw text, using only a limited amount of extra class information.

6.2 The Barista Model

We exploit the feature learning property of NLMs to learn similar representations for *words from the same semantic class* during training. Crucially, our approach only assumes a small seed of equivalences, but no parallel data. It then uses these to stochastically *reshuffle* or corrupt the source and target corpora. Specifically, as shown in Algorithm 2, we begin by shuffling the concatenation of the two corpora, $\mathcal{D} = \mathcal{D}^{en} \cup \mathcal{D}^{fr}$ (for example English and French data). We then pass over this data. For each word w in each sentence of $\mathcal{D}$, if the word is in the intial seed, we replace it by a random draw from its equivalence class with probability 0.5. In other words, half of the seed words w are replaced by words w' where $C(w) = C(w')$ (belong to the same class). For example, using translation equivalence classes, the text *build the house* could be reshuffled into *construire the house* or *build la*

[1] https://www.wiktionary.org/

maison, or some other combination of English and French words with the English word order. With POS equivalence classes, any of the words in *build the house* can be replaced with words with overlapping syntactic categories, e.g. *build the voiture*.

In other words, we require of the model to be able to accurately predict any word from the observed target word's class, given the same context. This has two attractive properties: Firstly, it forces the model to learn similar embeddings for words from the same class. And secondly, it is very fast to implement, and indeed our implementation runs roughly at the same speed as training an off-the-shelf `word2vec` model.

Algorithm 2 BARISTA training algorithm. The RESHUFFLE function stochastically replaces a word w with words from the class of words that w belongs to. We will evaluate the effect of replacing words from the context vs words in target positions in Section 6.5.2.1.

function BARISTA
  $\mathcal{D} \leftarrow \text{Shuffle}(\mathcal{D}^{en} \cup \mathcal{D}^{fr})$
  for $(w_t, h) \in \mathcal{D}$ **do**
    $\tilde{w}_t, \tilde{h} \leftarrow \text{RESHUFFLE}(w_t, h)$
    $\text{TrainStep}(\tilde{w}_t, \tilde{h})$
  end for
end function

6.3 Implementation Details

We implemented the proposed method by extending the popular `word2vec` package. Our implementation can be found at https://github.com/gouwsmeister/barista.

The code works by reading in equivalence class information prior to training. The code lazily allocates a `word2class` structure mapping words to their classes, and its inverse, a `class2word` structure mapping classes to their associated member words. Memory allocation is efficiently amortized by starting with an array structure of size one, and lazily reallocating to double its size once it fills up. Given a new word during training, it can be efficiently corrupted by mapping to its class and then mapping the class to the array containing its member words. By sampling from this array (which can be done in $O(1)$ if we know its total size), the word can be corrupted in a very efficient way, as shown below using Python pseudo code.

```
def reshuffle(word):
  C = word2class[word]
  idx = randint() % words_per_class(C)
  return class2word[C][idx]
```

6.4 Obtaining Equivalence Classes

In this work we explored three different types of equivalence classes: POS, word translations, and WordNet "supersense" tags. Das and Petrov distilled the 45 part-of-speech tags from the Penn Treebank into 12 generalized classes, such as `VERB`, `NOUN`, etc. Wiktionary is an online, crowd-sourced dictionary where users annotate words with parts of speech. This represents a rich, freely available resource which has been mined in prior work to derive freely-downloadable dictionaries of words and their parts of speech[2]. We use these derived dictionaries for extracting bilingual POS equivalence classes.

For word translations we made use of the online Google Translate (GT) system. `goslate`[3] is a very convenient Python package for accessing the GT system. It automatically supports batching large translation tasks to avoid exceeding the GT fair usage policy, among other things. We made use of this to translate a vocabulary file extracted from the training data into a bilingual dictionary of words and their translations. This can then be converted into the equivalence class format expected by our implementation, by simply assigning a unique integer identifier to each 'class' of translation pairs. Word-level translation is not a trivial task, and there are definitely some noise in the translations. However, as we will show in the evaluations, the models were able to learn fine-grained bilingual embeddings despite the noise.

6.5 Experiments

In this section we present qualitative and quantitative evaluations of the proposed BARISTA model. In the experiments we balance the source and target training corpora by subsampling from the bigger corpus. The vocabularies for all models were kept unrestricted, and usually resulted in around 1 million words per language pair. We train with a window of 4 words on either side of the target words, using linear discounting of the initial learning rate of 0.1. These parameters were set on the Spanish POS data.

6.5.1 Qualitative Evaluation

We first present a qualitative evaluation of English-German bilingual embeddings learned from the smaller Europarl corpus (roughly 50M words per language). Note that while this is parallel data, we do not exploit its parallel nature.

