Enhancing Word Representation Learning with Linguistic Knowledge

Diego Ramírez Echavarría

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Information Studies

University College London

October 4, 2022

I, Diego Ramírez Echavarría, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Representation learning, the process whereby representations are modelled from data, has recently become a central part of Natural Language Processing (NLP). Among the most widely used learned representations are word embeddings trained on large corpora of unannotated text, where the learned embeddings are treated as general representations that can be used across multiple NLP tasks. Despite their empirical successes, word embeddings learned entirely from data can only capture patterns of language usage from the particular linguistic domain of the training data. Linguistic knowledge, which does not vary among linguistic domains, can potentially be used to address this limitation. The vast sources of linguistic knowledge that are readily available nowadays can help train more general word embeddings (i.e. less affected by distance between linguistic domains) by providing them with such information as semantic relations, syntactic structure, word morphology, etc.

In this research, I investigate the different ways in which word embedding models capture and encode words' semantic and contextual information. To this end, I propose two approaches to integrate linguistic knowledge into the statistical learning of word embeddings. The first approach is based on augmenting the training data for a well-known Skip-gram word embedding model, where synonym information is extracted from a lexical knowledge base and incorporated into the training data in the form of additional training examples. This data augmentation approach seeks to enforce synonym relations in the learned embeddings. The second approach exploits structural information in text by transforming every sentence in the data into its corresponding dependency parse trees and training an autoencoder to recover the original sentence. While learning a mapping from a dependency parse tree to its originating sentence, this novel Structure-to-Sequence (Struct2Seq) model produces word embeddings that contain information about a word's structural context. Given that the combination of knowledge and statistical methods can often be unpredictable, a central focus of this thesis is on understanding the effects of incorporating linguistic knowledge into word representation learning. Through the use of intrinsic (geometric characteristics) and extrinsic (performance on downstream tasks) evaluation metrics, I aim to measure the specific influence that the injected knowledge can have on different aspects of the informational composition of word embeddings.

Impact Statement

NLP, the branch of *Artificial Intelligence (AI)* that deals with the automated interaction with human language, has seen an exponential growth in applications in the past years. At the heart of many contemporary NLP applications lie word embeddings, which are learned vectorial representations of words that encode linguistic information captured from training data.

The current trend in text representation is to use increasingly complex models trained on vast amounts of data (e.g. the widely used Bidirectional Encoder Representations from Transformers (BERT) model Devlin et al. (2018) contains billions of parameters trained on a dataset of several billion words). Nevertheless, constructing alternative models that can approximate the performance of these massive models at a fraction of the computational and data costs can have many potential advantages. Lightweight word embedding models can facilitate porting state-of-the-art NLP models to **low-resource languages**, i.e. languages for which large amounts of data is not readily available. Additionally, lowering the data requirement of these models gives more control over the selection of training data, where a more carefully selected training dataset can produce **specialised embeddings** for niche language domains where general-purpose models might perform poorly. Smaller and better understood datasets can also reduce the **bias** that word embedding capture from their training data, since these embeddings can reinforce negative stereotypes in the applications they are used for (for a more detailed discussion on this topic refer to Bolukbasi et al. (2016b)).

Getting a better understanding of the inner workings and theoretical and empirical limitations of word embeddings can help harness these models to create more inclusive NLP applications. This research aims to provide some of the necessary stepping stones towards that goal. In the longer term, this research has the potential to transform a wide range of NLP applications that have become ubiquitous in everyday life, such as machine translation (e.g. DeepL Translator, https://translate.google.com/) to grammar checkers (e.g. https://www.grammarly.com/, https://mail.google.com), search engines, or fake news detectors.

Acknowledgements

I am convinced I had the best supervisory team a PhD student could ever hope for. Luke Dickens helped me find my way through the complexities of machine learning and was always excited to listen to my ideas and take them for a walk. Rob Miller was instrumental in not letting these walks get too far, without his experienced advice (and his reassurance whenever my imposter syndrome kicked in) this thesis would have never been completed. Antonis Bikakis taught me how to think like a researcher, he gave me incredibly precise comments and consistently pointed out the angle I was missing. Andreas Vlachidis was genuinely interested in my approach to Natural Language Processing and our conversations on rule-based methods and linguistic theory were invaluable for my research. The highlights of this research came into existence over a cup of perfectly brewed coffee in a crammed office and in the midst of tangents, jokes, and anecdotes.

My London family made PhD life a real pleasure. My fellow inmates of G33 created the best possible work environment. Whenever work started to threaten my sanity I could count on a "babka lap" with Marco or a deep conversation about Python or the housing struggles of East London with Hannah. The biweekly visits to my adoptive PhD family at the SPIKE seminars at Imperial College always kept things interesting. The conversations with Ruben, Osama, Nuri, and the rest of the SPIKE group contributed to making my reading list unrealistic.

After work, my friends made London feel like a true home. Hot pot parties with Star, Marty, Jackie, and Bambino were the perfect place to disconnect from work, until Xiaoliang dragged me into an hour-long conversation on neural networks. Pizza and videogame nights with Michele, Ellen, Fabio, and Xiaoliang were an essential part of my PhD. I discovered the best views of the city during my Sundays in Regent's Canal atop Karen's boat. I also found a little piece of Latin America in London thanks to María, Inés, Andrea, Luis, and Morteza (Iranian, but an honorary Latin American).

On the other side of the Atlantic, I can never stop thanking Alexandra for putting up with me during the last stint of my thesis write-up. Koa and her kept me going through some of the hardest moments of the PhD. The support from my family and friends in Mexico has never waned. My parents, Celina and Eduardo, never stopped asking about my work, even when they didn't understand it. My siblings, Fernanda, Esteban, and María, shared my excitements and my frustrations and they patiently heard me say "I'm almost done with my thesis" hundreds times. My friends in Mexico always gave me a reason to miss home and to enjoy being back.

I want to express my gratitude to the Mexican National Council for Science and Technology (CONACYT) for funding my PhD project. I would also like to acknowledge the use of the UCL Myriad High Throughput Computing Facility (Myriad@UCL), and associated support services, in the completion of this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is quite an illusion to imagine that one adjusts to reality essentially without the use of language and that language is merely an incidental means of solving specific problems of communication or reflection. The fact of the matter is that the 'real world' is to a large extent unconsciously built upon the language habits of the group.

Edward Sapir, *The Status of Linguistics as a Science*

The volume of text data available today has far exceeded the capacity for manual human analysis. This has increased the need for computational methods that can process text data efficiently. The first step, and possibly the most crucial in the computational processing of language, is generating computer-actionable representations of text that are rich in syntactic and semantic information.

Representation learning, as defined by Bengio et al. (2013), is the field of machine learning that deals with explicitly designing how representations are modelled from data. The driving idea behind representation learning is that, by focusing on specific aspects of the data, a model can learn general representations that can be used across different tasks. Learned representations of words have recently become central to machine learning-based NLP (Young et al., 2018), where they have enabled models to exploit linguistic information from large text corpora to solve a wide array of different NLP tasks. Despite their widespread use, the exact manner in which these representation models capture and embed the information from the training data remains poorly understood (Gladkova and Drozd, 2016; Tsvetkov et al., 2015; Schnabel et al., 2015).

Learned representations of words typically obtain the totality of their information from training data. Many of the most common word representation models are based on some form of language model, where word representations are learned by observing the patterns of occurrence of a word within its *context*, where the concept of context can be defined and modelled in different ways, e.g. context in Word2Vec is defined as the words that appear within a fixed window around a centre word. These representations reportedly capture subtleties in the use of language that might escape even the most meticulous human analysis. However, in order to identify commonalities across different language usages, these representation models require large training datasets that are representative of a diversity of linguistic domains.

This thesis explores how the shortcomings of word representation learning models can be addressed by more closely emulating language acquisition in humans, where experiential learning is combined with pre-existing linguistic rules. Incorporating different sources of linguistic knowledge into the statistical representation learning process can potentially reduce the high data requirements of representation learning models while also producing word representations that are more invariant across linguistic domains. In this research I investigate two approaches that integrate linguistic knowledge into the unsupervised training of neural network-based word embeddings: the augmentation of training data with linguistic relations extracted from lexicons, and a training objective based on reconstructing grammatical structure. The aim of this research is to investigate how linguistic knowledge can be exploited to learn higher-quality word representations with smaller training datasets. To this end, I explore different approaches to inject linguistic knowledge into the statistical learning of word representations while preserving the distributional information of word occurrence in natural language. As argued in chapter 2, reproducibility remains a highly desirable, but not always attained, characteristic in NLP research. The research in this thesis has been continually shaped by transparent and reproducible experiments. In order to isolate and more clearly trace the effect of the injected linguistic knowledge in the learning process, this research places a high emphasis on tracking all experimental conditions by following a framework I designed as a model for experimental reproducibility. To test out the practical applications of this research, I propose new word representation models that can accommodate an ever-growing diversity of language uses.

1.1 Relevance of Word Embeddings

In recent years, large scale Transformer-based language model such as BERT (Devlin et al., 2018) have started displacing word embedding models like Word2Vec (Mikolov et al., 2013b) and Global Vectors (GloVe) (Pennington et al., 2014) as the preferred text representation technique for many downstream applications. However, as evidenced by the most recent instances of some of the main NLP conferences, word embeddings remain a very active research area, see for instance Zheng et al. (2022); Poumay and Ittoo (2021); Marchisio et al. (2021); Templeton (2021). Word embeddings are also used in commercial NLP applications and libraries such as the spaCy library (Honnibal and Montani, 2017), which, among other things, uses word embeddings as the foundation of its text similarity functionality. Word embeddings can provide an alternative when working with limited computational resources (Zheng et al., 2022; Poumay and Ittoo, 2021) or low-resource languages (Templeton, 2021).

1.2 Contributions

The main contributions of this research are:

- A **comprehensive analysis** of principles and features captured by different word representation models
- An automated pipeline to construct **feature-engineered word representations** from multiple sources of linguistic information
- A learning algorithm that incorporates external knowledge to enhance word embeddings by **augmenting the training data**

- A **partitioned word embedding** model that constrains information flow to different subspaces of the word vector
- A novel **Struct2Seq model** that trains semantic structure-aware word embeddings based on a dependency parse tree-to-text sequence encoder-decoder architecture
- An **alternative intrinsic evaluation metric** based on distance distributions of three sets of word pairs that provides further insight into the informational capacity of learned word embeddings

1.3 Thesis Structure

Chapter 2 describes representation learning for text from the perspective of a neural network learning algorithm, as well as different approaches for representing language computationally. Chapter 3 reviews some of the most widely used word representation approaches, together with their corresponding theoretical and empirical advantages and limitations. This chapter also introduces the experimental design that is used throughout this research, and evaluation results for a set of baseline word representation models. Chapter 4 introduces a novel data augmentation process to inject linguistic knowledge into the training data of a Skip-gram word embedding model, as well as an embedding partition mechanism that controls how this augmented information gets propagated through the word embedding. Chapter 5 presents a new neural network autoencoder architecture that learns structure-aware word embeddings by training a model that maps a dependency parse tree to its originating sentence. Finally, chapter 6 contains a general discussion of the results of this research, concluding remarks, and plans for future work.

1.4 Notational Conventions

The following notationa convention will be used throughout this thesis:

- Matrices are represented with bold uppercase letters, e.g. $\mathbf{X}$
- Vectors are represented with bold lowercase letters, e.g. $\mathbf{x}$

- Square brackes with subscripts are used to refer to a specific element in a vector, e.g. $[\mathbf{x}]_i$, a row in a matrix, e.g. $[\mathbf{X}]_i$, or a cell in a matrix, e.g. $[\mathbf{X}]_{ij}$
- Ordered elements in a sequence have a superscript (t) to denote they appear at the tth position, or *timestep*, of the sequence
- Calligraphic uppercase letters are used to refer to sets, e.g. $\mathcal{V}$ is used to denote the vocabulary as a set of words
- `Monospaced font` is used to refer to text that is used as data
- `Green monospaced font` is used to refer to named features

Chapter 2

Representation Learning in Natural Language Processing

There can be no informational sensitivity without representation. There can be no flexible and adaptive response to the world without representation. To learn about the world, and to use what we learn to act in new ways, we must be able to represent the world, our goals and options. Furthermore we must make appropriate inferences from those representations.

Kim Sterelny, *The representational theory of mind: An introduction.*

Abstract representations of objects and ideas are fundamental for the human mind's process of knowing, understanding, and reasoning about the world around it. In the context of AI, one of the main overarching goals is to imbue machines with (some manner of) the reasoning and abstractive capabilities of humans. The pursuit of this goal has produced a wealth of computer-actionable representation families, such as symbolic logic, which lies at the heart of all programming languages and is modelled after human reasoning; or graphs and relational tables, the foundations for databases, which resemble memory and relationships between objects or concepts. The usefulness of these representations depends not on how closely they resemble cognitive processes, but rather on how effective they are at providing abstractions of information that can subsequently be computed with.

Until recently, all representations of the world had been created (or designed) by humans, but with the advent of machine learning, computers are now capable of producing their own representations. These automatically learned representations leave behind all notions of interpretability and real-world mimicry in lieu of task-specific representations that are (theoretically) optimal in informational capacity. Since these representations are learned solely in terms of their usefulness in solving a given learning task, there are no guarantees as to what aspects or patterns from the data get distilled into these representations, they depend solely on the way the problem is posed. These task-specific (and oftentimes uninterpretable) representations are a fundamental part of neural network-based machine learning pipelines.

The process by which neural networks learn to represent information is highly complex and dependent on multiple interrelated elements and design decisions. Gaining a better understanding into the representational process of neural network models is essential in order to construct more interpretable and data-efficient models, as well as more transferable data representations, as argued by Bengio et al. (2013)

The idea of sharing representations between multiple machine **learning tasks** is very closely related with the concept of **transfer learning**.[1] For a complete discussion on transfer learning methods and approaches refer to Pan and Yang (2009) and Lu et al. (2015). Transfer learning in AI refers to the process of exploiting knowledge learned from one task to solve another related task and is inspired by humans' ability to reuse knowledge obtained in the past when confronted with new scenarios or problems. Humans achieve this by abstracting and generalising the acquired knowledge, analysing the new setting and adapting the knowledge correspondingly. In the context of learning algorithms, this translates to finding commonalities in the data that enable the transfer of knowledge from one learning task to a different one. The focus of this research is on a transfer learning approach termed **feature representation transfer** (Lu et al., 2015), which seeks to build representations of the data that encode the necessary knowledge to solve a variety of

[1]Transfer learning has also been associated with other terms through the years, such as *lifelong learning*, *knowledge transfer*, *meta-learning*, *multi-task learning*, and *domain adaptation*.

different learning tasks.

2.1 A Neural Network Learning Algorithm for NLP

Neural network models, which are loosely based on the inner workings of the brain, consist of layers of interconnected non-linear processing units. Neural network models have become some of the most successful machine learning models as exemplified by Krizhevsky et al. (2012), Mikolov et al. (2013a), Sutskever et al. (2014), Devlin et al. (2018), Lee et al. (2021), among many others. The success of neural networks is largely due to their capacity to learn an informationally efficient representation of the input features by combining *neuron* units to approximate an target function and iteratively update or *learn* the connection strengths between these units to improve their approximation. These units can be arranged in different ways to either provide more abstract representations of the input, composing multiple layers to create "deep" networks, or to accommodate different types of input, such as fixed-size structures (e.g. images), variable-length sequences (e.g. text, audio), graphs, etc.

As described by Belz (2021), **reproducibility of results** has become an increasingly central topic in NLP research in recent years. Some of the most important conferences on machine learning, e.g. ICML, ICLR, NeurIPS, have introduced workshops and research tracks on reproducibility. In a survey conducted by Mieskes et al. (2019), in which they asked two hundred NLP practitioners about their perception of the state of replicability of experiments in the field of NLP, they noticed that the majority of respondents conceive replicability as being of high importance in NLP research. In spite of this, Belz (2021), Mieskes et al. (2019), and Mieskes (2017) identify issues with the current state of reproducibility, arguing that sharing the code and data used in an experiment are not sufficient to reproduce its results. In response to this necessity for more thorough reproducibility, I propose a framework that allows every individual experimental condition to be clearly documented.

This section provides an overview of neural network models from the perspective of NLP. These models are best understood in the context of the **learning**

algorithms within which they are used, so I will start by sketching out a generic learning algorithm, shown in figure 2.1, that encompasses multiple different applications, datasets, and architectures.