[2] https://code.google.com/p/wikily-supervised-pos-tagger/downloads/list
[3] https://pypi.python.org/pypi/goslate

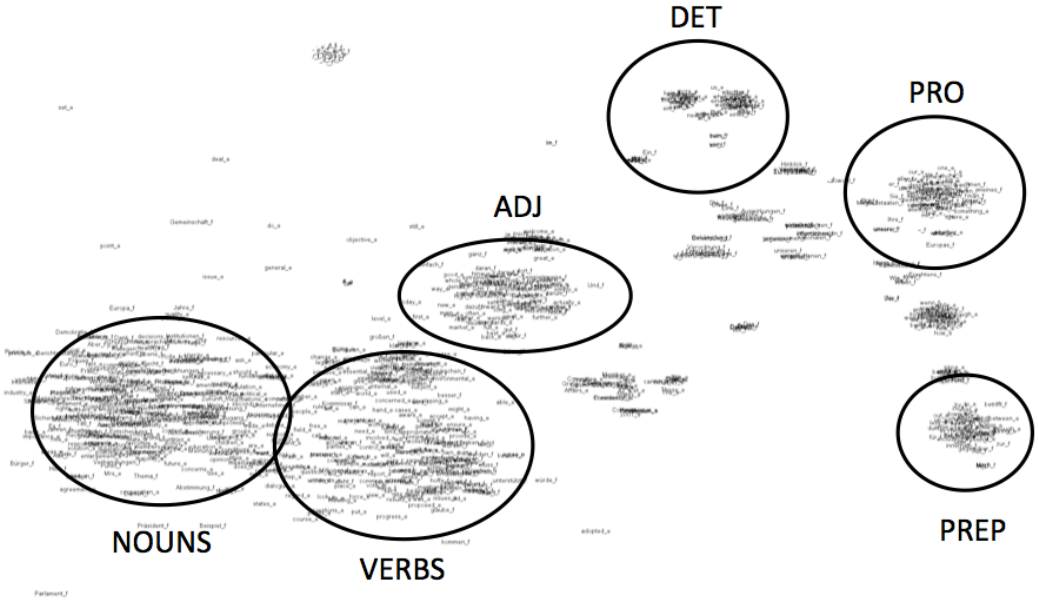


Figure 6.1: t-SNE visualization of BARISTA bilingual embeddings induced using POS classes, where we have annotated the induced clusters to highlight that the induced embeddings cluster quite distinctively by POS tag.

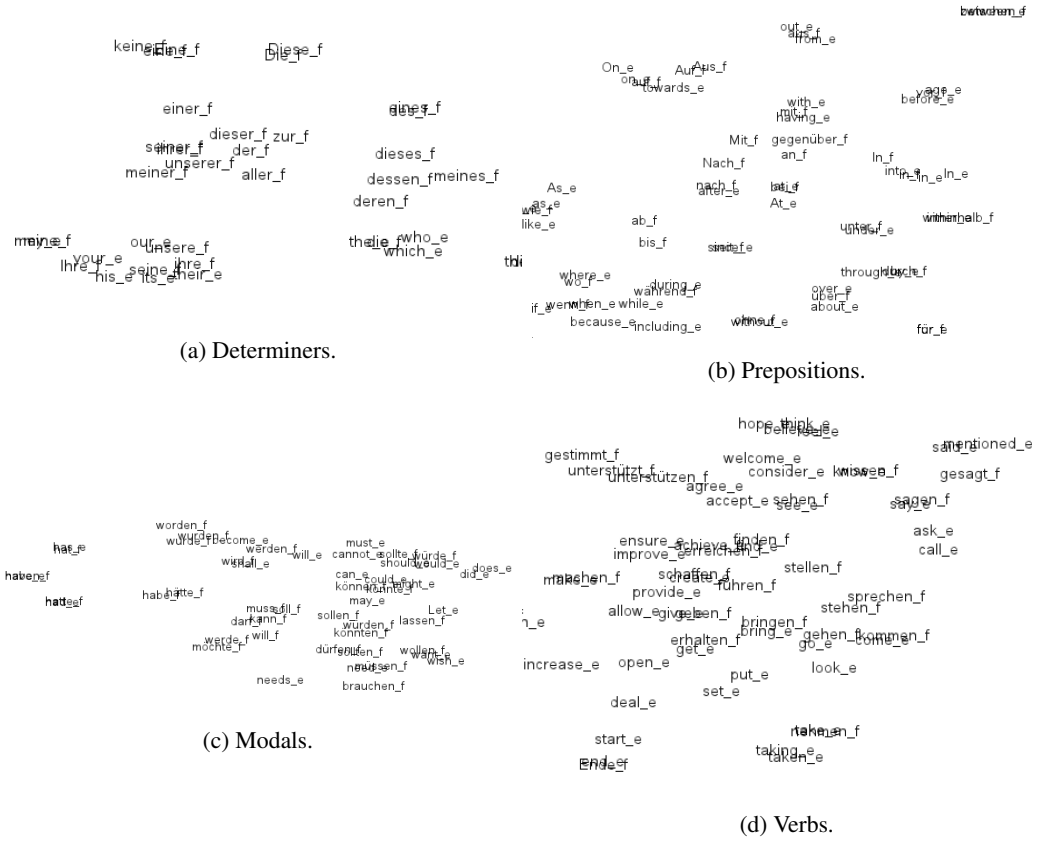
6.5.1.1 POS classes

POS classes were derived from Wiktionary as explained in Section 6.4. We train a CBOW model architecture by perturbing both the contexts and the targets. The model finished training in less than a minute on a Macbook pro.