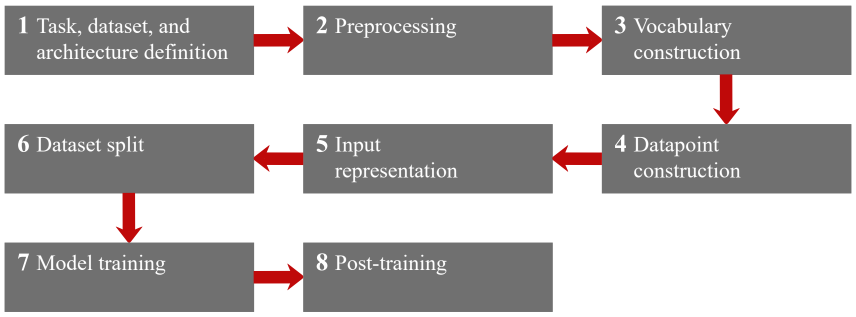
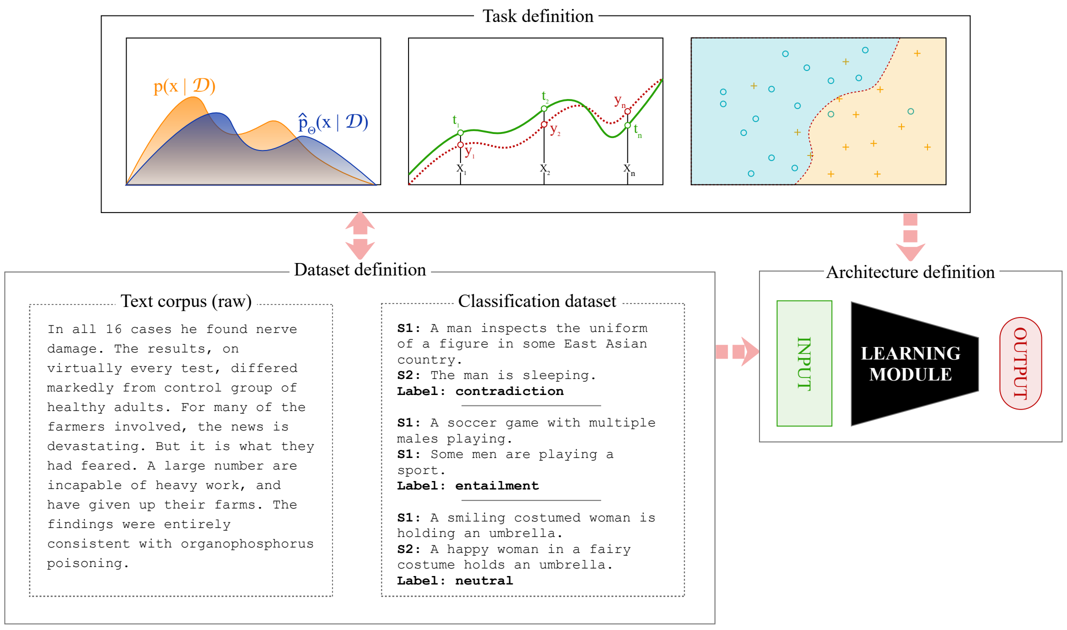


Figure 2.1: Simplified learning algorithm for an NLP task

2.1.1 Task, dataset, and architecture definition



```
In all 16 cases he found nerve
damage. The results, on
virtually every test, differed
markedly from control group of
healthy adults. For many of the
farmers involved, the news is
devastating. But it is what they
had feared. A large number are
incapable of heavy work, and
have given up their farms. The
findings were entirely
consistent with organophosphorus
poisoning.
```

```
S1: A man inspects the uniform
of a figure in some East Asian
country.
S2: The man is sleeping.
Label: contradiction

S1: A soccer game with multiple
males playing.
S1: Some men are playing a
sport.
Label: entailment

S1: A smiling costumed woman is
holding an umbrella.
S2: A happy woman in a fairy
costume holds an umbrella.
Label: neutral
```

Figure 2.2: Task, dataset, and architecture definition and their dependencies

The first step of a learning algorithm is to clearly and unambiguously define the main task ($\mathscr{T}$) to solve in terms of an target function which a model approximates by minimising a **loss function** ($\mathscr{L}$). The learning task will guide every other decision in the algorithm, from the choice of dataset to the success metrics and real world applications of a trained model. The task may be defined *directly* in terms of the end application, for example measuring the accuracy of a Part-of-Speech (POS) tagger (Huang et al., 2015); or *indirectly* as a proxy task towards a longer term goal, such as learning transferable text representations by solving a language modelling task (Mikolov et al., 2013b; Peters et al., 2018; Devlin et al., 2018).

The choice of dataset is one of the most important decisions for the success of a trained model. Machine learning models learn by example, so the number, quality, and diversity of the examples that are used to train these models will determine the limits of what the model will be able to learn. In a **supervised learning** setting, where the model is trained to associate a set of **inputs** ($\mathbf{x}$) to a pre-specified set of corresponding **targets** (τ), the selection of dataset is closely related to the specific prediction task or application the model is trained for. Given the high costs of annotating data (correctly and meaningfully), labelled datasets are very scarce, especially datasets that are large enough to meet the requirements of data-hungry neural network models. As a result, the choice of dataset is mostly constrained by the specific learning task, with many tasks having but a few, or even a single, viable datasets, such as the case of the **natural language inference (NLI)**[2] (Bowman et al., 2015) task, shown in the classification sample of figure 2.2, where the dataset was constructed specifically for this task. Even though there are several datasets for this task, like the datasets for *Recognizing Textual Entailment* tasks or the Winograd NLI dataset included in the *GLUE* (Wang et al., 2019b) and *SuperGLUE* (Wang et al., 2019a) benchmarks, the *Stanford NLI* (Bowman et al., 2015) and the *Multi-Genre NLI* (Williams et al., 2017) datasets are orders of magnitude larger,[3] which can be a significant concern when training data-intensive models such as large neural networks.

In **unsupervised learning**, dataset selection can be a fundamental part of the modelling process. Unsupervised learning refers to the family of tasks that do not require targets to train on, but rather aim to discover regularities in the data either

[2]NLI is a renaming of the *Recognizing Textual Entailment* three-way classification task first described by Dagan et al. (2006) where, given two text fragments, a *premise* and a *hypothesis*, the model must determine whether the (directed) relationship between the two fragments is an *entailment*, a *contradiction*, or whether the relationship between the two texts is semantically *neutral*.

[3]The *Stanford NLI* and *Multi-Genre NLI* contain ∼570k and ∼433k sentence pairs respectively, while the smaller datasets mentioned contain fewer than 1,000 sentence pairs each

to form datapoint groupings (*clustering*), to learn an approximation of the underlying data distribution (*density estimation*), or to learn a lower-dimensional mapping of the input (*dimensionality reduction*). Given that this type of learning does not require labels, the training dataset can be selected or constructed from a very wide variety of domains. Additionally, since there are no explicit labels to conduct the learning, these models will be very sensitive to the underlying distribution of the data, meaning that the choice of dataset will determine the type of information (and biases) the model will be able to capture (Bolukbasi et al., 2016a; Caliskan et al., 2017; Barikeri et al., 2021). In NLP, the choice of dataset has important implications with regards to the type of language usage (vocabulary employed, syntactic structures, level of formalism, etc.). Therefore, a model trained on, for example, a corpus of legal documents will be oblivious to the syntax of colloquial language, or the specialised vocabulary of scientific articles.

Finally, the design of the model's **architecture** is largely determined by the learning task. The model must have the right **capacity** to approximate the objective function. A very complex (high capacity) architecture might unnecessarily increase the computational cost of training, and is more prone to **overfitting**, which is the effect of a model becoming increasingly good at predicting the examples that it is trained with at the expense of its **generalisation** capabilities, i.e. its ability to make predictions on previously unseen data. Conversely, **underfitting** happens when the model is too simple to provide a good approximation to the objective function.

Moreover, the shapes of a model's input and output are also defined by the learning task. The loss function determines the shape of the output of a model, where a classification problem, for instance, requires the model to produce a probability distribution over a set of classes. The underlying structure of the dataset, in turn, defines the shape of the input to the model, which should be expressly designed to consume the variable length sequences that might be required when processing text, or the fixed size colour channels commonly used in image processing. The size of the dataset also has a bearing on the architectural design, since a large dataset can be used to train a more complex model without the risk of overfitting, but the increased number of parameters and training examples will cause the computational cost of training the model to increase as well.

2.1.2 Preprocessing

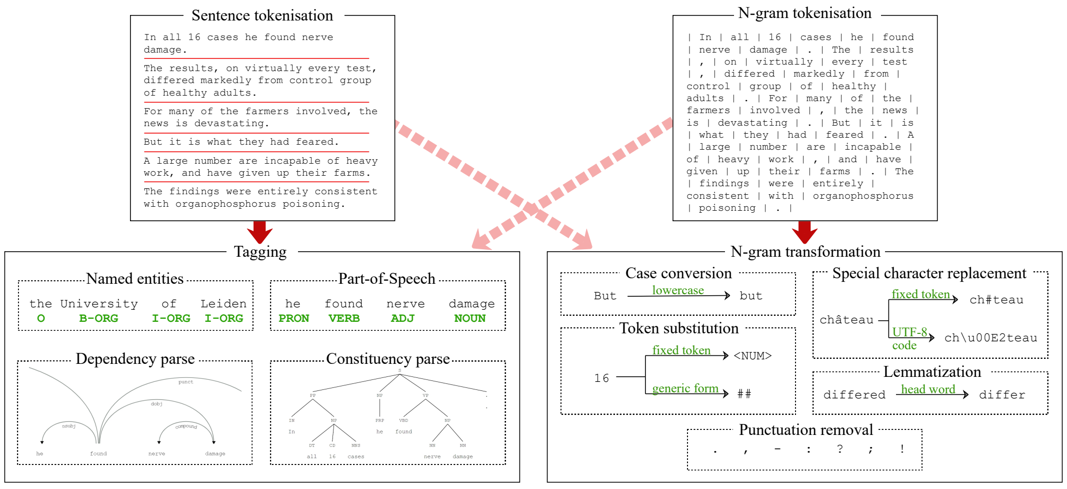


Figure 2.3: Preprocessing pipeline in two granularities: sentences and n-grams

Before the start of the training phase, the dataset typically undergoes a set of transformations to: (a) convert it into a format that is easier to process; (b) emphasise specific traits in the data; and/or (c) include any additional information that might be beneficial to the learning task. In an NLP learning algorithm, the first **preprocessing** decision is the text granularity to use during training. The transformations applied to the data when working at the document-level differ from those that are more useful at the sentence or **n-gram** (sequence of *n* textual tokens) levels. Some examples of preprocessing stages for text data can be seen in figure 2.3.

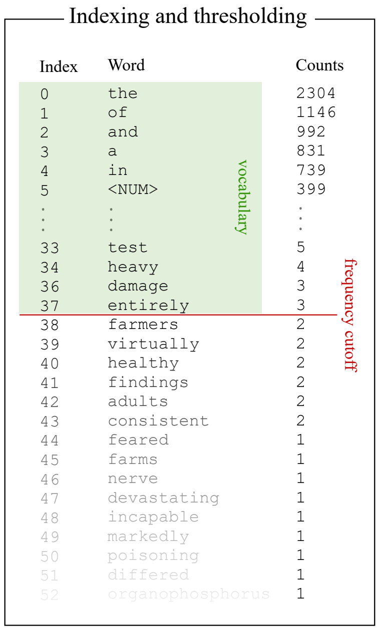
The most common initial preprocessing step in NLP is **tokenisation**, which is the process of splitting the text data into a set of tokens of a set **granularity** (e.g. paragraphs, sentences, n-gram, characters). Some applications require multiple tokenisation steps, and the order in which these steps are applied will determine the information that is lost in the process. For example, an application that works with interactions between words in a sentence would require the text corpus to first get tokenised into sentences, and subsequently tokenise each sentence into words. Performing these steps in the wrong order risks blurring or removing relevant information, such as the semantic delimitation that sentences provide in text. Tokenisation techniques can range from simple regular expressions (e.g. using white space to separate tokens) to complex rule-based systems like the *Stanford Tokenizer* (Stanford NLP Group), or learned subword segmenters such as the *WordPiece* model (Wu et al., 2016a).

Once the text has been tokenised it can be undergo additional transformations. Tagging sentences is a common way of incorporating linguistic information into the training data. Some of the most widely used tagging schemes are the directed grammatical relations between words provided by *dependency parse trees*, compositional information from *constituency parse trees*, words' functional roles from *POS* tags, or *named entity recognition (NER)* to identify entities such as locations, persons, or organisations. These types of tags are typically assigned at the sentence-level, since they depend on the full context of the sentence to determine whether, for example, the word *light* is used as an adjective, a noun, or a verb. Performance of current language taggers, such as the models provided by Honnibal and Montani (2017) or the current state-of-the art models in dependency parsing (Mrini et al., 2020) and POS tagging (Bohnet et al., 2018b), have allowed the tagging process to be automated with a relatively small number of incorrect tags.

At the n-gram level, additional transformations can help reduce the token diversity by unifying the letter case, and substituting or removing tokens (n-grams or characters). These transformations are often necessary to make a dataset computationally viable since they can substantially reduce the number of distinct tokens that appear in the data by merging tokens with slightly different spellings, special characters, or different letter casing. This reduction, however, comes at an informational cost, since it might merge together tokens that are meant to be distinct, like `Apple` (a company) and `apple` (a fruit), or it might entirely remove informationally relevant tokens such as punctuation marks or characteristic numbers.

In an unsupervised learning setting, the raw data is known as a *text corpus* ($\mathscr{Q}$). I will reserve the term *dataset* ($\mathscr{D}$) to refer to the data, both labelled and unlabelled, after it has undergone a preprocessing stage.

2.1.3 Vocabulary construction


Indexing and thresholding
vocabulary
frequency cutoff

Index	Word	Counts
0	the	2304
1	of	1146
2	and	992
3	a	831
4	in	739
5	<NUM>	399
⋮	⋮	⋮
33	test	5
34	heavy	4
36	damage	3
37	entirely	3
38	farmers	2
39	virtually	2
40	healthy	2
41	findings	2
42	adults	2
43	consistent	2
44	feared	1
45	farms	1
46	nerve	1
47	devastating	1
48	incapable	1
49	markedly	1
50	poisoning	1
51	differed	1
52	organophosphorus	1

Figure 2.4: Vocabulary construction process with a minimum frequency threshold

Processing written language implies dealing with discrete tokens, composed to form larger combinatoric structures. These compositional units can be collected into a dictionary where every term is assigned a unique index, this is the most basic process of numericalising a set of elements. When composing characters into words[4] the dictionary of characters is relatively small,[5] and gets composed into a finite set of discrete elements: a **vocabulary** ($\mathcal{V}$) of *valid* words which is orders of magnitude larger than the dictionary of their constituting characters. However, the phrases or sentences produced by composing words together can have arbitrary lengths and still be valid constructions in the language, and are therefore no longer a finite set. Words are commonly used as the base granularity in NLP due to the fact that, as described by Gasparri and Marconi (2016), they are the minimal units of meaning, yet they can still be efficiently processed as discrete tokens.

When using words as the basic language unit, there are several considerations

[4]For the remainder of this thesis I use the term *words* instead of *n-grams* since most of the work presented here is done on unigrams.

[5]While this is true for English, where the most basic character dictionary is the set of 128 characters in the *ASCII* encoding, for a logographic language like Chinese the full alphabet can comprise tens of thousands of characters.

to keep in mind. Even though the set of all words in a language is finite, the size of this set is usually too large[6] for NLP applications whose computational times might scale exponentially with the size of the vocabulary ($|\mathcal{V}|$). Additionally, when working with real-world data, there are bound to be errors in spelling, letter casing, punctuation, or spacing, as well as neologisms, which would further increase the size of this set of words. A practice commonly found in NLP literature (e.g. Collobert et al. (2011); Mikolov et al. (2013b)) for the construction of these vocabularies is to only include words that appear a certain number of times in the dataset, as depicted in figure 2.4. The rationale is that setting a *frequency threshold* removes misspelled words, which should be far less frequent than correct spellings, while at the same time allowing the model to focus on the subset of words for which more information is available. Using a limited vocabulary can also improve the model's learning process by reducing the model's size complexity and the sparsity of the representation space. However, there are downsides to using these smaller vocabularies, the most obvious of which relates to reducing the *coverage* of the data. Coverage in this context refers to the percentage of words in the data (either unique or repeated) that is captured by a given vocabulary. Removing infrequent words has little bearing on the coverage over repeated words in a dataset due to a phenomenon regarding the distribution of word frequencies in natural languages known as **Zipf's law** (Zipf, 1936). Zipf's law states that given a word w, the frequency of that word, $f(w)$, in a text corpus is roughly inversely proportional to its *frequency rank* $r(w)$:

$$f(w) \propto \frac{1}{r(w)^{\alpha}}$$

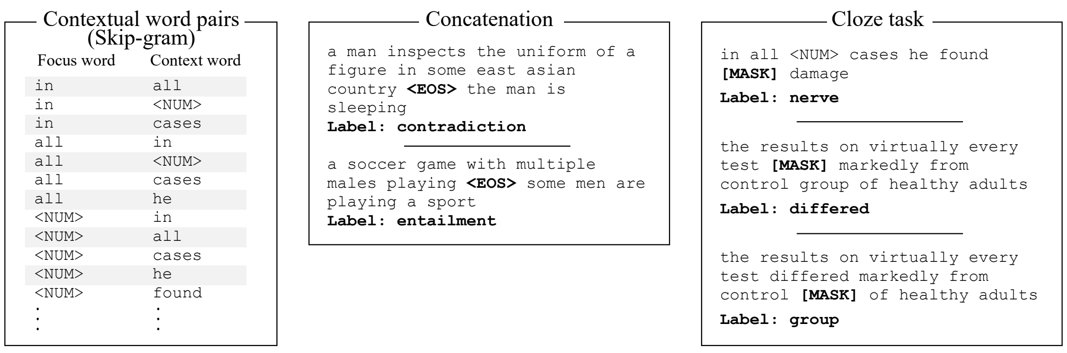
where $\alpha \approx 1$ is a constant.

As described by Piantadosi (2014), the original formulation of Zipf's law provides a simplified approximation to what is really a more complex word frequency distribution, but while there is no perfect fit for this distribution at this time, empirical evidence does seem to suggest it follows a *near-Zipfian* form.

[6]The *American Heritage Dictionary of the English Language* contains over 370,000 words, although other estimates calculate there are over one million words in the English language (Michel et al., 2011)

Even though for every word from the data that gets removed from the vocabulary there is a corresponding loss of information, this loss is not proportional to the coverage contribution of that particular word. A very infrequent word, such as `organophosphorus`, will have an almost negligible contribution to the coverage of the dataset, yet seeing such a rare word in the data can be extremely informative. For instance, in a document classification setting, a highly specialised word like `organophosphorus` could provide enough information to correctly classify a document as being related to agriculture. It is important to keep this tradeoff in mind when making decisions about the construction of the vocabulary.

2.1.4 Datapoint construction


Contextual word pairs (Skip-gram)

Focus word	Context word
in	all
in	<NUM>
in	cases
all	in
all	<NUM>
all	cases
all	he
<NUM>	in
<NUM>	all
<NUM>	cases
<NUM>	he
<NUM>	found
⋮	⋮

Concatenation

```
a man inspects the uniform of a
figure in some east asian
country <EOS> the man is
sleeping
Label: contradiction
```

```
a soccer game with multiple
males playing <EOS> some men are
playing a sport
Label: entailment
```

Cloze task

```
in all <NUM> cases he found
[MASK] damage
Label: nerve
```

```
the results on virtually every
test [MASK] markedly from
control group of healthy adults
Label: differed
```

```
the results on virtually every
test differed markedly from
control [MASK] of healthy adults
Label: group
```

Figure 2.5: Datapoint construction process for (a) the Word2Vec Skip-gram model for word embeddings (Mikolov et al., 2013a), (b) NLI sentence pair and labels, and (c) the Cloze task for sentence embeddings from Devlin et al. (2018)

Machine learning models are trained to map an input ($\mathbf{x}$) to its corresponding target (τ).[7] A datapoint in this context refers to the *input-target* tuple that makes up a single training example. The construction of individual datapoints from the data is therefore intimately connected with the learning task.