The resulting embeddings were visualized using the t-SNE technique [88], and is shown in Fig 6.1. It is evident from the visualizations that the model learns to cluster words very distinctively by their POS tag. However, crucially, *words from the same class in both languages cluster together*. Word embedding models with short context windows typically cluster roughly by POS classes. However, in this case, we see the same effect for bilingual embeddings, i.e. words from **both** languages with the same POS tag cluster together. Individual words do not appear to retain the fine-grained relationships one normally observes in word embeddings (where similar words cluster closer together). This is to be expected, since with only 12 classes for thousands of words, we are smoothing over quite a lot of distributional information during training.

6.5.1.2 Translation classes

Next, we induced English-German bilingual embeddings on Europarl using translations obtained from Google Translate as described in Section 6.4. We derived a dictionary of the



(a) Determiners.

(b) Prepositions.

(c) Modals.

(d) Verbs.

Figure 6.2: t-SNE visualizations of English-German bilingual embeddings induced using the BARISTA model trained with translation classes derived from Google Translate. English words are appended with '_e' and German words with '_f' (for foreign). The visualizations show how the model learns very fine-grained syntactical cross-lingual relationships.

top 20 thousand most frequent words in English, translated into German. The model again finished training in less than a minute .

The embeddings are visualized using the t-SNE algorithm and are shown in Fig 6.2. The visualizations show that the models are able to extract very fine-grained cross-lingual relationships, and some clusters still seem to correspond to parts of speech. We show examples of determiners from both languages clustering together, as well as prepositions, modals and verbs.

6.5.2 Cross-lingual part-of-speech tagging

Next, we evaluated the embeddings in the context of unsupervised cross-lingual part-of-speech tagging [96]. The goal is to train a tagger, in our case a structured perceptron [97], on labeled English data, and then evaluate the model on predicting parts of speech tags in another target language. We use data from Spanish, German, Danish, Swedish, Italian,

σ	**Accuracy**
0.001	81.8
0.01	**82.4**
0.1	79.7
1.0	69.9

Table 6.1: The effect of the σ scaling parameter on the cross-lingual POS prediction task using a held out selection of English-Spanish data.

Dutch and Portuguese. Since some tagging schemes differ, training and testing data, which are the same as used by Das and Petrov [96], were converted to use the same 12 universal parts of speech proposed by Petrov et al. [93].

All results in this section were obtained by training bilingual embedding models on the publicly-available, pre-tokenized versions of Wikipedia [76], and then using the trained embeddings as features in an efficient publicly available implementation of the structured perception[4]. We use ortographic features as well as the embedding vector of the target word. In addition, we use type constraints from Wiktionary to prune the search lattice during decoding [98]. This type of prediction model is referred to as "weakly supervised" [94].

6.5.2.1 Hyperparameters and different perturbation schemes

When using embedding vectors as features for training discriminative models like the structured perceptron or CRFs, along with other binary features (e.g. capitalization), one needs to be careful to scale the individual embedding components to balance the contributions from the different feature types. This was first observed by Turian et al. [9]. We use the same scheme whereby embeddings are first *whitened* by dividing by their standard deviation, and then scaled by some hyperparameter σ, where σ is tuned on a held out validation set.

For setting hyperparameters, we chose the English-Spanish language pair[5]. It turns out that the selection of σ has a very noticeable effect on the final prediction performance on the POS task. The results are displayed in Table 6.1, where in our experiments, different values of σ can cause results to vary between 81.8 and 69.9. We chose $\sigma = 0.01$ for the rest of the experiments.

The reshuffling can be applied to training any type of monolingual word embedding model, and we considered the CBOW and skipgram models, since they have been shown to learn high quality word embeddings fast. It can furthermore be applied to corrupt different parts of the input: one could potentially corrupt target words only, context words only, or all

[4] https://github.com/coastalcph/rungsted/tree/pydecode

[5] It is standard in cross-lingual methods to tune on one language pair and then report results over all pairs using the chosen hyperparameters.

Model Architecture	Target-only	Context+Target
CBOW	81.9	**82.6**
Skipgram	81.7	82.0

Table 6.2: Effect of model architecture and where the perturbation is applied on the accuracy of the induced embeddings on the English-Spanish cross-lingual POS tagging task.