In supervised learning tasks, such as *text classification* or *NLI* (figure 2.5b provides an example of NLI), the process of constructing datapoints is straightforward, since the labelled datasets used for these tasks have already been formatted as input-target pairs. Datapoints in unsupervised learning tasks, on the other hand, do not

[7]This generalisation does not directly apply to reinforcement learning models. Since reinforcement learning falls outside of the scope of this research, these models are not considered in this discussion.

have pre-specified targets and can hence be defined in different ways. For example, in language modelling, an unsupervised learning task that consists of the probabilistic modelling of a word w occurring in a sequence given a conditioning *history* h, $p(w|h)$, datapoints are usually constructed by treating the history h as the input, and the word w as the target.

2.1.5 Input representation

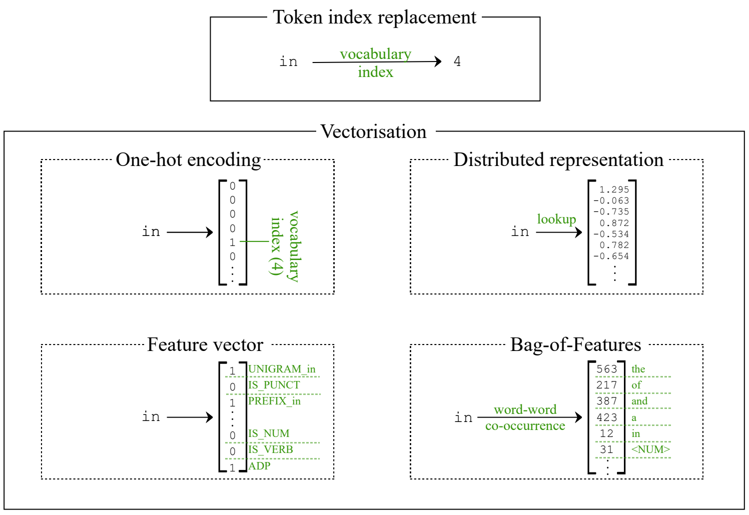


Figure 2.6: Common input representation strategies

The next step after constructing the datapoints is to package them in a numerical form that machine learning models can work with. The representation of the datapoints will have a large bearing on the quality and generalisational capabilities of the final model. Even though different machine learning models can learn to focus on the *features*[8] that are most relevant for the learning task they are trained to solve, the representation that is used in training determines the universe of information that the model can learn from. The best representation for a particular task is therefore determined by the specific information that the task requires (Collobert et al., 2011), but the scope of what constitutes "relevant information" is rarely discernible.

The possible representations of the input to an NLP learning algorithm are determined by the choice of text granularities. Elements are generally represented

[8] *Feature*, as used throughout this thesis, can be defined as the set of properties or constituting elements that are used to characterise a dataset.

either as distinct tokens from a finite dictionary, like words, characters, or linguistic features; or as compositional structures made up of collections of these tokens, like sentences as sequences of words. Representations for both tokens and structures can either be manually engineered or learned from data (a more thorough discussion on the construction of text representations is provided in section 2.2). Figure 2.6 shows some of the most widely used input representation strategies in NLP. The question of how best to represent textual data numerically remains a very active research problem, and the main focus of this thesis.

2.1.6 Dataset split

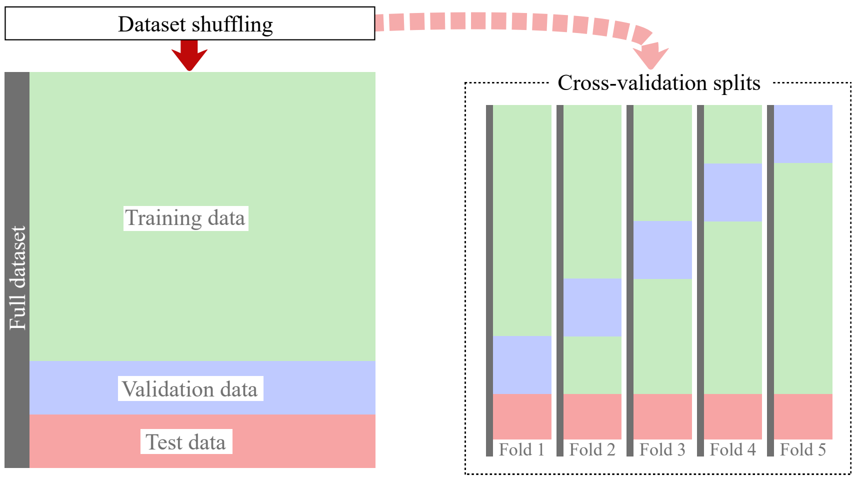


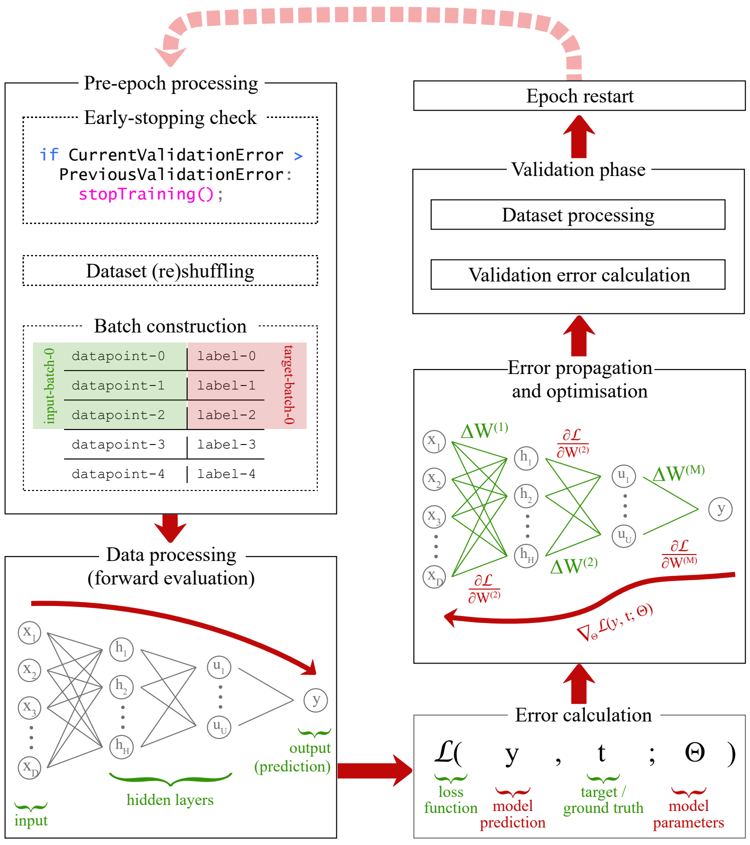
Figure 2.7: Train-Test-Validate dataset splitting methods

Machine learning models are trained to fit a dataset in order to match inputs to targets. With the current availability of computational resources, models can be arbitrarily complex. The implication of this increase in complexity is that these larger models can overfit even large datasets. To keep track of how well a model is fitting the data, along with its generalisation capabilities, the dataset is typically split into disjoint sets: a **training set** which is used for the data-driven optimisation of a model's parameters; a **validation set** to keep track of how well a model is generalising to unseen data as part of a model selection stage (i.e. selecting the model with the best predictive performance); and (sometimes) a **test set** that is used to measure the final model's performance on new data.

The model parameters will be updated in the training phase, meaning that the model will only ever explicitly fit the training set. The training error can give a sense of how well a model is fitting the data, and it should consistently decrease after every **epoch**, i.e. iteration over the full training set. Since the training set is the only partition that will be used to modify the parameters of the model, this partition should be the largest in size, a common heuristic in NLP literature is to use between 70% and 90% of the full dataset, although the exact number depends on the dataset size and the learning task. The validation set is typically the remainder of the full dataset (or around half of the remaining data when there is an additional test set). The expected behaviour of the validation error is for it to decrease during the initial epochs, until it reaches a point of inflection, generally regarded as the point of overfitting (or *optimal capacity* (Goodfellow et al., 2016)), after which subsequent training will result in poorer generalisation capabilities identified as an increasing validation error. A model can be prevented from reaching the point of overfitting through **early stopping**, also known as *stopped search* (Sjöberg et al., 1995), a process in which the validation error is used to signal the end of the training phase. In some cases, the dataset is partitioned into multiple train-validation splits, also known as folds, in a process known as **cross-validation** (Stone, 1974), which can be visualised in figure 2.7. Cross-validation allows the model to make more efficient use of the available data and provides a better estimate of the model's predictive performance on unseen data. Whenever the validation set is used to select model hyperparameters, it can no longer provide an unbiased evaluation of the model's performance. In these cases, the performance of the final model should be evaluated on a separate test set, made up of data that the model has never interacted with before.

The order of the training data is also an important factor. Natural language data is typically collected from real-world sources, such as news articles or books. The language or style used at the beginning of a dataset might differ greatly from that which appears in the middle or towards the end of the dataset as a result of differences in text categories (e.g. a dataset that combines fiction books with scientific articles, as with the *British National Corpus* (BNC Consortium, 2007) or the *Project Gutenberg* dataset (N.d.)), the time at which a particular text was written, different authors, etc. Since training happens sequentially, running through a dataset in order might have unwanted effects, such as specialising the model to the linguistic style that happens towards the end of the dataset, or using a validation set that does not reflect the full linguistic diversity of the data. It is therefore useful to consider shuffling the dataset before creating these partitions, as discussed by Goldberg (2017).

2.1.7 Model training



```
if CurrentValidationError >
  PreviousValidationError:
    stopTraining();
```

Figure 2.8: Generic model training process

The main part of the learning algorithm is the actual training of the model, which is abstracted in figure 2.8. *Gradient descent*-based models, such as neural networks (LeCun et al., 1989, 1998), may require multiple passes over the dataset before they reach a point of convergence, or local optimum. The first decision to make here is how long to train for, which is most commonly defined in terms of a fixed number of epochs, or some early stopping criterion typically linked to the validation error obtained at the end of an epoch.

Once the length of training is defined, everything inside the epoch is a repeating sequence of steps divided into a **pre-epoch** block, a **training phase** and a **validation phase**. The pre-epoch block consists of all actions that need to occur before the training and validation phases, which can include an early stopping check to determine whether or not to continue training, as well as actions that determine the order of training, like the shuffling of the dataset and the construction of datapoint **batches**, which are groups of datapoints that can speed up computation by being processed as a single input-target block. Dataset shuffling and batch construction can either be performed once before training begins, or at the beginning of every epoch.

The training phase is itself made up of three stages: it starts with the **data processing**, commonly called a *forward pass* in the context of neural networks, which refers to the processing of a datapoint or batch through the model to produce an output. The output produced is known as a model's *prediction*, $\mathbf{y} = f(\mathbf{x};\Theta)$, where $\mathbf{y}$ is the prediction, f is the predictive function with parameters Θ, and $\mathbf{x}$ is the input. In the second stage, this prediction is compared to the *target* (or *ground truth*) in terms of a loss function that defines a measure of dissimilarity between the prediction and the target and produces a *loss* (also called *error* or *cost*) term $\mathscr{L}$. The final stage consists of two steps: first it uses the loss term to calculate the contribution of the parameters of the model to the prediction in a process of error propagation that calculates the *gradient* of the loss function with respect to the parameters, $\nabla_\Theta \mathscr{L}$. For neural networks, the number and degree of connectivity of parameters can make the calculation of these gradients computationally expensive. The **back-propagation** algorithm (Rumelhart et al., 1986) offers a way to speed up this calculation by using the *chain rule of calculus* to break it down into a sequence of *partial derivatives*, which can be stored and reused in subsequent calculations. Consider a simple feedforward neural network: a *multilayer perceptron* with one hidden layer (this architecture will be described in more detail in the next section) defined as the mapping $\mathbf{y} = \sigma(\mathbf{W}^{(2)} g(\mathbf{W}^{(1)}\mathbf{x}))$, where $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ are the trainable parameters of the model, and σ and g are **activation functions** (i.e. element-wise, and commonly non-linear, functions). For a simple example consider an input $\mathbf{x} = \{x_1, x_2, x_3\}$, a scalar output $\mathbf{y}$ and a hidden layer of dimensionality 2, as depicted in figure 2.9, with the following parts and dimensions (superscript (n) refers to the layer number):

$$
\begin{aligned}
\Theta &= \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\} & \mathbf{W}^{(1)} &\in \mathbb{R}^{2\times3}\,,\ \mathbf{W}^{(2)} \in \mathbb{R}^{1\times2} \\
\mathbf{a}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} & \mathbf{x} &\in \mathbb{R}^{3}\,,\ \mathbf{a}^{(1)} \in \mathbb{R}^{2} \\
\mathbf{z}^{(1)} &= g(\mathbf{a}^{(1)}) & \mathbf{z}^{(1)} &\in \mathbb{R}^{2} \\
\mathbf{a}^{(2)} &= \mathbf{W}^{(2)}\mathbf{z}^{(1)} & \mathbf{a}^{(2)} &\in \mathbb{R} \\
\mathbf{y} &= \sigma(\mathbf{a}^{(2)}) & \mathbf{y} &\in \mathbb{R}
\end{aligned}
$$

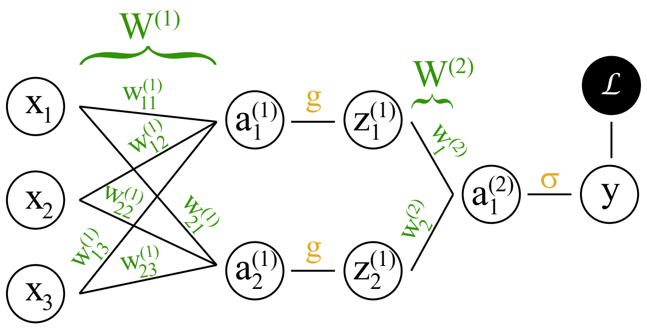


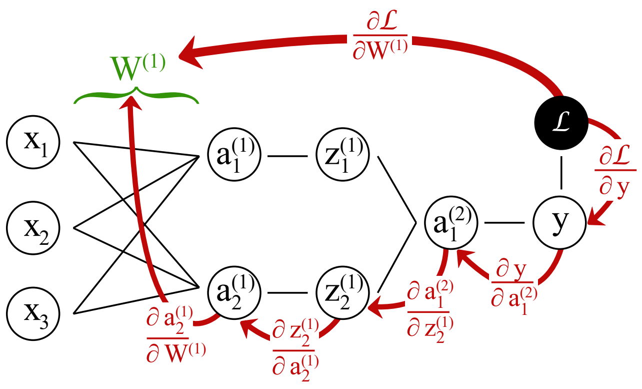
Figure 2.9: Basic multilayer perceptron

The gradient of the loss with respect to the parameters of the first layer $\mathbf{W}^{(1)}$ has the following decomposition,[9] represented graphically in figure 2.10:

[9]In the notation in equation 2.1, $\nabla_{\mathbf{W}^{(1)}}\mathscr{L}$ refers to the gradient of the loss $\mathscr{L}$ with respect to the weight matrix $\mathbf{W}^{(1)}$, the first right-hand side $\frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(1)}}$ is a scalar by matrix derivative, and all partial derivatives after that are scalar by scalar derivatives.

$$\nabla_{\mathbf{W}^{(1)}}\mathscr{L} = \frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(1)}} \tag{2.1}$$

$$= \frac{\partial \mathscr{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{a}_1^{(2)}} \sum_{i=1}^{2} \frac{\partial \mathbf{a}_1^{(2)}}{\partial \mathbf{z}_i^{(1)}} \frac{\partial \mathbf{z}_i^{(1)}}{\partial \mathbf{a}_i^{(1)}} \sum_{j=1}^{3} \frac{\partial \mathbf{a}_i^{(1)}}{\partial \mathbf{W}_{ij}^{(1)}} \tag{2.2}$$



$\frac{\partial \mathcal{L}}{\partial W^{(1)}}$
$W^{(1)}$
$\mathcal{L}$
$\frac{\partial \mathcal{L}}{\partial y}$
x_1
x_2
x_3
$a_1^{(1)}$
$z_1^{(1)}$
$a_2^{(1)}$
$z_2^{(1)}$
$a_1^{(2)}$
y
$\frac{\partial y}{\partial a_1^{(2)}}$
$\frac{\partial a_1^{(2)}}{\partial z_2^{(1)}}$
$\frac{\partial z_2^{(1)}}{\partial a_2^{(1)}}$
$\frac{\partial a_2^{(1)}}{\partial W^{(1)}}$

Figure 2.10: Back-propagation through a multilayer perceptron

This calculation is followed by an **optimisation**[10] step, which uses the gradients calculated during back-propagation to improve the predictive performance of the model by iteratively updating its parameters to minimise the loss function and more closely approximate the target function. This type of iterative optimisation is called *gradient descent* (Cauchy, 1847), and has the general form $\Theta \leftarrow \Theta + \eta \nabla_{\Theta}\mathscr{L}$, where η is a *hyperparameter* that controls the size of the update to the parameters and is commonly called a **learning rate**. Gradient descent can be visualised as the process of climbing down a function by taking *steps* in a specific *direction*. There are hence two main decisions to make regarding the optimisation: how to calculate this direction; and, once that direction is known, the size of step to take (η). The gradient can be calculated on the full training set (i.e. *batch* gradient methods (Nocedal and Wright, 1999)), but this can become costly with large datasets, even with the computational advantages that back-propagation contributes to the gradient cal-

[10]In the context of neural network, the non-linear structure of the models makes *analytic* (or *closed-form*) *optimisation* impossible, so this term will be reserved for *gradient-based* optimisation, which is conventionally used in neural network training.

culation. A common alternative is to use some type of *online* or *stochastic* methods, such as **stochastic gradient descent (SGD)** (Bottou, 1998), which estimates the gradient from a single, or a small number of, sampled datapoints. The second decision regards the learning rate, which can play an important role in the context of (highly) non-convex function minimisation. A large learning rate, for instance, can overshoot *local minima*,[11] while a very small rate can take too long to converge. To address this, an optimiser can use learning rates that are either *dynamic*, like *momentum* methods (Qian, 1999; Sutskever et al., 2013) or decaying learning rates found in modern SGD implementations; or *adaptive*, where individual learning rates for every parameter adapt over the training phase based on information from the partial derivative of the loss with respect to the specific parameter, such as the history of the (squares of the) gradients, as in *AdaGrad* (Duchi et al., 2011) and *ADADELTA* (Zeiler, 2012), or the first and second moments of the gradients used by the *Adam* optimiser (Kingma and Lei Ba, 2015).

The last part of the epoch is the validation phase, which consists of a subset of the steps described in the training phase: the forward pass followed by the error calculation. This phase is much shorter than training because it skips the error propagation and optimisation steps, and is typically performed on a much smaller dataset. The sole objective of the validation phase is to provide an indication of the generalisation capabilities of the model when confronted with unseen data, which is in some cases (i.e. early stopping) used to signal the end of training.

[11]Unlike convex functions, which have a single *global optimum*, non-convex functions can have multiple *local optima*, which correspond to points where the function takes a maximum or minimum value relative to neighbouring points in the function.

2.1.8 Post-training

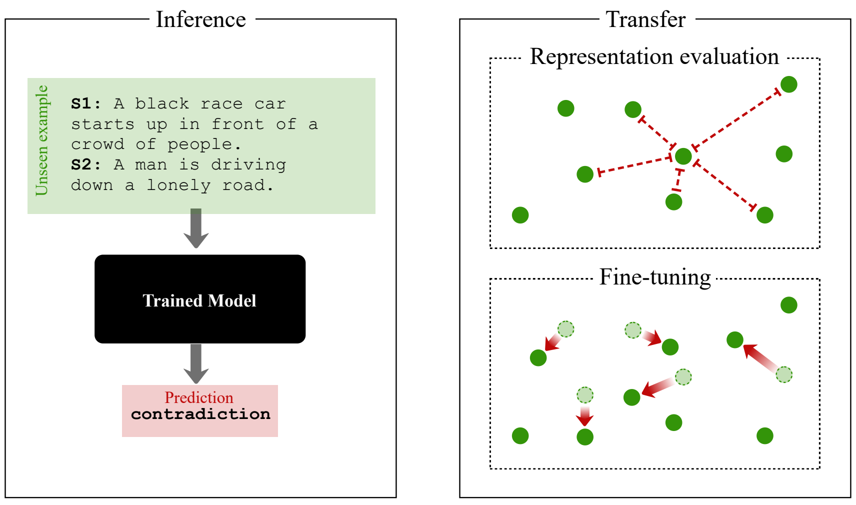

Inference
Unseen example
S1: A black race car starts up in front of a crowd of people.
S2: A man is driving down a lonely road.
Trained Model
Prediction
contradiction
Transfer
Representation evaluation
Fine-tuning

Figure 2.11: Steps after model training is concluded

Upon conclusion, model training produces a set of final parameters, $\hat{\Theta}$, that constitute its best approximation to the target function. The final *trained model*, $M_{\hat{\Theta}}$, can be used in two scenarios (shown in figure 2.11): **inference** and **transfer**. For inference, the model's predictive performance is typically evaluated on a held-out *test set*, and the model is subsequently used to make predictions on unseen data. In the transfer scenario, the trained model is used as part of a different learning algorithm. A common transfer scenario, especially in the realm of NLP, is known as *representation pre-training*, where the trained model produces data representations to be used as input to another model. In this case, the model has to be evaluated not in terms of the task it was trained to solve, but in the quality of representations that it produces.

A trained model does not need to be static. Data changes over time, text data, for instance, can evolve through the incorporation of new terminology, stylistic changes, or the emergence of new linguistic domains. The idea that a trained model can continue learning has been explored under different names, such as *never-ending learning* (Mitchell et al., 2018), which optimises a joint performance metric over a set of learning tasks through *cumulative self-supervised learning* over a sustained period of time; or *lifelong learning* (Parisi et al., 2019), which learns from a “continuous stream of information” while avoiding *catastrophic-forgetting* (McCloskey and Cohen, 1989; Goodfellow et al., 2015), where new information overwrites patterns learned from earlier datapoints.

Another option to tackle trained model staticity, which is more widely used in the NLP community, is model *fine-tuning* (also referred to as *post-processing*), a family of techniques that modifies the parameters of a trained model to adapt it or *specialise* it to a different domain or an evolving data distribution. Some common fine-tuning techniques include *specialisation*, where the parameters of a trained model are used as the initial parameters of a new model; or *retrofitting*, which directly modifies the geometric placement of learned representations to reinforce specific relations (Faruqui et al., 2015; Vulić et al., 2018).

2.2 Representing Language

Language is a structured system used to compose and communicate information. Written language, or *text*, maps the utterances of spoken language into a graphical system constructed through the ordered concatenation of a set of reusable symbols. Text is therefore the foremost representation of natural language. As any other representational system, text aims to preserve as much of the (semantic) information contained in language as possible while mapping it into a compressed format consisting of a limited set of symbols and compositional rules. Text is a representation constructed by and for humans, it is tailored to humans’ cognitive faculties. Humans can reconstruct the ideas that are encoded into these representations through a process of interpretation (i.e. translating the set of symbols into mental concepts or abstractions) and extrapolation (i.e. contextualising these mental concepts to infer the missing information).

Unlike humans, computers deal not with abstractions and concepts, but with numbers and logical operations. Text in its *natural* form cannot be processed by a computer, most text-based computational tasks require a process of *numericalisation* by which text is translated into numbers. The simplest way to numericalise text is to convert every character in a string into its corresponding code from a predefined encoding, such as ASCII or UTF-8. While this process is efficient for activities like editing text documents or printing text to a screen, these representations are devoid of the semantic information that is necessary for virtually any meaningful NLP task.[12] Operating on the semantics of language therefore requires semantically rich representations, but building these representations for machines that lack real-world contextual information and common sense requires rethinking the abstraction process.

Humans typically learn to read alphabet-based languages by first memorising the individual characters and then the words that can be formed by them. Both symbols (graphemes) and words are treated as distinct tokens from a finite set, an *alphabet* (or *character set*) for symbols, and a *vocabulary* for words. Characters and character groups are associated with sounds, and words are associated with a meaning.[13] Words are used as the semantic and grammatical building blocks of language. Humans do not learn all possible combinations of words as distinct elements, but rather learn the words and their compositional rules to create larger structures like phrases, sentences, paragraphs, or documents. The order in which words are combined is also fundamental in the construction of higher level concepts.

The first decision when representing text is the base granularity to represent. Using words as the basic building blocks will have different computational and informational implications than using characters, n-grams, or phrases. Once the granularity is chosen, the simplest representation technique is **indexing**, which treats every token (e.g. word) as an element in an ordered dictionary (e.g. vocabulary) and replaces every token by its corresponding index in that dictionary. This representation requires a simple replacement operation which can be easily reversed to recover the original input. Nevertheless, the use of numeric indices[14] brings about

[12]It is worth noting that character-level neural network models have achieved important results in tasks like text classification (Zhang et al., 2016) and neural language models (Bojanowski et al., 2017; Kim et al., 2016), among others, these successes should not be attributed to the semantic information contained in individual characters, but rather to the compositional capabilities of the neural network models that are used to process them.

[13]In *logographic* languages, characters can have an associated meaning. However, the current discussion is centred on alphabetic language and so character-level semantics will not be considered.

[14]Hash maps, which will not be discussed here, can be used as an alternative to numeric indexing, but they bring about an additional set of complications, since their mappings are irrecoverable and

additional artifacts related to the properties of integers, such as their inherent ordering and their arithmetic operations, which are not defined for tokens of language. To exemplify, there is no linguistic notion of a phrase being *greater than* another, or of what results from multiplying two words; and words with numerically close indices should not necessarily be considered semantically similar.

Some of the downsides of indexing are addressed by **one-hot encodings**, which replace integer indices with vectors whose dimensionality equals the number of elements in the token dictionary, where every dimension contains a 0 except for the dimension whose position matches the index of the token. A one-hot encoding of the fourth token in a dictionary would therefore be $\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & ... & 0 \end{bmatrix}$. The use of a positional system removes unwarranted relationships between tokens due to the fact that all of these vectors are orthogonal and equidistant to each other. Currently, one-hot encodings are used, at least as a first representation step, in most NLP applications and learning algorithms. They owe this popularity to their geometric properties (i.e. orthogonality and unit norm) and their ability to cast a lookup function as a matrix operation, which is extremely useful to interface with (matrix-based) machine learning architectures. But one-hot encodings are not without their shortcomings. The structure of these vectors makes them *informationally dull*, since their strict orthogonality and unit norm requirements make them incapable of representing any manner of similarity or relation between tokens. Additionally, incorporating new elements to the dictionary will affect the dimensionality of all existing vectors.

The dimensions of a representation vector do not need to represent tokens, one-hot encodings can be extended to represent elements in terms of their constituting parts. This family of vectorised representations treats every dimension as a feature from a shared feature dictionary. **Feature vectors** are binary vectors that can represent discrete elements, such as words or n-grams, in terms of (presence-absence of) a set of (typically) hand-crafted linguistic features, such as whether they are numeric, their POS tag(s), or their position in a sentence, as shown in the example in

can result in collisions, making them unsuitable for many applications.

Table 2.1: Example of manually constructed feature vectors for the words in the phrase `Calling number 8`. Features include bi-grams, positional information (first or last word), suffixes, and whether the word is numeric

	uppercase	SUFFIX_ing	numeric	<START>_calling	calling_number	number_8	8_<END>
`Calling`	1	1	0	1	0	0	0
`number`	0	0	0	0	0	1	0
`8`	0	0	1	0	0	0	1

table 2.1. Alternatively, a **bag-of-features** representation considers each element as a collection of features that may occur several times. The most common example of this is the *bag-of-words* method, which represents a phrase or document in terms of the frequency counts of its constituting words (or other importance-weighted frequency measures such as term frequency-inverse document frequency[15] (Harris, 1954)). These two representational approaches are able to represent any number of elements as sparse vectors (i.e. made up mostly of zeros) without modifying their dimension, since the features they use are shared across all elements, which makes them capable of representing combinatorial structures like documents. However, the effect that specific sets of features have on a particular task is not entirely predictable, even when these features are engineered from solid linguistic principles. These representations also come at an informational cost, where in many cases the original input cannot be fully recovered from its representation, either because the feature dictionary fails to cover the entirety of the information in the input (e.g. words in a document that get pruned from the bag-of-words dictionary), or because the process of representation destroys relevant information (e.g. word order in bag-of-words).

Distributed representations, first described by Hinton et al. (1986), shift away from the constraint that every dimension in a representation vector must correspond to a concrete element (e.g. token or feature), instead replacing it with the idea that large numbers of tokens or features can be represented as "patterns of

[15]Term frequency-inverse document frequency, factors the counts of a word in a document together with the number of documents in the dataset that contain the term, which weights down words that appear in many of the documents, considering them *uninformative* in terms of distinguishing among different documents.

activity"[16] in *lower dimensional*[17] *dense* vectors (in contrast with the sparse vectors produced by the earlier vectorisation approaches). These representations form a shared, low-dimensional, continuous *space* that can accommodate an arbitrary number of elements. The dimensions of these vectors are abstract, which eliminates the need for feature-engineering. But the complex dynamics between these abstract features have no well-defined human interpretation. This is the greatest paradigm shift when moving from feature-engineering to feature learning: the most efficient representations that can be fed to a machine to solve a particular task need not hold any resemblance with the representations humans are used to working with.

Distributed representations are typically derived by performing some type of *dimensionality reduction* on higher dimensional representations, which aims to capture meaningful information into a lower dimensional representation while preserving the original geometry of the data. Dimensionality reduction techniques can be divided into *linear* and *non-linear*, and further subdivided into *supervised* and *unsupervised*, as described by Sugiyama (2016). Linear dimensionality reduction maps the original input $\mathbf{x} \in \mathbb{R}^D$ to a lower dimensional space $\mathbb{R}^H$ through a *linear transformation* $f : \mathbb{R}^D \mapsto \mathbb{R}^H$ defined as $f(\mathbf{x}) = \mathbf{P}\mathbf{x}$, where $\mathbf{P} \in \mathbb{R}^{H \times D}$ is a projection matrix. The best-known method in unsupervised linear dimensionality reduction is **Principal Component Analysis (PCA)**, which orthogonally projects the data onto a *principal subspace* $\mathbb{R}^H$, where the H principal components are the directions that preserve the most variation in the data. PCA is an unsupervised method given that it is optimised in terms of a *reconstruction error*, which is the cost of recovering the original data from the lower-dimensional representation. Linear dimensionality reduction can also be *supervised*, as in the case of *Fisher's linear discriminant* classifier, which takes in input-target pairs, $\{(\mathbf{x}_i, \tau_i)\}$, to find a linear transformation

[16]*Activity*, in this context, refers to the values across all dimensions of the vector representation, and is meant as an analogy with the patterns of electrical activations of neurons in the brain.

[17]Distributed representations are considered to be lower dimensional when compared to sparse representations. In NLP applications, it is common for the size of a vocabulary or feature dictionary to be in the tens or hundreds of thousands, or even in the millions (Mikolov et al., 2013a), while the number of dimensions in distributed representations typically range in the hundreds (e.g. 300 for Word2Vec word vectors (Mikolov et al., 2013a), or 768-dimensional text representations in BERT representations (Devlin et al., 2018)).

that maximises class separation, i.e. a projection that brings together datapoints belonging to a class while separating them from datapoints from other classes.

Neural networks with at least one *hidden layer* and a non-linear *activation function* are currently the most widely used non-linear dimensionality reduction models. The hidden layer in a neural network learns a non-linear mapping of the input $\mathbf{x}$ to a (typically)[18] lower dimensional space $\mathbb{R}^H$. By optimising this mapping with respect to a loss function the expectation is that the neural network learns a compressed representation of the input that is informationally optimal to solve the task it is being trained for. This process is sometimes referred to as the *information bottleneck* method (Tishby et al., 1999). I refer to the representations learned by neural networks as **embeddings**, to distinguish them from other distributed representations.

Autoencoders are a clear example of neural network models that perform unsupervised dimensionality reduction. An autoencoder is made up of two processing blocks: an **encoder** learns a mapping $f : \mathbb{R}^D \mapsto \mathbb{R}^H$ from the input $\mathbf{x}$ into a latent space $\mathbf{h} = f(\mathbf{x})$, and a **decoder** learns a reconstruction function $r : \mathbb{R}^H \mapsto \mathbb{R}^D$ that attempts to recover the original input $\mathbf{x}$ from this latent representation (or *code*) $\mathbf{h}$ through the function $\hat{\mathbf{x}} = r(\mathbf{h})$. These unsupervised models are optimised with respect to a reconstruction loss $\mathscr{L}(\mathbf{x}, \hat{\mathbf{x}})$ where the input doubles as the target. In contrast, supervised neural networks learn a mapping of the input that simplifies some associated learning task. In a classification task, for example, the intuition is the same as with Fisher's linear discriminant: to learn a mapping of the input that encodes members of the same class close to one another while separating members of distinct classes, where the non-linear quality of these models allows them to learn more complex mappings.

The mappings learned by neural networks are informationally (near-)optimal[19] only *in terms of the learning task*, which means that these representations offer no

[18] *Overcomplete* hidden layers, common in denoising autoencoders, have a higher dimensionality than the input. The current discussion, however, focuses on the *undercomplete* version of neural networks.

[19] Neural networks usually optimise a non-convex function, so there is no guarantee of global optimality from a trained model.

informational guarantees outside of this narrowly defined task. However, if the task that the neural network is trained for is general enough, such as the reconstruction objective of autoencoders or language modelling, the model can potentially learn representations that can be reused or *transferred* to solve a different task than the one they were optimised for. Throughout this thesis I make a terminological distinction between these two representational paradigms and refer to the process of learning a mapping for a narrowly defined task as **feature learning**, while I use the term **representation learning**, as originally proposed by Bengio et al. (2013), for the mapping that seeks to learn more general representations that can be transferred to other tasks or domains. Representation learning in NLP is usually seen in the form of *unsupervised pretraining*, which is the process of using large unlabelled datasets to learn embeddings that can be exploited when solving *downstream tasks*. These embeddings are most commonly used to provide an input representation for words (Mikolov et al., 2013a; Pennington et al., 2014), phrases (Devlin et al., 2018), documents (Le and Mikolov, 2014b), etc. that is, in theory, imbued with semantic and syntactic information that can aid in solving the new task. Pretrained embeddings have been empirically shown to improve performance on a wide variety of NLP tasks, and are used as input representation for a large proportion of the state-of-the-art models in NLP (Brahma, 2018; Devlin et al., 2018; Bahuleyan et al., 2018; Lin et al., 2017).

As described in the previous section, the structure of the input in an NLP learning algorithm depends on the type of data, the task definition, and the choice of text granularity. The input structure, together with the learning task, are the most important factors for the design of a neural network architecture. The **multilayer perceptron (MLP)** is the most basic neural network architecture, and is also known as a *feedforward neural network* because processing happens in a single direction, without *feedback* connections. An MLP consisting of a single hidden layer is defined as the composition of functions $\mathbf{y} = f^{(2)}(f^{(1)}(\mathbf{x}))$, where each function $f^{(\ell)}$ is a *layer* defined as a parametric function $f^{(\ell)} = g(\mathbf{W}^{(\ell)}\mathbf{x}^{(\ell)})$ that applies an affine transformation to the input (of that layer) $\mathbf{x}^{(\ell)}$ of the form $\mathbf{a}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{x}^{(\ell)}$ (the bias term is obviated here for notational simplicity) which is then passed through a non-linear *activation function* $g^{(\ell)}$. The full expression for a two-layer MLP is then $\mathbf{y} = \sigma(\mathbf{W}^{(2)} g^{(1)}(\mathbf{W}^{(1)}\mathbf{x}))$, where $\Theta = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$ is the set consisting of the trainable parameters of all the layers of the network, and g and σ are non-linear functions.