Language	Baseline	Random	Klmtv	POS-50	POS-300	Tr-50	Tr-300	DP	B-K
Spanish	80.6	81.8	79.8	82.4	81	81.6	82.6	**84.2**	80.2
German	80.4	82.7	82.8	81.8	84.1	82.6	**84.8**	82.8	81.3
Danish	63	68.9	-	68.9	72.4	71.8	78.4	**83.2**	69.1
Swedish	71.6	73.7	-	75	76	75.4	77.5	**80.5**	70.1
Italian	80.1	81.3	-	82.1	80.9	82.1	80.7	**86.8**	68.1
Dutch	74.5	77.2	-	78.3	77.4	78.7	**80.3**	79.5	65.1
Portuguese	76.9	78.1	-	77.3	76.1	80.6	80.5	**87.9**	78.4
Avg	75.3	77.7	-	78	78.3	79	80.7	**83.6**	73.2

Table 6.3: Results on the cross-lingual POS-tagging task. **Klmtv** uses the bilingual embeddings from Klementiev [16], DP is the approach by Das and Petrov [96], **POS-X** refer to X-dimensional BARISTA embeddings trained with part-of-speech equivalence classes, and likewise **Tr-X** is X-dimensional BARISTA embeddings trained using translation classes.

words in the training data. We evaluated the effect of these two choices on the cross-lingual POS-tagging problem for the English to Spanish prediction task. The results are shown in Table 6.2, which shows that the CBOW architecture gave the best results on the held out set when we reshuffle both targets and contexts. This is the setting that was used for the rest of the experiments.

6.5.2.2 Results

Our baseline method is a type-constrained structured perceptron with only ortographic features, which are expected to transfer across languages. We also experiment with using *random* embeddings, as well as off-the-shelf bilingual embeddings provided by Klementiev [16] (Klmtv), available online[6]. POS-X refer to X-dimensional BARISTA embeddings trained with part-of-speech equivalence classes, and Tr-X are X-dimensional BARISTA embeddings trained with translation classes obtained from Google Translate. As an upper bound, we compare to the results reported by Das and Petrov (DP), since their method requires more additional insample unlabeled data as well as parallel bilingual data. Our results are displayed in Table 6.3. All results were obtained by running our implementation of

[6] http://people.mmci.uni-saarland.de/~aklement/data/distrib.

the BARISTA model on a standard Macbook Pro, and inducing embeddings and training the tagger for one language-pair took between 15 and 45 minutes for 50 and 300-dimensional embeddings respectively.

We notice that *training with random embeddings improves over the baseline*, implying that the random features are acting as a regularizer, which enables the baseline system to generalize better. This might seem strange, but can be understood as follows: The regularized objective of the averaged perceptron is to minimize LOSS + NORM. Let M_1 be the baseline model, and M_2 baseline plus random features. In theory, $\text{LOSS}_{M_1} = \text{LOSS}_{M_2}$, but $\text{NORM}_{M_1} < \text{NORM}_{M_2}$ (simply because the norm increases with many random features). This forces $\text{LOSS}_{M_1} < \text{LOSS}_{M_2}$, since the new weight is on random features, not original features, and loss increases (subject to noise injection). A greater loss is likely to lead to underfitting and hence have a regularizing effect. Also, we note that the bilingual embeddings provided by Klementiev for the English-Spanish and English-German tasks actually degrades performance for Spanish and only shows a very slight gain for German.

Overall, two observations can be made: First, training the BARISTA model with POS classes improves over the random baseline, but training with translation classes gives even better performance. For both approaches, using more embedding features improves the performance (although increasing the dimensionality to 500 did not improve results, presumably since there was not enough training data or due to lack of regularization on the embeddings during training).

The best model trained with translation classes with 300-dimensional embeddings is able to improve over the unsupervised Berg-Kirkpatrick (B-K) baseline for every language-pair, and over the Das and Petrov results for the English-German and English-Dutch language pairs.

6.5.3 Cross-lingual super sense tagging

Finally, we also used the BARISTA-embeddings for English-Danish (with parameters still set on Spanish POS) on *another* task, namely super sense tagging [99]. We train a system on a mixture of 1000 randomly sampled sentences from English SemCor[7] and 320 labeled Danish sentences. We compare using 300-dimensional bilingual embeddings trained with equivalence classes from English and Danish WordNets (WN-300), to embeddings trained using translation equivalence classes (Tr-300). We use a most-frequent-sense baseline (**MFS**), as well as structured perceptron model trained only with ortographic features. The evaluation metric is a weighted average over F_1-scores for the 41 semantic classes. Results are shown in Table 6.4. Note that BARISTA embeddings induced using the WordNet knowledge base is superior to using translation equivalences, but both embeddings are superior to both our baselines. In fact, using the BARISTA embeddings lead to a 21% improvement

[7] http://web.eecs.umich.edu/~mihalcea/downloads.html#semcor

	Baselines		Barista	
	MFS	bl	Tr-300	WN-300
blogs	49.1	46.7	50.5	**61.9**
forum	44.5	41.2	45.0	**53.9**
magazine	46.5	45.2	50.4	**51.5**
newswire	48.4	45.4	52.7	**60.9**
reviews	48.4	44.9	50.4	**55.8**
speech	51.1	48.4	51.5	**58.4**
Avg	48.1	45.4	50.5	**58.3**

Table 6.4: Cross-language super sense tagging

in accuracy. The Danish training (newswire only) and test data (six different domains) – is made publicly available with the Barista code.