The activation functions for each layer can be different, but they generally come from a limited family of element-wise non-linear functions. Two common choices for activation functions are *rectified linear unit (ReLU)*, $\mathrm{ReLU}(x) = \max(0, x)$, or *hyperbolic tangent (tanh)*, $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, in part because they have simple derivatives that simplify the gradient calculations:

$$\mathrm{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\tanh'(x) = 1 - \tanh^2(x)$$

The choice of activation function for a hidden layer usually follows common heuristics (i.e. popular choices for a particular architecture or problem domain), or empirical comparisons. The **output function** σ, however, is different from hidden layer activation functions in that it is directly connected with the definition of the loss function. For example, a *regression* problem, that predicts a numerical value from an input by approximating an underlying (continuous) function, usually uses a single unbounded real number as output, whereas a multi-class *classification* problem might require a vector of probabilities (i.e. bounded in $[0, 1]$ and summing to 1) over all classes.

A well-known theoretical result by Hornik et al. (1989), and further generalised to non-continuous and non-smooth functions by Leshno et al. (1993), establishes that a MLP with at least one non-linear hidden layer can approximate any measurable function with arbitrary precision. The characterisation of MLPs as **universal approximators** only holds when the size of the hidden layer is sufficiently large, although increasing the number of layers (*depth*) in the network can, in principle, reduce the number of parameters required for the approximation function. While this result guarantees an *approximation potential* for MLPs, the approximation itself is dependent on the full learning algorithm. Architectural variations of these basic neural networks can simplify the approximation process by incorporating priors that provide information on the hierarchical or compositional structure of the data, the levels of abstraction or amount of compression required by the task, etc.

In NLP, MLPs have been successfully used for tasks like text classification (Joulin et al., 2017), embedding words (Mikolov et al., 2013a; Bojanowski et al., 2017) and documents (Le and Mikolov, 2014a), among others. The structure of the MLP only allows it to process a *fixed size* input consisting of a vector (datapoint) or matrix (batch of datapoints) of fixed dimensionality, which requires text to be represented in a static format such as the bag-of-words representation discussed above. Despite the current state-of-the-art being dominated by more complex architectures, MLPs are still exploited as a lightweight alternative that can efficiently train on large datasets in reasonable computational time.

Convolutional neural networks (CNNs) (LeCun, 1989) are an architectural variation of feedforward neural networks that effectively reduce the number of parameters of the network by replacing the (first few) *fully connected* layers of an MLPs with *feature maps*. A feature maps constructs a lower-dimensional representation of the input by extracting its *local salient features*. This mapping is done by scanning the input with a fixed-size *kernel* (or *filter*) function that uses a shared set of parameters to perform a *convolution* operation followed by a non-linear activation function on a region of the input. A subsequent *pooling* operation is performed on the resulting activations to return a single value to describe that region. The most common pooling functions include taking the maximum value (*max-pooling* (Zhou and Chellappa, 1988)) or an average over the activations. Feature maps can be stacked to produce increasingly abstract representations of the input. CNNs have been most commonly used in the field of computer vision, since they can process (fixed-size) images and extract localised features, like lines, and compose them into larger structures, like shapes or objects, in low dimensional representations, with the added benefit that these features are *invariant* to transformations (e.g. translation or scaling).

The fixed input size restriction makes the application of CNNs to NLP less obvious. While CNNs could, in principle, be used to process a bag-of-words representation for a piece of text, the concept of a *local feature* is not defined for this input since bag-of-words disregards ordering information from the original text and vicinity is only defined in terms of the term order in the vocabulary, which is not necessarily meaningful. To exploit the compositional capabilities of CNNs, text needs to be represented in a way that preserves order or relationship information. The most widely used technique to address this is to represent text as a sequence of tokens (usually words or characters) of fixed length, where longer sequences get truncated and shorter sequences get *padded* by filling the empty slots at the end of the sequence with a generic token (`<PAD>`). CNN models have achieved state-of-the-art performance on tasks like text classification (Kim, 2014; Zhang et al., 2016; Conneau et al., 2017; Johnson and Zhang, 2017), NLI (Mou et al., 2016a), language modelling (Dauphin et al., 2017), and machine translation (Gehring et al., 2017).

Many real-world problems have an underlying *temporal* aspect. Feedforward neural networks can be extended to model time by bestowing them with *memory*, which can be modelled by allowing information to flow from one *timestep* to the next in terms of a *recurrent* connection between the hidden layers at times t and $t+1$, the resulting architecture is called a **Recurrent Neural Network (RNN)** (Elman, 1990). The basic implementation of an RNN can be realised by introducing a set of shared parameters that connect the hidden layers at consecutive timesteps, these parameters are known as *hidden-to-hidden* connections (*hh*), $\mathbf{W}_{hh}$. Every hidden layer is therefore defined in terms of the input at time t and the hidden layer at time $t-1$: $\mathbf{h}^{(t)} = f(\mathbf{x}, \mathbf{h}^{(t-1)}; \Theta)$. The hidden layers in an RNN contain information about the full sequence up to that timestep. Since there is an inherent loss of information in mapping full sequences of arbitrary length into a fixed dimensional vector, the network must learn to use this hidden layer efficiently by only keeping track of the aspects of the sequence that are most relevant in solving the current training task.

The gradient calculation for RNNs must account for the different timesteps of the input. This is typically done by a process of *unrolling* the RNN, where all timesteps are put together in a single computational graph and all hidden layers are stored in memory to perform back-propagation on the unrolled RNN by treating it as a very deep network where every timestep gets interpreted as a layer in the network. This process is called *back-propagation through time (BPTT)* (Werbos, 1990).

The basic RNN architecture (Elman, 1990) has three important inherent issues:

- The **computational cost** of storing the parameters for all timesteps
- **Vanishing or exploding gradients**, which are a byproduct of repeatedly multiplying by the same parameters when modelling long sequences with RNNs. The exploding gradient problem has been solved by *gradient clipping* (Pascanu et al., 2013), which consists of limiting the size of the norm or the elements of the gradient to a set threshold before updating the weights. Various solutions have been proposed to address the vanishing gradient problem that involve modifying the architecture or adding a regularisation term (Pascanu et al., 2013), but at the time of writing there is no agreed upon solution.
- Downweighting or "forgetting" **long-term dependencies**, given that the use of a single set of *memory* parameters will cause the interactions between inputs that are far apart in the sequence to be weighted down in comparison to those closer together. Architectural variants of RNNs called *gated RNNs* refine the modelling of long-term dependencies by introducing a gating mechanism in the hidden layers that is selective about the information that is retained and that which gets "forgotten". The most widely used gated RNNs are the *gated recurrent unit (GRU)* (Cho et al., 2014) and the *LSTM* (Hochreiter and Schmidhuber, 1997). An additional variant to improve dependency modelling is to make the RNNs bidirectional by concatenating two hidden layers, where one layer has connections from the past to the future, and the second has connections from the future to the past, which allows the context to account for both the previous and the next elements of the sequence when

processing the current input.

The sequential structure of language benefits from the temporal abstraction in RNNs, since these networks are equipped to accommodate arbitrarily long sequences and their encoded memory can learn to represent dependencies between different timesteps. RNNs, and specifically LSTMs, have been the standard architecture for NLP, where they have achieved state-of-the-art results in several tasks such as NER (Akbik et al., 2018; Huang et al., 2015), POS tagging (Bohnet et al., 2018a; Ling et al., 2015), sentiment analysis (Howard and Ruder, 2018; Brahma, 2018), and language modelling (Melis et al., 2020).

Even though gated RNNs make an improvement on the modelling of long-term dependencies, their sequential processing still retains a bias towards proximal elements in the sequence. To better model very long sequences and capture specific interactions between elements in the sequence regardless of their position, Bahdanau et al. (2015) incorporate an **attention mechanism** into the modelling of the sequence which combines hidden layers with a set of *attention weights* that explicitly model dependencies between elements in a sequence. The original formulation of attention by Bahdanau et al. (2015) uses these weights to learn an alignment between two sequences in the context of a machine translation task, but the *self-attention* mechanism proposed by Lin et al. (2017), which learns attention weights between the elements of a single sequence, has been more widely adopted for other NLP tasks. While attention mechanisms have proven to be beneficial when incorporated to RNN architectures in tasks like machine translation (Bahdanau et al., 2015) or text classification (Bahuleyan et al., 2018), they appear to achieve their full potential when combined into *attention networks*. The **Transformer** architecture (Vaswani et al., 2017) replaces recurrence with stacks of self-attention layers for sequence modelling. Transformer-based models have recently displaced RNNs as the state-of-the-art in several NLP tasks like language modelling (Ma et al., 2019; Wang et al., 2019d), NLI (Yang et al., 2019; Devlin et al., 2018; Radford et al., 2018), question answering (Yang et al., 2019; Devlin et al., 2018), machine translation (Edunov et al., 2018), and sentiment analysis (Yang et al., 2019), among others.

Chapter 3

Word Representations: Engineering, Learning, and Evaluation

Language disguises the thought; so that from the external form of the clothes one cannot infer the form of the thought they clothe, because the external form of the clothes is constructed with quite another object than to let the form of the body be recognized.

Ludwig Wittgenstein, *Tractatus Logico-Philosophicus*

This chapter aims to provide a panoramic view of word representations as they are currently employed in the NLP field, from the way they are constructed and evaluated, to the way the different models compare to each other in terms of their underlying architecture and their empirical performance. The computational processing of natural language requires dealing with numeric representations of language, for which words are commonly used as the base granularity due to the ease of processing them as discrete tokens which are also semantically complete structures. Ideally, computational representations of words should reflect lexical, semantic and syntactic information, expressed in terms of a finite set of numeric **features** that allow this information to be processed and transferred to solve different applications. Different methods and techniques have been proposed to **construct** or **learn** useful numeric representations from these sources of information.

As discussed in section 2.2, mapping text to numbers is an inherently lossy process. Nevertheless, information loss can be beneficial in the construction of representations, since it can filter out unnecessary information. Given that some of the information in these representations might be useful for one task but not for another, evaluating the informational content in word representations is not straightforward and remains an open research question. In this chapter I present the main approaches to the construction and learning of numeric word representations, as well as a set of baseline word representation models that are representatitve of these approaches. I also introduce the base experimental framework that will be used for the remainder of this thesis. The empirical evaluation portion of this framework comprises a suite of some of the most relevant metrics used to evaluate word representations. These metrics, which include an analysis of word pair distances that I proposed, strive to provide an insight into the informational content of different word representations, as well as their relative strengths and shortcomings.

3.1 Sources of Lexical Information

Word representation models have commonly relied on three main sources of lexical information: **morphology**, **lexicons**, and **syntactic context**.

Linguistic morphology is the study of the internal structure and formation of words.[1] This refers to the information that is contained in the *surface form* of a word, i.e. the sequence of textual characters that represent a word. Morphological information can be linguistically informed, such as *syllables*, which are determined by the *sonoricity* of spoken words, and *morphemes*, which are units of *grammatical function*. As an example, the word `majority` contains the syllables `ma`, `jor`, `i`, and `ty`. The same word has a morpheme decomposition into a *stem* `major` and a suffix `-ity`. Alternatively, morphological information can also come from arbitrarily constructed units, such as character n-grams, e.g. the non-overlapping bigrams `ma`, `jo`, `ri`, and `ty`. Morphological units can also be learned word segments, such as those produced by the Byte Pair Encoding (BPE) (Sennrich et al., 2016b) or the

[1]The definition of morphology used here is shared by authors like Kornai (2001) and Aronoff and Fudeman (2011), but differs from the more restricted concept of morphology as studying the *minimal meaningful units of language* presented by Kracht (2007).

WordPiece (Wu et al., 2016a) algorithms. BPE iteratively merges the most frequent pairs of character n-grams (*bytes*) into a single symbol that gets added to the vocabulary. The WordPiece model iteratively learns a language model over the *word units* in a vocabulary and adds the most likely word unit that results from combining two units from the current vocabulary. For a word like `organophosphorus`, which is likely to be rare and therefore not included in a word-level vocabulary, the BPE segmentation could be `_organ`, `oph`, `osph`, and `orus`, while its WordPiece segmentation might be `organ`, `##op`, `##hos`, `##ph`, and `##orus`. These morphological word segmentations will be referred as **subwords** for the remainder of this discussion. A final source of morphological information that is not dependent on subwords is what I will refer to as *word form*, which uses a small set of symbols to construct a prototypical surface form for words. These replacements can include replacing lowercase letters with `x`, uppercase letters with `X`, and numeric characters with `#`, which represent `Apple` with the word form `Xxxxx`, and `USB3.0` with `XXX#.#`. Morphological information can help capture regularities between words (e.g. common suffixes), minimise the difference between misspelled words (e.g. `ma` · `jo` · `ri` · `ty` and `ma` · `yo` · `ri` · `ty`), and can help deal with words that are not in a predefined vocabulary.

Lexicons in general refer to *databases* of lexical knowledge that may contain semantic, syntactic, and relational information about words.[2] **Semantic lexicons**, which are frequently used in NLP, are dictionary-like resources that store words together with their corresponding semantic information. Semantic lexicons can store information such as definitions, semantic relations between units like synonyms, meronyms, hypernyms, or hyponyms (e.g. the WordNet lexical database (George A. Miller, 1995) shown in figure 3.1), *semantic frames* (e.g. the FrameNet database (Baker et al., 1998) shown in figure 3.2), or *common-sense* conceptual relations (e.g. ConceptNet (Speer et al., 2017)). The lexical knowledge in lexicons pertains to the words themselves, irrespective of their context of use, which can be

[2]Lexicons are technically made up of *lexical units*, which include root words, compound words, and affixes, but for ease of exposition I will refer to these as *words* for the remainder of this discussion.

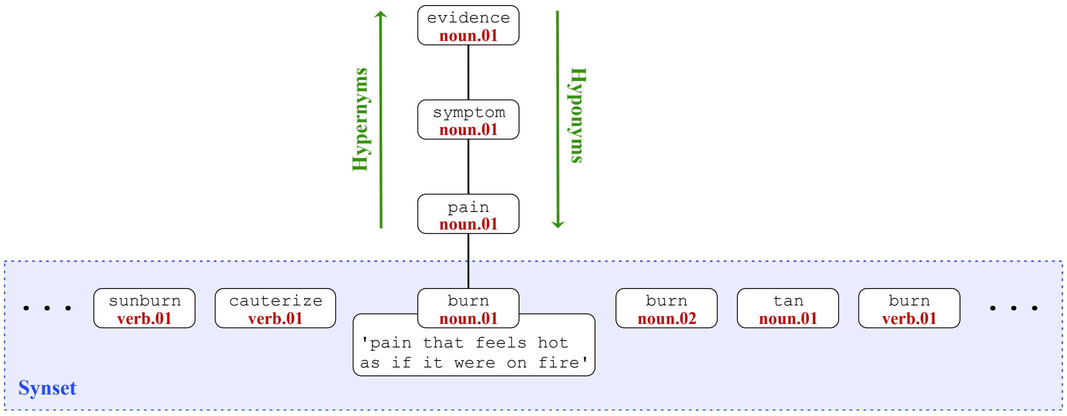
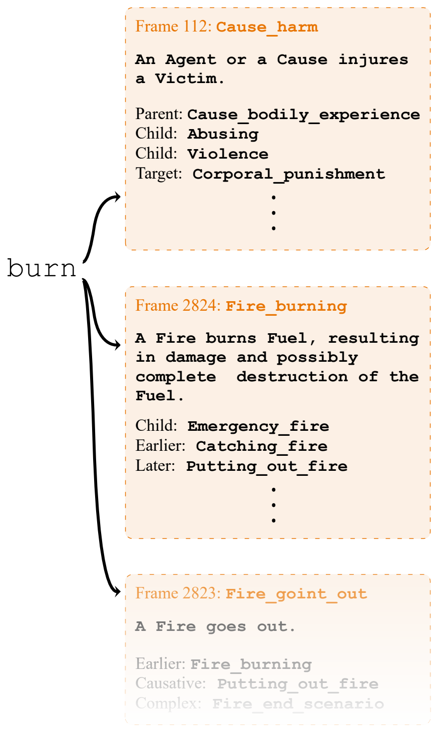


Figure 3.1: WordNet knowledge base synsets and hypernyms for the word `burn`



burn
Frame 112: Cause_harm
An Agent or a Cause injures a Victim.
Parent: Cause_bodily_experience
Child: Abusing
Child: Violence
Target: Corporal_punishment
Frame 2824: Fire_burning
A Fire burns Fuel, resulting in damage and possibly complete destruction of the Fuel.
Child: Emergency_fire
Earlier: Catching_fire
Later: Putting_out_fire
Frame 2823: Fire_goint_out
A Fire goes out.
Earlier: Fire_burning
Causative: Putting_out_fire
Complex: Fire_end_scenario

Figure 3.2: FrameNet knowledge base frames for the word `burn`

helpful when constructing transferable word representations that can be used across NLP tasks spanning multiple linguistic domains.

Context provides invaluable information about a word's semantics and usage. From the perspective of *structuralist semantics* (as described by Gasparri and Marconi (2019)), the meaning of a word is defined in terms of the role of that word within the structured system of language, as well as the relations it maintains with other words. Within the structuralist linguistic tradition, the *distributional hypothesis* (Harris, 1954), which will be discussed in more detail in chapter 4, posits that a word's context, understood as a word's *neighbourhood* of surrounding words, is informative enough to infer the meaning of that word. As an example, in the sentences `She took her dog out for a walk.` and `She took her puppy out for a walk.`, the fact that `dog` and `puppy` appear in the same context should give some notion of semantic proximity between the two words. Contextual information is derived from the statistical usage of a word in real world language, and is therefore not necessarily aligned with general linguistic consensus. In theory, by the law of large numbers, the *universal* meaning of a word could be extracted by processing all of the contexts in which that word has been used. In practice, however, this information is constrained by the size, style, and diversity of the language corpus from which these meanings are extracted. The contextual information for the word `light`, for example, would differ substantially when extracted from a Physics textbook as opposed to a novel or an Art History textbook. Unlike morphological and lexical information, contextual information does not provide *hard* knowledge about a word's meaning, but the more *nuanced* information obtained from context is able to accommodate diverse uses of language and treat a word's semantics as an evolving continuum. The dependence of contextual information on specific language corpora can also provide domain-specific information which can be desirable for certain NLP applications.