6.6 Conclusion

This chapter introduced Barista, a very simple and computationally efficient data corruption technique for learning task-specific bilingual word embeddings using monolingual word-embedding algorithms. Barista leverages additional semantic information about how words map into equivalence classes during the training of word embedding models. We evaluated using part-of-speech classes derived from the crowd-sourced Wiktionary, translation classes obtained using the online Google Translate system and supersense semantic categories extracted from WordNet. We presented qualitative results showing that the proposed technique efficiently trades off learning normal distributional relationships with learning to cluster the induced embeddings by their class. Coarse POS classes preserve POS information but tend to lose a lot of fine-grained syntactic information. Translation classes enable the models to efficiently learn to cluster translation-pairs across languages. We presented results on cross-lingual part-of-speech tagging over seven languages and showed that the proposed method achieves strong results overall, and is able to outperform off-the-shelf bilingual embeddings as well as two strong baseline methods on the English-German and English-Dutch language pairs. We presented results on a cross-lingual English-Danish supersense tagging tasks, where Barista embeddings induced using WordNet classes leads to a 21.2% overall relative improvement in prediction accuracy. The method is highly scalable and we make our implementation available at https://www.github.com/gouwsmeister/barista.

Chapter 7

Conclusion

In the following sections we provide a short synopsis of the work in this dissertation as well as a summary of the main research findings.

7.1 Synopsis

This dissertation studies the problem of automatically *learning* to represent one or more natural languages from raw text data in order to successfully predict *different* aspects of their meaning by computational means. Chapter 3 provides an in-depth overview of neural language models (NLMs), with a particular emphasis on NLMs for learning word embeddings. It also presents a **novel geometrical perspective** for understanding how word embedding models are trained, by viewing the gradients of the objective function as exerting push-pull forces on the learned embedding point-vectors during training. Chapter 4 introduces **SimTree**, an efficient $O(V \log V)$ algorithm for jointly learning a likelihood-optimal tree structure during training of an NLM with a hierarchical softmax output layer. Section 4.3 introduces a **simple subsampling training technique** for improving both the speed and the quality of embeddings induced using the hierarchical softmax. Chapter 5 introduces **BilBOWA**, an efficient bilingual word embedding model which learns representations for words across two or more languages by only making use of a limited sample of parallel raw text, and unlimited amounts of monolingual raw text. Finally, Chapter 6 introduces **Barista**, a semi-supervised bilingual word embedding model which can make use of additional semantic information in the form of word equivalence classes during the embedding process. Throughout, this dissertation presents qualitative evaluations of the embeddings induced using the different methods through visualisations and inspection of nearest neighbours in the learned space, as well as quantitative experiments on the monolingual word analogical reasoning task, and cross-lingual document classification, part-of-speech tagging, supersense tagging and word translation.

7.2 Summary of Findings

This dissertation investigated the primary hypothesis that neural word embeddings induced from raw text can *jointly* represent two or more languages in order to accurately predict *different* aspects of their meaning through computational means. Through the course of the dissertation, we evaluated this hypothesis by proposing and evaluating different word embedding models on a variety of tasks and languages.

A Geometrical View of Word Embedding Models We started with a careful review of the state-of-the art in monolingual word embedding algorithms in Chapter 3. We identified the key components of current monolingual word embedding models: a) how the model selects context words, b) how the network maps context words to predict target words, and c) how it is trained.

Word embeddings define a geometrical manifold structure in the induced embedding space. By studying the most popular loss functions for training word embedding models, we introduced a common geometrical framework for understanding how word embedding models operate during training that directly applies to shallow word2vec-type models and provides a useful intuitive insight into the working of nonlinear NLMs (§ 3.5): The loss function can be understood as assigning a cost to the geometrical structure of the manifold spanned by the learned embeddings with respect to a new training pair. The loss function can further be decomposed into positive pulling forces acting on the learned word embeddings that are balanced with negative pushing forces. The goal of training is to find an equilibrium geometrical configuration that minimises the total *expected* cost of the manifold structure under the data generating distribution.

The result of training is that the distances from a word's learned embedding vector to the vectors learned for other words are proportional to the frequency with which they are observed together in the training data. This represents a rich geometrical encoding of the co-occurrence statistics of natural languages. The Distributional Hypothesis (§ 3.2) states that a word's meaning is reflected in its co-occurrence statistics, and in this work we relied on an extension of this notion to more than one language by jointly training bilingual word representations. The bilingual experimental results we will be summarising below suggest that these models indeed succeed in extracting useful information from the distributional statistics that are useful not only for predicting linguistic attributes in one language, but also across different languages.