3.2 Word Representation Engineering and Learning

To construct useful representations, the information about a word's morphology, semantic relations, and context needs to be processed, selected, contextualised, pruned, and transformed into usable **features**. A feature in this context refers to a single unit of information that is used to build a representation. For the purpose of this discussion, a representation can be understood as a vector of numbers, where every dimension of the representation vector is referred to as an individual feature. There are a wide variety of representational approaches that convert the aforementioned sources of linguistic information into the features used to represent a word.

Representational approaches can be split into two broad categories: **feature engineering**, which leverages **expert knowledge**, such as morphological and lexical knowledge from pre-existing resources, and **feature learning**, which refers to the process of **statistically tuning** or selecting optimal features for a specific task.

3.2.1 Feature-Engineered Word Representations

Feature engineering has been informally defined in different ways in the literature and is frequently conflated with the concepts of *feature extraction*, *feature transformation*, *feature encoding*, *basis functions* etc. (Goldberg, 2017). This term has commonly been used to refer to a preprocessing step by which the features from the input data are modified in different ways (e.g. normalising to get all values for a feature in the range of $[0,1]$, or $[-1,1]$ with a mean of 0). In this discussion, however, feature engineering is taken as the process of **hand-crafting features out of domain knowledge**. This aligns with the *componential* view of lexical semantics (as described by Gasparri and Marconi (2019)) which describes word meaning in terms of *semantic components*, e.g. the word `leg` can be described by semantic primes such as `IS_ENTITY`, `IS_PART_OF`, `IS_NOUN`, etc.

A **feature-engineered word representation** extracts features from the lexical information that is deemed to be most relevant to solve a particular task. This process typically requires heuristics and fine-tuning to select a useful combination of features. I lay out one possible way to engineer an initial set of features for words in the following algorithm:

1. Pre-select lexical knowledge resources to use

2. Define a word vocabulary

3. For every word in the vocabulary, query the lexical resources to obtain its lexical information, e.g. POS tags, hyponyms, word category, etc.

4. Look up the existing feature dictionary for every extracted piece of information, if the piece of information is not already in the feature dictionary add it as a new feature

5. Extract relevant morphological information: full word, subwords, word form, character n-grams, etc.

6. Look up extracted morphological information in the feature dictionary, add new features when appropriate

7. Given the full dictionary of features, prune any feature that appears in fewer words than a set threshold

Feature-engineered word representations are vectors of numbers where every dimension corresponds to a feature from the feature dictionary, and its value can indicate the presence or absence of the feature (0 or 1), the feature counts (e.g. how many times a subword appears in a word), a feature's normalised frequency, etc. An example of these features is provided in table 3.1. Since feature dictionaries tend to be large, oftentimes containing tens to hundres of thousands of features, and words are commonly represented by only a small number of those features, these vector representations have a high dimensionality and tend to be **sparse**, i.e. made up mostly of zeros. Given the sparsity of these vectors, and the fact that some of the features will only contain non-zero values for a small number of words, it is common for these representations to undergo a **feature selection** process. Feature selection consists of both the **selection of information sources** from which the features are obtained, as well as the **pruning** strategy used to reduce the size of the feature dictionary.

Feature selection is an important step for computational reasons, since very high-dimensional feature vectors are more computationally expensive to process. However, reducing the number of features can also have the counterintuitive effect of making the representations more informative by removing what Manning et al. (2008) describe as *noise features*. Even though the individual features might be grounded in sound linguistic theory, the combination of such features that is most effective in terms of solving an NLP task can vary substantially from task to task. This can be attributed to the fact that representations, in the context of machine learning, are not processed in terms of their individual features, but rather

Table 3.1: Example of manually constructed feature vectors for the words in the phrase `Calling number 8`. Features include bi-grams, positional information (first or last word), suffixes, and whether the word is numeric

	uppercase	SUFFIX_ing	numeric	<START>_calling	calling_number	number_8	8_<END>
Calling	1	1	0	1	1	0	0
number	0	0	0	0	1	1	0
8	0	0	1	0	0	1	1

as a complex interaction of all of these features. Such interactions between large numbers of features are mostly incomprehensible for humans, so selecting effective combinations of features requires resorting to heuristics, fine-tuning, and empirical validation.

3.2.2 Learned Word Representations

Feature engineered representations have the downside of being high-dimensional, sparse, and requiring expert knowledge. **Data-driven word representations** approaches offer an alternative, where representations are learned by exploiting lexical information (e.g. word context distributions) from **patterns of word repetition** in data. Learning word representations from data entails a shift in the representational paradigm, where words are no longer represented by high-dimensional sparse vectors of discrete and interpretable features, but rather by dense vectors of **latent features**, commonly referred to as distributed representations, as described in chapter 2.

Distributed representations of text are typically implemented as **semantic proximity models**, which Deerwester et al. (1990) describe as models that construct a *semantic space* where similar items are placed in close geometric proximity. These models capture general patterns from the data across all of their latent features, which means that the individual features of these representations are no longer interpretable. However, this loss of interpretability can be beneficial when constructing representational similarity. As mentioned in the previous section, feature vectors tend to be very sparse, which can result in a lot of the representations being orthogonal (i.e. they share no non-zero elements) which would be interpreted as maximal dissimilarity. As an example, the sentences `The press was inquisitive` and `Reporters were asking questions` are entirely orthogonal in terms of their bag-of-words representations given that they have no words in common, even though they are semantically very similar. Distributed representations, on the other hand, provide a spectrum of similarity between representations in terms of their geometric distance, where this distance is a function of the statistical similarity between the represented texts.

Distributed word representation learning most commonly relies on two sources of statistical information: **word-document** and **word-word co-occurrence counts**. Word-document statistics are usually described in terms of a matrix $\mathbf{X}_{w\text{-}d} \in \mathbb{R}^{V \times D}$, where V is the size of the vocabulary and D is the number of documents in the dataset. In this matrix, every row corresponds to a word in the vocabulary and every column corresponds to a document, where each cell $[\mathbf{X}_{w\text{-}d}]_{ij}$ contains the counts[3] for word i in document j.

Word-document matrices are commonly used in the context of **topic modelling**, which is the process of extracting abstract topics from text data. Latent Semantic Indexing (LSI) as presented by Deerwester et al. (1990), also found in the literature as Latent Semantic Analysis (LSA), is an early topic modelling method that uses Singular Value Decomposition (SVD) to factorise the word-document matrix $\mathbf{X}$ into three matrices $\mathbf{X} = \mathbf{W}\Sigma\mathbf{D}^\top$, where $\mathbf{W}$ and $\mathbf{D}$ are two orthogonal matrices and Σ is a diagonal matrix that conains the set of *singular values*. In SVD, the largest singular values account for the most variance in the original matrix. A rank-k approximation $\mathbf{X} \approx \hat{\mathbf{X}} = \mathbf{W}_k\Sigma_k\mathbf{D}_k^\top$ minimises the information loss from the original matrix by only preserving the k largest singular values. Every row of the resulting matrices $\mathbf{W}_k$ and $\mathbf{D}_k$ can be interpreted as representing a single word and document, respectively, in terms of *abstract topics*. The ideas from LSI were further expanded to consider that documents can not only come from one of these abstract topics, but rather from a distribution over topics in the probabilistic Latent

[3]The cells of word-document matrices can alternatively contain *presence-absence* (0 or 1), normalised frequencies, or Term Frequency - Inverse Document Frequency (TF-IDF) scores of the words in a document.

Semantic Indexing (pLSI) model (Papadimitriou et al., 1998). More recently, Blei et al. (2003) proposed the Latent Dirichlet Allocation (LDA) topic model, a generative model of topic distributions in documents that assumes that a document comes, not from a single sampled topic, but rather from a distribution over topics, meaning that a document comes from multiple topics and every word in the document is obtained by conditioning on a topic sampled from that topic distribution. Even though these models have been mostly used to produce document representations for tasks like text classification, they also produce distributed word representations in the same process which, like the learned document representations, are expressed in terms of abstract topics extracted from the data. The *lda2vec* model[4] proposed by Moody (2016) applies the topic modelling approach from LDA to explicitly learn distributed word, document, and topic representations that live in the same embedding space, such that words or documents that are closely associated with a topic will be embedded close to the embedding of the corresponding topic.

Word-word co-occurrence models attempt to capture information about a word's context, understood here as its neighbouring words. A word's neighbourhood is most commonly implemented as a context window of fixed length C, meaning the C words appearing directly before and after a centre, or *focus*, word. Therefore, two words co-occur if they appear within each other's context windows. Word-word co-occurrence can be recorded for the full dataset or corpus into a single word co-occurrence matrix $\mathbf{X}_{w\text{-}w} \in \mathbb{R}^{V \times V}$, where V is the size of the vocabulary. In this matrix, every row and column correspond to a word in the vocabulary and every cell $[\mathbf{X}_{w\text{-}w}]_{ij}$ contains the counts[5] (or frequencies) for word w_i and word w_j.

Unlike topic modelling approaches which focus on document representations, word-word co-occurrences are typically exploited in the learning of word representations, as they contain no document-level information. As mentioned in chap-

[4]lda2vec is modelled after the Skip-gram model, but only uses a single embedding unit, and uses the idea of continuous Bag-of-Words (CBOW) and vLBL of constructing a context vector by combining (averaging or summing over) the individual embeddings of the words that make up the context.

[5]Word-word co-occurrence is can also be weighted by the distance between two co-occurring words, e.g. in the sentence `the cat sat`, even though `the` and `sat` co-occur, they can be recorded as co-occurring less strongly than `the` and `cat`.

ter 2, these statistical distributed representations of words are most commonly referred to as **word embeddings**. An early example of word embeddings based on these word co-occurrence matrices is the hyperspace analogue to language (HAL) model (Lund and Burgess, 1996), which uses PCA to obtain the M principal components of a word co-occurrence matrix[6] to produce word embeddings. The word co-occurrence matrix used by HAL is constructed by placing the previous (left) context of a word in the row for that word and the future (right) context in the column for that word, where co-occurrence is weighted by context distance, and the full matrix is constructed by concatenating the rows and columns for every word, hence producing vectors of size $2V$. Lebret and Collobert (2014) propose a similar approach, where the Euclidean distance is replaced with the Hellinger distance, which measures the difference between two probability distributions P and Q and is calculated as $Hellinger(P,Q) = \frac{1}{\sqrt{2}}||\sqrt{P} - \sqrt{Q}|| = \frac{1}{\sqrt{2}}\sqrt{\sum_i \sqrt{p_i} - \sqrt{q_i}}$, in the PCA processing of word co-occurrence *probability*[7] matrix to produce **HellingerPCA** word embeddings.

The **GloVe** model proposed by Pennington et al. (2014), arguably one of the most widely used word embedding models to this day, gets its name from training on a *global* word co-occurrence matrix. Pennington et al. (2014) use the term *global* to distinguish corpus-level word co-occurrence from the *local contexts* used in other models such as Word2Vec (Mikolov et al., 2013b,a). Given that cells in this matrix contain word-word co-occurrence counts, in this model the probability that a word w_j appears in the context of word w_i is expressed as the normalised counts $p(w_j|w_i) = \frac{[\mathbf{X}_{w\text{-}w}]_{ij}}{\sum_{k\in\mathcal{V}}[\mathbf{X}_{w\text{-}w}]_{ik}}$. The GloVe model makes the assumption that, given two words w_i and w_j, and a *context* word w_k, the ratio of the probabilities for each word given the context $\frac{p(w_k|w_i)}{p(w_k|w_j)}$ provides information about the relationship between words w_i and w_j with respect to word w_k. A large ratio implies that w_k is closely related to w_i, but not to w_j, a small ratio implies the opposite, and a ratio close to 1

[6]Lund and Burgess (1996) report the best empirical performance when using the $M = 100$ or $M = 200$ principal components of the word co-occurrence matrix.

[7]The value of every cell in this matrix is the number of times a word appears in a given context, divided by the number of times any word appears in that context, hence it is described as the probability of a word given a context.

means that w_k is as related (or unrelated) to w_i as it is to w_j. These ratios are then expressed in terms of a word embedding vector $\mathbf{u}_x = [\mathbf{U}]_x$, and a *context embedding* vector $\mathbf{v}_x = [\mathbf{V}]_x$:

$$\frac{p(w_k|w_i)}{p(w_k|w_j)} = \frac{F(\mathbf{u}_i^\top \mathbf{v}_k)}{F(\mathbf{u}_j^\top \mathbf{v}_k)} \tag{3.1}$$

where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{V \times M}$ are the word and context embedding matrices, respectively, V is the vocabulary size, M is the dimensionality of the embeddings, and F is an encoding function.

By approximating this ratio of probability distributions, the embeddings **u** and **v** are expected to capture the semantic information in the co-occurrence matrix, where the resulting embedding for a word w is the sum of its two embeddings: $\mathbf{w} = \mathbf{u}_w + \mathbf{v}_w$. The embeddings are learned by optimising a weighted least squares cost function:

$$J = \sum_{i,j=1}^{V} f([\mathbf{X}_{w\text{-}w}]_{ij})(\mathbf{u}_i^\top \mathbf{v}_j + b_i + \hat{b}_j - \log[\mathbf{X}_{w\text{-}w}]_{ij})^2 \tag{3.2}$$

where b_x and $\hat{b}_x$ are bias terms, and f is the following piecewise weighting function:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

where $x_{\max}$ is a cutoff point, and α controls the curvature of the function before the cutoff point. This weighting function deals with the zeros in the co-occurrence matrix, since the logarithm of zero is undefined, and prevents overweighting very frequent and very rare co-occurrences.

Models that, like GloVe, use two interacting and independently varying linear units are referred to as **bilinear models**. Bilinear models have been commonly used in language modelling-based word embedding learning. More specifically, bilinear models have been used as an efficient solution to **neural language models**[8] (Bengio et al., 2003), which describe the probability of a *target word* w appearing after

[8]Neural language models will be discussed in more detail in chapter 4.

a *history* of $t-1$ words $h = \langle w_1, w_2, \ldots, w_{t-2}, w_{t-1} \rangle$ in terms of a parameterised function of the word and history $f(w,h;\Theta)$. This next word prediction objective is a type of multi-class classification problem, where the target is a word from the full vocabulary $\mathscr{V}$, which can be expressed as a softmax function:

$$p(w \mid h) = \frac{\exp(f(w,h;\Theta))}{\sum_{w' \in \mathscr{V}} \exp(f(w',h;\Theta))} \tag{3.3}$$

In bilinear models, the function f describes the relationship between the elements in a conditional distribution, which in this case are the word w and the history h, in terms of an interaction (typically a dot product) between their respective distributed representations. The original *log-bilinear language model* proposed by Mnih and Hinton (2007) uses a single embedding matrix $\mathbf{U} \in \mathbb{R}^{V \times H}$, where V is the size of the vocabulary, H is the embedding dimension, and $\mathbf{u}_x = [\mathbf{U}]_x \in \mathbb{R}^H$ is the embedding vector for word x which corresponds to a row in the embedding matrix. The embedding matrix $\mathbf{U}$ is used to embed the target word $\mathbf{u}_w$ and the embedding for the history is given by summing over the embeddings of its individual words[9] $\mathbf{h} = \sum_{i=1}^{t-1} \mathbf{u}_i$. Mnih and Teh (2012) extend this log-bilinear language model by adding a second embedding matrix $\mathbf{V} \in \mathbb{R}^{V \times H}$ used to embed the target word $\mathbf{v}_w = [\mathbf{V}]_w \in \mathbb{R}^H$, and the *history embedding* is calculated by a weighted sum over the embeddings of the words in the history $\mathbf{h} = \sum_{i=1}^{t-1} \mathbf{C}_i \mathbf{u}_i$, where the weighting matrix $\mathbf{C}$ contains position-dependent context weights, and the word embeddings come from the first embedding matrix $\mathbf{u}_x = [\mathbf{U}]_x \in \mathbb{R}^H$.

By extending the idea of a word's *history* to the more general concept of a word's *context*, bilinear models can be applied to word co-occurrence statistics. The language modelling objective then becomes predicting a **focus word**[10] f given its **context** $\mathscr{C}$ of co-occurring words. The **Skip-gram** model proposed by Mikolov et al. (2013a) flips the language modelling objective around and seeks to predict each of the context words $w_c, c \in \mathscr{C}$ individually given the focus word w_f:

[9]The formulation by Mnih and Hinton (2007) includes an *interaction matrix* as well as additional bias terms which are omitted here for ease of exposition and to allow for a clearer comparison with other models.

[10]The change in terminology, from *target word* to *focus word* is done to keep the discussion consistent with the terminology used in the literature.

$$p(w_c \mid w_f) = \frac{\exp(\mathbf{v}_c{}^\top \mathbf{u}_f)}{\sum_{w \in \mathscr{V}} \exp(\mathbf{v}_w{}^\top \mathbf{u}_f)} \tag{3.4}$$

where $w \in \mathscr{V}$ is a word from the vocabulary, and $\mathbf{u}_w$ and $\mathbf{v}_w$ are the input and output embeddings for word w, respectively. The input embeddings $\mathbf{u}_f$, i.e. the embeddings for the focus words, are then used as the learned word embeddings. The Skip-gram model is described in more detail in chapter 4. Mikolov et al. (2013a) also proposed the CBOW model, which predicts a focus word from a context embedding calculated as the average of the embeddings of the words appearing in the context $\mathbf{c} = \sum_{c \in \mathscr{C}} \mathbf{u}_c$. However, since the CBOW model uses a single embedding matrix it is not included in this discussion on bilinear models.

Despite their different architectures and training regimes, these word co-occurrence models process the same information. Levy and Goldberg (2014) show that both the Skip-gram (Mikolov et al., 2013b) and *log-bilinear noise-contrastive estimation (NCE)* (Mnih and Kavukcuoglu, 2013) models can be cast as a weighted factorisation of word co-occurrence matrices. In their derivation, Levy and Goldberg (2014) describe the matrix factorised by the Skip-gram model as the *Pointwise Mutual Information (PMI)* matrix, and the one factorised by the log-bilinear NCE model as a log-conditional-probability matrix, both of which are defined in terms of word co-occurrence counts.

Word co-occurrence neural language models remain among the most widely used word embedding models in NLP pipelines. Fine-tuning and understanding the linguistic and informational properties of this family of word embedding models remain active areas of research.

3.3 Word Representation Evaluation

There is no consensus as to what constitutes an *ideal* word representation. Despite the lack of an objective and universal evaluation metric, certain general characteristics can help determine the quality of word representations. There are two main outlooks to measure these representations: the *linguistic* and the *pragmatic*. The linguistic aspect refers to the **syntactic and semantic information** that is captured by these representations. The pragmatic aspect focuses on how useful these representations are in terms of **solving concrete tasks** or applications. These aspects conform what Schnabel et al. (2015) describe as the **intrinsic** and **extrinsic evaluation** of word representations, respectively.

There is no single way to empirically evaluate the intrinsic characteristics of word representations. For this reason, specific aspects of these representations have been used to probe for syntactic and semantic information. The most widely used **intrinsic evaluation metrics** gauge the internal linguistic structure of the semantic space by focusing on one or more of the following aspects:

- **Semantic relations in the representation space**, analysed in terms of how closely the geometric properties of the space mimic human notions of semantic relations. This aspect of the representation space has been evaluated through the correlation between human-assigned similarity scores between pairs of words and the geometric distance between their corresponding (vectorial) representations. This evaluation requires a dataset of human annotated similarity scores between pairs of words, such as the word similarity (and word relatedness) datasets created by Finkelstein et al. (2002), Hill et al. (2015), Gerz et al. (2016), among others.

- **Coherence of the semantic space**, described by Schnabel et al. (2015) as the property of the representation space that places semantically related words within a close vicinity of each other. Unlike the preservation of semantic relations, which looks at relations between pairs of words, the coherence of the space analyses local *neighbourhoods* (i.e. clusters) of words in representation space. As a way to measure this, Schnabel et al. (2015) propose an *intruder detection* task, where given a query word, two related words, and an *intruder* word, in a coherent representation space the two nearest neighbours to the query word should be the related words, and the intruder should appear outside of that neighborhood. Other evaluation metrics that focus on semantic space coherence include analysing the *stability* of word representation models in terms of the overlap between word neighbourhoods in different representation models (Wendlandt et al., 2018), or the *spatial distribution* of word classes (Sasano and Korhonen, 2020), among others.

- **Geometric mapping of linguistic properties** consists of probing the representation space for a specific set of linguistic features. One potential way to measure this is by considering linguistic properties as canonical translations in representation space. An example of this type of evaluation is the *analogical reasoning* test proposed by Mikolov et al. (2013b), where the relation between words a and b is analogous to that of c and d, i.e. a is to b as c is to d, and the test is to determine whether this relations hold in representation space by performing vector algebra on the word representations. As an example, consider the relation `man` : `king` :: `woman` : `queen`, the vector representation $\mathbf{u}_{\texttt{queen}}$ should be close to the result of $\mathbf{u}_{\texttt{king}} - \mathbf{u}_{\texttt{man}} + \mathbf{u}_{\texttt{woman}}$. Alternatively, linguistic features can be evaluated at the lexical level, as done by *QVEC* (Tsvetkov et al., 2015), an evaluation scheme that compares every word representation with a *linguistically ideal* representation[11] for the same word (i.e. a vector of canonical linguistic features for that word), and the quality of a word representation is measured by how well the individual dimensions of the word representation can be aligned with the features in the canonical representation.

- **Semantico-spatial ambiguity**[12] concerns the concurrent mapping of multiple senses of a word to the representation space. Even though, as argued by Yaghoobzadeh and Schütze (2016) and Gladkova and Drozd (2016), word representations should ideally capture all senses of a word, capturing and measuring these senses can be a considerable challenge, especially given that certain senses might be overrepresented in a dataset. As exemplified by Gladkova and Drozd (2016), depending on the domain of the text data,

[11]Wang et al. (2019c) and Bakarov (2018) stress the importance of *multifacetedness*, i.e. the incorporation of linguistic aspects such as morphological or syntactic properties, into the construction of word representations.

[12]Semantico-spatial ambiguity is closely related to the idea of *meaning conflation* discussed by Camacho-Collados and Pilehvar (2018).

the dominating sense for the word `Apple` might be `fruit` rather than `corporation`. Evaluation methods have been proposed to gauge how word representations capture polysemy, such as the analysis of the nearest neighbours of polysemous words (Athiwaratkun et al., 2018), or the human-assigned contextual similarity scores proposed by Huang et al. (2012). These methods, however, can be thought of as being mostly qualitative and have not been widely adopted in the NLP community. Quantitatively evaluating the capture of multiple senses in word representations remains an open research question. In regards to this, Gladkova and Drozd (2016) suggest to "embrace ambiguity as an intrinsic characteristic of word embeddings".

Evaluating word representations **extrinsically** has a broader definition which can be summarised as the performance gain obtained by using a given set of word representations to represent the input to an **unrelated downstream NLP task**. *Unrelated* in this context pertains to the difference in either data domain or training regime. In other words, these word representations are evaluated in terms of their contribution to solving a task they were not explicitly trained to solve. *Downstream task* refers to a task that is attempted *after* the word representations are fully constructed. The expectation with the extrinsic evaluation process is that the linguistic information captured by the trained word representations is **transferrable** to other NLP tasks and should therefore improve the performance of any model that can benefit from such information. In this sense, any NLP task that takes words as input can be used as an extrinsic evaluation method. While extrinsic evaluation gives a notion of the **practical utility** of a word representation model as a representation technique, the diversity of learning algorithms, training regimes, and model complexities used for different downstream tasks can obfuscate the comparison of different word representations. I argue that a fair extrinsic evaluation metric for word representations should, in principle, have a minimal reliance on the training regime and trainable parameters, and should be based on a task definition that relies as much as possible on the word representations themselves. While this might not be the best indication of the performance of word representations in a wider range of real world applications, it can provide a clearer comparison between representations that is focused on their encoding of semantic features.

Current evaluation metrics focus on a set of informational aspects and properties present in word representations. Each metric on its own offers limited insight into the quality of these representational models. As noted by Tsvetkov et al. (2015), Schnabel et al. (2015), and Chiu et al. (2016), in many cases a model's performance on an extrinsic task does not correlate with its intrinsic evaluation scores, or even with its performance on other extrinsic tasks. The partial views of a word representation model provided by these different evaluation metrics can be combined in what Gladkova and Drozd (2016) describe as an *exploratory approach* to word representation quality in order to better understand the relative strengths and weaknesses of different representational approaches.

3.4 Empirical Comparison of Word Representations

Throughout this research I explore the principles and limitations of different families of word representation models, including widely adopted models, as well as variations and novel models of my own design. In order to gain a better insight into the comparative advantages of these models, I propose the following experimental conditions which are designed to provide an even ground on which to compare different word representations. This relies on keeping as many of the experimental variables fixed as possible to minimise the variability between models. Some of the most important conditions to account for are the **dataset** used to train or construct the models; the **complexity** of the **model architecture**, which in the case of word representations involves, among other things, the **choice of vocabulary**; and finally, the **testing conditions** used to compare these models, which need to be diverse and representative enough to provide a thorough and detailed evaluation of the different models. These experimental conditions remain constant throughout the experiments presented in this thesis.

3.4.1 Dataset

The BNC is designed to be representative of British English usage and consists of transcribed speech and written text produced between 1960 to 1993. The data is obtained from a variety of sources such as lectures, bestselling books, news commentaries, arts and science books and periodicals, business meetings, etc. The full XML edition of the BNC comes segmented into 6 million sentences and approximately 100 million tokens, and contains linguistic annotations such as POS tags, head words, multi-words, and speech turns, a sample of which can be seen in listing 3.1. This choice of dataset is meant to provide a diverse set of examples of English usage, in line with the argument made by Lai et al. (2016) that the domain of the corpus can have a significant impact on the resulting representations.

Listing 3.1: BNC XML sample sentence

```
<s n="758">
      <w c5="NP0" hw="dougal" pos="SUBST">Dougal </w>
      <w c5="VBD" hw="be" pos="VERB">was </w>
      <w c5="AJ0" hw="pleased" pos="ADJ">pleased </w>
      <w c5="PRP" hw="with" pos="PREP">with </w>
      <w c5="PNP" hw="it" pos="PRON">it</w>
      <c c5="PUN">.</c>
</s>
```

Due to computational limitations, I reduce the full corpus by randomly selecting 10% of its tokenised sentences, which amounts to 601,818 sentences and 11,126,083 tokens. Sentences in this subset contain between 0 and 1,452 tokens, with an average length of 18.5 tokens. Hereforth, all mentions to the *dataset* will refer to this subset of the full corpus. The use of a small subset of the dataset responds to the need to train large numbers of models on limited computational resources. This dataset of 11 million tokens falls within the range of 10-100 million words that Zhang et al. (2020) describe as being sufficient to encode syntactic and semantic features into language model-based representation models.[13] I do,

[13]Zhang et al. (2020) limit this range to the learning of text representations. They also mention that

Table 3.2: 10 most frequent words in the BNC XML dataset (subset 10%)

Word	Counts	Frequency
`the`	601,852	0.0609
`of`	303,678	0.03074
`and`	261,012	0.0264
`to`	260,550	0.0264
`a`	215,176	0.0218
`in`	194,470	0.0197
`that`	111,673	0.0113
`it`	105,349	0.0107
`is`	99,452	0.0101
`was`	87,992	0.0089

nonetheless, acknowledge the wider consensus that training on larger datasets has been shown to produce better word representations (Pennington et al., 2014; Lai et al., 2016), and leave the exploration of larger scale model training as future work.

3.4.2 Vocabulary

After removing punctuation marks, the 10% subset of the full dataset contains 9,879,226 words[14] and a corresponding vocabulary of 141,044 unique words. This vocabulary is subsequently pruned by removing any word that appears fewer than 5 times in the dataset, resulting in a vocabulary of 45,769 unique words. The pruned vocabulary is made up of 32.45% of the unique words from the original vocabulary, but mainains a coverage of 98.48% of the words in the dataset (9,728,672 out of the full 9,879,226 words). This vocabulary will hereforth be referred to as the **BNC vocabulary**. A sample of the ten most frequent and ten of the least frequent words in the vocabulary are presented in tables 3.2 and 3.3, respectively.

Setting a frequency threshold on the vocabulary serves the double purpose of reducing the complexity of the model,[15] as well as removing very rare words which

Natural Language Understanding (NLU) tasks require considerably more data than representation learning.

[14]The change in terminology from "token" to "word" is meant to make a distinction of the elements before and after cleanup (e.g. removing punctuation), although "word" in this context can also refer to tokenised digits, accronyms, and other elements that are not filtered during cleanup.

[15]Given that many NLP models tend to be represented by matrices, where one of the axes represents every item in the vocabulary, reducing the size of the vocabulary effectively results in decreasing the dimensionality of this matrix, and therefore the complexity of the model.

Table 3.3: 10 of the least frequent words in the BNC XML dataset with a vocabulary cutoff of 5 (subset 10%)

Word	Counts	Frequency
`1/8`	5	5e-07
`easels`	5	5e-07
`cancellations`	5	5e-07
`knelle`	5	5e-07
`klan`	5	5e-07
`deplete`	5	5e-07
`lockable`	5	5e-07
`marmite`	5	5e-07
`patiño`	5	5e-07
`seelig`	5	5e-07

typically include misspellings, acronyms, or overly specialised terms. Even though removing these terms can have a positive effect on the model's performance, the choice of frequency threshold can also have a detrimental impact on model performance, since a high threshold reduces a vocabulary's **coverage** of the dataset. Coverage refers to the proportion of text in the dataset that is covered by a vocabulary, where any word in the dataset that is not part of the vocabulary is labelled as an *unknown* or *out-of-vocabulary (OOV)* token. The number of OOV tokens in a datasets entails a proportional loss of information, especially when considering that rare words can contain fundamental semantic information. As an example, the word `marmite`, from table 3.3, which would get pruned if the frequency threshold was increased to 6, can serve as a strong indication that a piece of text is referring to British food.

The frequency threshold of 5 used for this vocabulary was chosen because it achieves a significant reduction in the computational complexity while preserving an acceptable coverage of the dataset. This vocabulary remains constant throughout the rest of the experiments in this thesis.

3.4.3 Baseline Word Representations

In order to get a view of the different families of representational approaches described in this chapter, I compare a diverse set of word representations. This selection aims to give a broad perspective of empirical advantages and disadvantages of the different word representation paradigms. The word representations delineated here are compared with the new models I propose in chapters 4 and 5.

3.4.3.1 Feature-engineered word representations

I implement these word representations as a presence-absence matrix $\mathbf{F}_{\text{features}}$ where the rows correspond to the words in the vocabulary, the columns correspond to the distinct features from a **feature dictionary**, and cells contain binary values where a 1 indicates the presence of a feature. I designed this feature dictionary to be representative of the **morphological**, **lexical**, and **semantic** information sources for word representations I described earlier. For the morphological features I use **WordPiece** (Wu et al., 2016b) subword units, obtained by using the pre-trained model from the HuggingFace library (Wolf et al., 2020), which obtains the minimal set of word segments that cover a text corpus. A wordpiece can be a full word, if it is frequent enough, or a commonly occurring segment, like `##ing` (where `##` marks the beginning of a word). In this feature dictionary, wordpieces have the prefix `sub_`. For the lexical features I use the **Linguistic Inquiry and Word Count (LIWC)** word level properties (Pennebaker et al., 2015), which provide information about word-level categories such as *Health* (`liwc_health`) or *Causation* (`liwc_cause`). I complement the lexical knowledge with associated *POS tags* (`syns-pos`), related *hypernyms*[16] (`syns-hyper_`), and *synonyms* (`syns-lex_`) queried from the **WordNet** (George A. Miller, 1995) knowledge base. Finally, I extract semantic information from the **FrameNet** knowledge base (Baker et al., 1998), which provides a word's *semantic frame* (`frame_`), i.e. the conceptual structure for a particular meaning of that word, where frames interact with other frames in relations like *cause*, *parent*, *target*, etc. To reduce the dimensionality and sparsity of this matrix, the feature dictionary is pruned by removing any features that are present in fewer than 5 words from the vocabulary, which results in a dictionary of 6,672 features. The number of features by cate-

[16]Hypernyms are more generic (or less specific) terms in a semantic field, such as `animal` is to `dog` or `cat`.

Table 3.4: Number of features per category after pruning the feature dictionary.

WordPiece subwords	3,019
LIWC features	813
FrameNet features	1,073
POS tags	4
Synonyms	40
Hypernyms	1,723
Total Features	**6,672**

Table 3.5: Present features for the words `imbalance`, `match`, and `food`

imbalance	match	food
sub_##balance	frame_Beyond_compare	frame_Food
sub_im	frame_Compatibility	syns-hyper_substance
syns-lex_attribute	frame_Evaluative_comparison	syns-lex_Tops
syns-lex_state	syns-lex_artifact	syns-lex_cognition
syns-pos_noun	syns-lex_change	syns-lex_food
	syns-lex_cognition	syns-pos_noun
	syns-lex_competition	
	syns-lex_contact	
	syns-lex_event	
	syns-lex_group	
	syns-lex_person	
	syns-lex_possession	
	syns-lex_quantity	
	syns-lex_social	
	syns-lex_stative	
	syns-pos_noun	
	syns-pos_verb	

gory are shown in table 3.4 and a sample of word features are provided in table 3.5. Despite the high dimensionality of the resulting feature-engineered word representations, especially when compared to the distributed word embeddings used in this thesis (200-768 dimensions), I avoided any dimensionality reduction on these representations to ensure that all comparisons are made with the feature vectors themselves.

3.4.3.2 Word2Vec Embeddings

Input embeddings from a Skip-gram model (Mikolov et al., 2013a) pre-trained on the Google News dataset, which is made up of around 100 billion words. These pre-trained word embeddings, obtained through the Gensim library,[17] have a dimensionality of 300, and the model vocabulary comprises 3 million distinct tokens.[18] To maintain a fair comparison, these embeddings are used with the BNC vocabulary described in the previous section. Certain words in the pre-trained model are embedded with a zero vector. In the interest of consistency I embed missing words (i.e. words from the BNC vocabulary that do not have an associated embedding in the model) in the same manner.

Even though the similarity-distance correlation scores contain a fixed set of words, the BNC vocabulary is used in these evaluation metrics to construct a single set of words that all models are evaluated on. Therefore, the intersection between the training vocabulary and the similarity-distance correlation word pairs creates the set of scores on which all models can be consistently evaluated. This allows for a like-for-like comparison between pre-trained word embeddings and the new embeddings that are trained for this research.

3.4.3.3 GloVe Embeddings

GloVe word embeddings pre-trained by Pennington et al. (2014) on a dataset made up of the Gigaword5 corpus (4.3 billion tokens) and a 2014 Wikipedia dump (1.6 billion tokens), which are also taken from the Gensim library. Even though pre-trained GloVe embeddings are available in different dimensionalities, to keep experimental conditions consistent the 300 dimensional embeddings are used in these experiments. The vocabulary in this model is made up of the 400,000 most frequent words. As with the Word2Vec embeddings, these embeddings are evaluated on the BNC vocabulary, where missing words are assigned a zero vector embedding.

[17] `https://radimrehurek.com/gensim`

[18] The Word2Vec vocabulary includes multi-word tokens such as `Moroccan_Islamic_Combatant`.

3.4.3.4 HellingerPCA Embeddings

Lebret and Collobert (2014) build their HellingerPCA embeddings from a word co-occurrence matrix constructed from a dataset composed of a Wikipedia dump, the Reuters corpus, and the Wall Street Journal corpus, where the resulting dataset is made up of roughly 1.6 billion words. The pre-trained embeddings are made available in three different dimensionalities (50, 100, and 200) through the author's website.[19] The 200 dimensional embeddings are used for these experiments given that it is the most similar dimensionality to the rest of the models used here. This model has a vocabulary of 178,080 distinct words which results from pruning words that appear fewer than 100 times in the dataset. As with the other models, these embeddings are evaluated on the BNC vocabulary and missing words are assigned the zero vector.