Learning Tree Structures for Hierarchical Output Layers The hierarchical softmax speeds up training of NLMs by grouping words into classes, and structuring the calculation of $P(w|h)$ according to the class tree structure. Different trees trade off prediction accuracy and training speed. To optimise for prediction speed, the optimal tree structure corresponds to the Huffman encoding, where each word's depth in the tree is inversely proptional to its frequency of occurrence in the data (i.e. frequent words have short paths). To optimise for

likelihood, words should cluster in the trees in order to balance their empirical conditional likelhoods in the data. Therefore, for a fixed context, words that are more probable should cluster together in the tree. However, nodes are shared between words, which creates a dependency between tree structure and node parameters. Since tree structure is unobserved, learning the best tree structure requires solving a latent-variable problem.

We introduced the SimTree algorithm (§ 4.2), an efficient $O(V \log V)$ algorithm for learning an approximate likelihood-optimal tree structure. SimTree uses an alternating maximisation approach which iterates between learning optimal model parameters given a fixed tree, and learning an improved tree structure given fixed model parameters. The context-dependent nature of the tree structure suggests that there might be many possible approximately-optimal tree structures. SimTree averages over these by keeping a running average of each word's empirical context-vectors, and reclustering according to these. We showed that the algorithm is very efficient to run in practice, learns interesting semantic clusters, and improves the likelihood of the model in a language modelling task.

Bilingual Word Embeddings using Limited Sentence-aligned Data

BilBOWA (Chapter 5) is a computationally-efficient model for inducing general-purpose bilingual distributed word representations directly from raw text without requiring word-level alignments or dictionaries. Instead, the model trains directly on a limited sample of parallel data and unlimited amounts of monolingual data. The model combines advances in training monolingual word embeddings with a novel, efficient, sampled cross-lingual objective. In the process, the model trades off learning general linguistic features with learning cross-lingual relationships.

BilBOWA was motivated as a faster approximation of the bilingual embedding model proposed by Klementiev et al. [16]. The model was evaluated on a semantic cross-lingual English-German document-topic classification task, in the same setup as Klementiev [16], as well as in a lexically fine-grained word translation task, in the same setup as the offline Translation Matrix approach by Mikolov [82]. Our experimental results show that BilBOWA is able to significantly improve upon the results of the original model by Klementiev, despite a computational speedup of three orders of magnitude. This suggests that the Monte Carlo approximation employed by the BilBOWA model is indeed an effective approximation of the original expensive cross-lingual regulariser employed by Klementiev [16].

In the translation task, we trained bilingual BilBOWA embeddings and used distance in the induced vector space to select Spanish translation candidates for a selection of English words. Using the jointly-trained bilingual BilBOWA embeddings on this task shows significant improvements over embeddings aligned after being trained in a disjoint monolingual setting, using the Translation Matrix method. This suggests that the *joint nature* of the optimisation employed by the BilBOWA model is able to discover finer-grained lexi-

cal correspondences, even without using any word-level information, as was used by the previous state-of-the-art Translation Matrix method.

Bilingual Task-specific Word Embeddings BARISTA (Chapter 6) is a very simple wrapper technique for inducing task-specific bilingual word embeddings using only monolingual data and a small seed of extra semantic information. The algorithm exploits a prior semantic mapping of how words in the source and target languages map into near-equivalence classes (such as POS), and includes this information during the embedding process. We showed qualitatively that the model trades off learning linguistic features with learning to cluster by the prior semantic class information. We also showed experiments on cross-lingual part-of-speech tagging using POS classes and word translation classes, as well as cross-lingual supersense (SS) tagging using classes extracted from WordNet. In both cases, the bilingual BARISTA embeddings were able to improve upon off-the-shelf bilingual embeddings, and for English-Danish supersense tagging, led to a 10% improvement in the absolute accuracy of the model.

Interestingly, for the POS task, bilingual embeddings induced using translation classes outperformed embeddings induced using POS classes. However, for SS-tagging, WordNet classes substantially outperformed translation classes. We can only conjecture that this is perhaps related to the granularity of the classes: with 12 POS classes, the model is smoothing over a lot of distributional information during the embedding process, compared to 41 SS classes, or the roughly 20 thousand translation "classes". Evidently, for POS tagging, the richer syntactic information retained using the finer-grained translation classes represents useful information for the POS-tagging task.

7.3 Future directions

Since SimTree is designed to optimize expected model likelihood, it often generates trees that are very unbalanced and hence slow to use in practice. It might be a good idea to combine the likelihood objective with a speed objective (as was done in Zweig and Makarychev [57] for a shallow tree) to learn tree structures that are both fast to evaluate and lead to good model likelihoods.

The BilBOWA-loss was originally motivated as an approximation to an all-pairs distance loss. However, combining the Euclidian distance with the monolingual losses based on the dot-product can lead to scaling issues. The interaction between embedding norms and the distance function used needs to be studied in more detail. It may be better to define the BilBOWA-loss in terms of the cosine distance between word pairs, or alternatively to impose a norm-constraint on embeddings during training.

7.4 Conclusion

During the course of this dissertation, we have shown improvements in cross-lingual document-topic prediction, cross-lingual part-of-speech tagging, cross-lingual supersense tagging and word-level translation by primarily utilising bilingual word embeddings learned from raw text as the predictive features. Together, these results provide overwhelming evidence in favour of our primary research hypothesis (§ 1.2). We therefore conclude that distributed neural word embeddings induced from raw text offer a suitable and efficient vehicle for *jointly* representing two or more languages in order to accurately predict various different aspects related to their meaning.

Bibliography

[1] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

[2] J. Allen, "Natural language understanding," 1987.

[3] T. Winograd, "Understanding natural language," *Cognitive psychology*, vol. 3, no. 1, pp. 1–191, 1972.

[4] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: The MIT Press, 1999. [Online]. Available: http://nlp.stanford.edu/fsnlp/

[5] N. A. Smith, *Linguistic Structure Prediction*, ser. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May 2011.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

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

[9] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: A simple and general method for semi-supervised learning," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 384–394.

[10] M. Bansal, K. Gimpel, and K. Livescu, "Tailoring continuous word representations for dependency parsing," in *ACL*, 2014.

[11] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[12] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Journal of Machine Learning Research*, 2003.

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

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

[15] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations." in *HLT-NAACL*, 2013, pp. 746–751.

[16] A. Klementiev, I. Titov, and B. Bhattarai, "Inducing crosslingual distributed representations of words," in *Proceedings of the International Conference on Computational Linguistics (COLING)*, Bombay, India, December 2012.

[17] S. Gouws, G. van Rooyen, and Y. Bengio, "Learning structural correspondences across different linguistic domains with synchronous neural language models," in *Workshop on Crosslingual Technologies (xLite) at the Neural Information Processing Conference*, 2012.

[18] T. Cover, J. Thomas, J. Wiley *et al.*, *Elements of information theory*. Wiley Online Library, 1991, vol. 6.

[19] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[20] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006, vol. 1.

[21] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[22] S. Marsland, *Machine Learning: An Algorithmic Perspective*. CRC Press, 2009.

[23] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[24] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[25] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[26] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation. [Online]. Available: http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf

[27] P. Werbos, "Beyond regression: new fools for prediction and analysis in the behavioral sciences," *PhD thesis, Harvard University*, 1974.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699.

[29] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[30] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[31] D. Gildea and D. Jurafsky, "Automatic labeling of semantic roles," *Computational Linguistics*, vol. 28, no. 3, pp. 245–288, 2002.

[32] P. Brown, P. Desouza, R. Mercer, V. Pietra, and J. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[33] P. Liang, "Semi-supervised learning for natural language," Master's thesis, Massachusetts Institute of Technology, 2005.

[34] S. Miller, J. Guinness, and A. Zamanian, "Name tagging with word clusters and discriminative training," in *Proceedings of HLT*, 2004, pp. 337–342.

[35] L. Ratinov and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2009, pp. 147–155.

[36] M. Candito and B. Crabbé, "Improving generative statistical parsing with semi-supervised word clustering," in *Proceedings of the 11th International Conference on Parsing Technologies*. Association for Computational Linguistics, 2009, pp. 138–141.

[37] T. Koo, X. Carreras, and M. Collins, "Simple semi-supervised dependency parsing," vol. Association for Computational Linguistics (ACL), 2008.

[38] J. Suzuki, H. Isozaki, X. Carreras, and M. Collins, "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*. Association for Computational Linguistics, 2009, pp. 551–560.

[39] H. Zhao, W. Chen, C. Kit, and G. Zhou, "Multilingual dependency learning: a huge feature engineering method to semantic dependency parsing," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, 2009, pp. 55–60.

[40] P. Dhillon, D. P. Foster, and L. H. Ungar, "Multi-view learning of word embeddings via cca," in *NIPS*, 2011, pp. 199–207.

[41] S. Cohen, M. Collins, D. Foster, K. Stratos, and L. Ungar, "Spectral learning algorithms for natural language processing," Nov 2013. [Online]. Available: http://www.cs.columbia.edu/~scohen/naacl13tutorial/

[42] J. Goodman, "A bit of progress in language modeling," *Computer Speech & Language*, vol. 15, no. 4, pp. 403–434, 2001.

[43] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[44] G. E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the eighth annual conference of the cognitive science society*, vol. 1. Amherst, MA, 1986, p. 12.

[45] J. R. Firth, *A Synopsis of Linguistic Theory 1930–55*. Oxford, 1957.

[46] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[47] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[48] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[49] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Proceedings of NIPS, 2000*, 2001.

[50] Y. Goldberg and O. Levy, "word2vec explained: Deriving mikolov et al.Õs negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[51] A. Mnih and G. Hinton, "Three new graphical models for statistical language modelling," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 641–648.

[52] Y. LeCun, S. Chopra, and R. Hadsell, "A tutorial on energy-based learning," *To appear in ÒPredicting Structured Data*, vol. 1, p. 0, 2006.

[53] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *AISTATSÕ05*, 2005, pp. 246–252.

[54] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[55] A. Mnih and G. E. Hinton, "A scalable hierarchical distributed language model," in *Advances in neural information processing systems*, 2009, pp. 1081–1088.

[56] Y. Bengio and J.-S. Sénécal, "Adaptive importance sampling to accelerate training of a neural probabilistic language model," *IEEE Transactions on Neural Networks*, vol. 19, no. 4, pp. 713–722, 2008.

[57] G. Zweig and K. Makarychev, "Speed regularization and optimality in word classing," in *ICASSP'13*, 2013, pp. 8237–8241.

[58] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[59] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[60] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *arXiv preprint arXiv:1211.5063*, 2012.

[61] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, University of Toronto, 2013.

[62] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *INTERSPEECH*, 2010, pp. 1045–1048.

[63] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5528–5531.

[64] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[65] A. Mnih and Y. W. Teh, "A fast and simple algorithm for training neural probabilistic language models," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[66] O. Levy and Y. Goldberg, "Dependency-based word embeddings," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, vol. 2, 2014.

[67] G. E. Dahl, R. P. Adams, and H. Larochelle, "Training restricted boltzmann machines on word observations," in *International Conference on Machine Learning*. Omnipress, 2012, pp. 679–686.

[68] Y. Bengio and J.-S. Senécal, "Quick training of probabilistic neural nets by importance sampling," in *AISTATS Conference*, 2003.

[69] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 297–304.

[70] M. U. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 307–361, 2012.

[71] R. Rosenfeld, "A whole sentence maximum entropy language model," in *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, Dec. 1997, pp. 230–237.

[72] A. Mnih and Y. W. Teh, "Learning label trees for probabilistic modelling of implicit feedback," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2816–2824.

[73] A. E. Choromanska and J. Langford, "Logarithmic time online multiclass prediction," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 55–63.

[74] D. A. Huffman *et al.*, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[75] C. A. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 1962.

[76] R. Al-Rfou', B. Perozzi, and S. Skiena, "Polyglot: Distributed word representations for multilingual nlp," in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association for Computational Linguistics, August 2013, pp. 183–192. [Online]. Available: http://www.aclweb.org/anthology/W13-3520

[77] J. Blitzer, R. McDonald, and F. Pereira, "Domain adaptation with structural correspondence learning," in *Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, 2006.

[78] H. Daumé III, "Frustratingly easy domain adaptation," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009.

[79] S. J. Pan and Q. Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 1345–1359, 2010.

[80] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, vol. 12, 2014.

[81] W. Y. Zou, R. Socher, D. Cer, and C. D. Manning, "Bilingual word embeddings for phrase-based machine translation," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[82] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," in *International Conference on Learning Representations (ICLR)*, 2013.

[83] M. Faruqui and C. Dyer, "Improving vector space word representations using multilingual correlation," in *Proceedings of EACL 2014*, 2014.

[84] K. M. Hermann and P. Blunsom, "Multilingual distributed representations without word alignment," *arXiv preprint arXiv:1312.6173*, 2013.

[85] S. Chandar, S. Lauly, H. Larochelle, M. M. Khapra, B. Ravidran, V. Raykar, and A. Saha, "An autoencoder approach to learning bilingual word representations," *Proceedings of NIPS 2014*, 2014.

[86] F. J. Och and H. Ney, "A systematic comparison of various statistical alignment models," *Computational Linguistics*, vol. 29, no. 1, pp. 19–51, 2003.

[87] C. Dyer, V. Chahuneau, and N. A. Smith, "A simple, fast, and effective reparameterization of ibm model 2." ACL, 2013.

[88] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008.

[89] P. Koehn, "Europarl: A parallel corpus for statistical machine translation," in *MT summit*, vol. 5, 2005, pp. 79–86.

[90] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *The Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[91] T. Kočiskỳ, K. M. Hermann, and P. Blunsom, "Learning bilingual word representations by marginalizing alignments," *arXiv preprint arXiv:1405.0947*, 2014.

[92] A. Haghighi, P. Liang, T. Berg-Kirkpatrick, and D. Klein, "Learning bilingual lexicons from monolingual corpora," in *ACL*, vol. 2008, 2008, pp. 771–779.

[93] S. Petrov, D. Das, and R. McDonald, "A universal part-of-speech tagset," in *LREC*, 2011.

[94] S. Li, J. Graa, and B. Taskar, "Wiki-ly supervised part-of-speech tagging," in *EMNLP-CoNLL'12*, 2012, pp. 1389–1398.

[95] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, "Learning sentiment-specific word embedding for Twitter sentiment classification," in *ACL*, 2014.

[96] D. Das and S. Petrov, "Unsupervised part-of-speech tagging with bilingual graph-based projections," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 600–609.

[97] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *EMNLP*, 2002.

[98] O. Täckström, D. Das, S. Petrov, R. McDonald, and J. Nivre, "Token and type constraints for cross-lingual part-of-speech tagging," *TACL*, vol. 1, pp. 1–12, 2013.

[99] M. Ciaramita and Y. Altun, "Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger," in *Proc. of EMNLP*, Sydney, Australia, Jul. 2006, pp. 594–602.