3.4.4 Evaluation

An empirical testbed that allows for a fair comparison of word representation models must evaluate different aspects of these representations under the same conditions. The set of evaluation metrics used in this research is designed to provide a view of the relative advantages of these models with respect to their extrinsic performance and intrinsic properties.

Extrinsic: WMD Document Classification uses Word Mover's Distance (WMD) (Kusner et al., 2015), a distance metric between documents that is calculated by solving a transport problem based on the *Earth Mover's Distance* (Rubner et al., 1998), in which the word distribution of the first document (i.e. its normalised word counts) has to be transformed into the word distribution of the second, where the *travel cost* between two words is the Euclidean distance between their corresponding embeddings. This **extrinsic evaluation** consists of a document classification task solved with a kNN classifier based on the WMD between document pairs. The classification task is performed on the 20 Newsgroups dataset (Joachims, 1997), which contains 18,846 news articles from 20 relatively well-balanced classes (or *news groups*). These experiments are carried out on the 11,314 news articles that

[19] `http://www.lebret.ch/words/embeddings/200/`

make up the training set of this dataset, obtained from the SciKit Learn (Pedregosa et al., 2011) library.[20] The kNN classifier is used with a constant value of $k = 10$. As I discussed in section 3.3, an extrinsic evaluation that is unencumbered by training regimes and parameters can potentially give a more unbiased measurement of the transferable information in a word representation model. For this reason, and given that this task does not rely on any additional training and is based entirely on word representation distances, I use WMD document classification as the extrinsic evaluation task in this experimental framework.

Intrinsic: Similarity-Distance Correlation is the task of correlating the distances between pairs of words with their corresponding human-assigned similarity score, such that a stronger correlation indicates that higher similarity scores correspond to smaller word representation distances. This **intrinsic evaluation** reports the Spearman's rank-correlation coefficient, or Spearman's ρ, which measures the dependence between two ranked variables, for three different word pair similarity datasets. The first is **WordSim353** (Finkelstein et al., 2002), a set of 353 word pairs with a corresponding similarity scores. Agirre et al. (2009) identified some ambiguity between the concept of *similarity* and *relatedness* in the original WordSim353 scores[21] and proposed a split of the dataset into two distinct sets, **WordSim353-Sim** (202 word pairs) and **WordSim353-Rel** (251 word pairs), for similar and related word pairs, respectively. This split is used in these experiments. The second dataset is **SimLex-999** (Hill et al., 2015), a dataset consisting of 999 pairs of words which presents the human annotators with a less ambiguous set of instructions that explicitly require similarity, and not relatedness, to be considered by the annotators. The third dataset is called **SimVerb-3500** (Gerz et al., 2016), which contains similarity scores for 3,500 verb pairs. This combination of datasets is meant to evaluate the presence of different types of word similarity. Any words that are missing from the

[20]`https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html`

[21]The crowdsourcing prompt for the human annotators lent itself to this ambiguity: "Assign a numerical similarity score between 0 and 10 (0 = words totally unrelated, 10 = words VERY closely related) ... when estimating similarity of antonyms, consider them "similar" (i.e., belonging to the same domain or representing features of the same concept), not "dissimilar"."

vocabulary or have a zero vector are removed from the correlation score calculation.

Intrinsic: Word Pair Distance Distributions is an **intrinsic evaluation** metric I designed to gain some insight into the underlying geometric properties of the learned representation space. This evaluation consists of uniformly sampling 1,642 words from the vocabulary as *base words* and getting three additional sets of 1,642 word pairs: **synonyms** for the base words (extracted from the WordNet knowledge base), **contextual** words (i.e. words appearing directly before or after the base word in the BNC dataset), and a set of **random** words (i.e. sampled uniformly from the vocabulary and unrelated to the base words). The distance distribution for every set of word pairs provides a notion of how synonymic and contextual similarity relate to geometric distance in the representation space, with the random word pairs serving as a baseline.

3.5 Results

The four word representation baseline models, **Word2Vec**, **GloVe**, **HellingerPCA**, and **Feature Vectors**, are compared following the suite of evaluation metrics delineated in the previous section. The results for the WMD kNN document classification task, similarity-distance correlation, and word pair distance distributions are presented in this section alongside some interpretation.

WMD Document Classification accuracies, with their corresponding error bounds, are presented in table 3.6. For reference, given that the 20 Newsgroups dataset is well balanced (i.e. similar proportion of samples per class), selecting classes at random would have an expected accuracy of 5%. GloVe achieves the highest accuracy by a significant margin, with the Feature Vectors performing substantially worse than the rest of the models. Word2Vec and HellingerPCA obtain comparable accuracies in the task (within error bounds).

To gauge the effect of the vocabulary size on model performance, a Word2Vec model with the original vocabulary of 3 million tokens is evaluated on the WMD kNN document classification task. The results in table 3.7 show that the vocabulary size has a large impact on extrinsic performance, with the model with the larger vo-

Table 3.6: WMD kNN document classification accuracies on the 20 Newsgroups dataset. *Due to the computational cost of calculating distances for high-dimensional vectors, the results for the feature vectors were calculated on a subset of 2,500 out of the 11,314 articles from the 20 Newsgroups dataset used for the rest of the models.

Model	Accuracy
Word2Vec	62.75% ($\pm$ 0.89)
GloVe	**66.78% ($\pm$ 0.87)**
HellingerPCA	61.79% ($\pm$ 0.90)
Feature Vectors*	40.16% ($\pm$ 1.92)

Table 3.7: WMD kNN document classification results on the 20 Newsgroups dataset - Word2Vec vocabulary comparison (3M vs 45K words)

Model	Accuracy
Word2Vec (3M words)	**79.62% ($\pm$ 0.74)**
Word2Vec voc5 (45K words)	62.75% ($\pm$ 0.89)

cabulary achieving considerably higher accuracies (∼17% increase) than the same model with a reduced vocabulary. The large effect observed after restricting the vocabulary on the performance of pre-trained Word2Vec embeddings points at the informational value of rare words. As discussed in section 2.1.3, rare words can provide information that is especially useful in tasks such as the document classification task on which these results are reported. Therefore, pruning these rare words can be detrimental to a model's performance on these tasks. To prevent the presence or absence of rare words from dominating the comparisons between the models presented in this thesis, model performance will be interpreted in terms of relative performance where all word embedding models restricted to the same vocabulary.

Similarity-Distance Correlation results are presented in tables 3.8 and 3.9 for cosine and Euclidean distances, respectively. A correlation score that is closer to zero indicates a weaker correlation, which in this task is interpreted as a poor capacity to couple semantic similarity with geometric distance in a model's representation space. Larger negative values imply an inverse relation between the similarity score and the geometric distance between word representations, which is what these representations are expected to capture, i.e. more similar words get represented closely.

These results show the inconsistencies between word representation evaluation metrics. In contrast with the extrinsic evaluation, the GloVe model performs worse than Word2Vec in every similarity-distance correlation dataset. This difference in performance between GloVe and Word2Vec is even more apparent when looking at the Euclidean distance scores in table 3.9.

Inconsistencies in correlation scores are most evident on the HellingerPCA model, which obtains the strongest correlation score for WordSim353-Sim with cosine distance, but performs significantly worse than GloVe and Word2Vec on that same dataset when measured with Euclidean distances. Even though Feature Vectors are once again the worst performing in general, they do manage to outperform HellingerPCA on both SimLex-999 scores (cosine and Euclidean), and outperform HellingerPCA and GloVe on SimVerb-3500 with Euclidean distance, despite obtaining the weakest score on SimVerb-3500 with cosine distance. Unlike all other models, which get better scores when calculating Euclidean distances, HellingerPCA obtains weaker scores on Euclidean than on cosine distance, but also seems to be the least affected by the change in distance metric. Some of the differences in the results for Euclidean vs. cosine distances can potentially bear some relation with the shape of the training objective of the different word embedding models being compared. The dot products used in the Word2Vec and GloVe objective functions are more closely related to the cosine distance than the Euclidean distance, while the square root of the squared elementwise difference in the Hellinger distance is more reminiscent of the Euclidean distance. However, this apparent "closeness" between the distance metric being used for the evaluation and the objective function seems to be inversely proportional to the scores observed, where dot product models seem to perform better on Euclidean distances and the Hellinger model appears to achieve better scores when using cosine distance. A more thorough investigation, which falls outside the scope of this research, is required to more fully understand these differences.

Word co-occurrence models (Word2Vec, GloVe, and HellingerPCA) exhibit

Table 3.8: Similarity-distance correlation results with **cosine** distance

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.5807	-0.4200	-0.3241	-0.2490
GloVe	-0.5803	-0.3618	-0.3252	-0.1930
HellingerPCA	-0.6070	-0.1755	-0.1729	-0.1325
Feature Vectors	-0.2277	-0.1600	-0.1969	-0.1219

Table 3.9: Similarity-distance correlation results with **Euclidean** distance

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.7706	-0.6150	-0.4427	-0.3565
GloVe	-0.6865	-0.5657	-0.3708	-0.2274
HellingerPCA	-0.5932	-0.1740	-0.1635	-0.1297
Feature Vectors	-0.4737	-0.1541	-0.2911	-0.2503

a similar trend in their performance across the different datasets: WordSim353-Sim scores are the highest by a large margin, followed by WordSim353-Rel, then SimLex-999, and SimVerb-3500 obtaining significantly worse correlation scores. Feature Vectors differ from this trend in that they perform better on SimLex-999 than they do on WordSim353-Rel.

Word Pair Distance Distributions are shown in figure 3.3 for cosine distances for random, contextual, and synonym word pairs, where the white dot represents the mean of the distribution. For reference, means that are closer to zero indicate that word pairs in that group are, on average, represented closer together. The expectation is that word representation models that encode substantial semantic information will display larger distances in the random word pairs than in contextual and/or synonym word pairs. The distributions of the Word2Vec and GloVe models exhibit a pattern in line with my intuition that random pairs should be the most spread out (i.e. largest distances), synonyms should be closest together in representation space, and contextual pairs should be somewhere in the middle. While HellingerPCA and Feature Vectors also place synonyms closest together, in both cases the distance distributions of random pairs have smaller means than the distributions for contex-

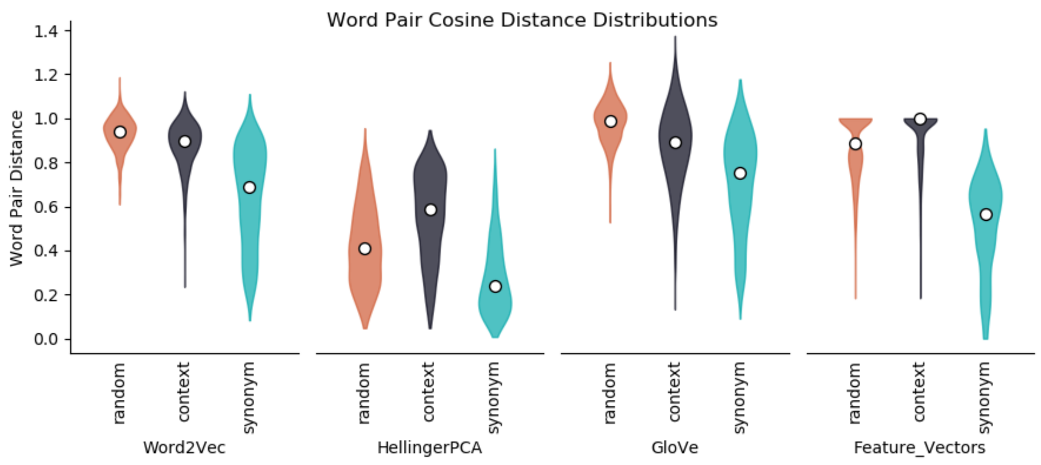


Figure 3.3: Cosine distance distributions between random, contextual, and synonymous word pair sets

tual pairs. These patterns persist when analysing Euclidean distance distributions, but given the sensitivity of Euclidean distance to vector norms, it is harder to draw a clear comparison between models since the different distributions have different scales.

Analysis

One of the most salient conclusions for these experiments is that, despite having desirable properties like interpretability and the use of invariant linguistic knowledge, Feature Vectors seem to consistently underperform data-driven word co-occurrence models across extrinsic and intrinsic evaluation metrics. A possible interpretation is that the continuum of semantic similarity that is facilitated by the dense vectors of real numbers used by Word2Vec, GloVe, and HellingerPCA can more easily accommodate the subtleties of real-world language, as opposed to the strictly binary relations represented by Feature Vectors.

Another possible explanation for the poor performance of Feature Vectors is that contextual information might be critical in the construction of a well-structured semantic representation space. The construction of these Feature Vectors does not involve any source of contextual information, which can also explain why the distances of contextual word pairs are shown to be placed further apart from each other than random pairs in figure 3.3. The negative impact on performance of the lack of contextual information could also explain why HellingerPCA is the worst performing word co-occurrence model, since HellingerPCA displays a similar trend in word pair distance distributions to that of Feature Vectors, where contextual pairs are placed further apart than random pairs.

These results also seem to suggest that differences in training conditions, such as the linguistic domain of the training data, the amount of data that a model is trained on, or the underlying architecture and representation dimensionality, can have unexpected effects on different aspects of these representations. It is still not fully understood why a word representation model can outperform another in a set of tasks, but perform significantly worse in a different task. To understand these differences better, training conditions for different models should remain identical, and different experiments and ablation studies should be performed to measure changes in performance when varying a single experimental variable.

Additionally, the comparison of vocabulary sizes for the same model highlights a commonly overlooked fact: that vocabulary sizes must be accounted for when comparing word representation models. Although it falls out of the scope of this thesis, a thorough analysis of the effects of vocabulary size is required to more fully comprehend the theoretical properties of word representation models.

As evidenced in the experiments in this chapter, existing word embedding models trained on very large datasets such as Word2Vec or GloVe can obtain high scores across several evaluation metrics. However, each individual model can exhibit a significant variation between the scores obtained in different metrics. Additionally, similar models trained on substantially smaller datasets appear to obtain significantly lower scores, even when using similar architectures and training regimes (see Chapter 4). In this research, I hypothesise that by incorporating linguistic knowledge into the learning of word embeddings, models can improve their performance on intrinsic and extrinsic evaluation metrics, as well as reduce the variability in the results obtained. My expectation is that, by making certain linguistic relations more explicit during training, even models trained with relatively small datasets can learn high quality word embeddings. This may have benefits when data is less readily available, such as in specialised or technical domains or low-resource languages. However, identification of, and evaluation with, data from such domains is outside the scope of this thesis.

Chapter 4

Knowledge-Augmented Word Embeddings

A language is not just words. It's a culture, a tradition, a unification of a community, a whole history that creates what a community is. It's all embodied in a language.

Noam Chomsky, *We Still Live Here: Âs Nutayuneân*

Pre-trained word embeddings have become a pervasive input representation strategy in NLP pipelines. Among the most widely used word embedding models are those trained on large corpora of unannotated text using **word co-occurrence statistics**, such as the Word2Vec (Mikolov et al., 2013b) and GloVe (Pennington et al., 2014) models. **Word co-occurrence models**[1] aim to learn vector representations of words such that proximity between two vectors is reflective of the semantic similarity between the words these vectors represent. The expectation of these models is that semantically similar words, such as synonyms, get represented as nearby points in vector space, while the representations of unrelated words fall outside of each other's geometric vicinity.

One of the main advantages of **word co-occurrence models** lies in the efficient and unsupervised nature of their training process, which makes it possible to train these models on very large datasets in a reasonable amount of time and with little to

[1]This family of models is also described by Turney (2010) as **vector space models of word-context**.

no manual intervention (e.g. no need for human-provided annotations). The quality of the learned embeddings is reportedly determined by the size and **linguistic diversity** of the text corpus used to train them, since word embeddings can only learn from the statistical information they see during training. These embeddings will therefore be unable to effectively capture information about linguistic style or usage that is not present, or frequent, in the training data. For example, word embeddings that are trained on news articles will lack information on the linguistic patterns and specialised language that occur in scientific publications, and vice versa.

Word embeddings that capture more diverse uses of language can, in principle, be used more effectively across NLP tasks and linguistic domains. **Linguistic knowledge** can be exploited as a source of invariance to reduce the distance between linguistic domains. Approaches that seek to increase the **transferability** of word embeddings by incorporating linguistic knowledge into statistical representation learning processes have recently become a topic of great interest (Bian et al., 2014; Faruqui et al., 2015; Fried and Duh, 2015). These approaches, however, face an ensuing trade-off between preserving the distributional information and incorporating external knowledge. In this chapter, I propose **knowledge-augmented distributional learning**, an approach that addresses this tradeoff by injecting linguistic knowledge into the statistical learning of word embeddings through a **synthetic text data augmentation** regime that is tailored to fit the statistical learning objective. This data augmentation approach is able to incorporate external knowledge while retaining the distributional information of the original data. The knowledge-augmentation approach presented in this chapter focuses exclusively on those linguistic relations that are considered invariant according to the definition provided by Keenan and Stabler (2010), i.e. those those relations which preserve the semantic structure. In particular, the experiments carried out in this chapter focus on synonyms, since they represent a linguistic relation that preserves semantic and functional role by definition. The embeddings learned under this new knowledge-augmented paradigm capture semantic information from real language data, while at the same time enforcing lexical relations that are invariant across linguistic styles, but might not be fully represented in the training data.

4.1 Related Work

Despite the most recent shift towards Transformer and RNN language models for pre-trained text embeddings (as described in chapter 3), word co-occurrence models are still widely used in NLP pipelines for NLI (Kim et al., 2017), machine translation (McCann et al., 2017), paraphrase generation (Egonmwan and Chali, 2019), text classification (Wu et al., 2018; Wang et al., 2018c; Cheng et al., 2018), among others. The success of word embedding models like Word2Vec (Mikolov et al., 2013b) and GloVe (Pennington et al., 2014) has prompted a large body of follow-up research that attempts to understand the informational characteristics of these models, as well as their limitations and potential ways to overcome them. The knowledge-augmented distributional learning of word embeddings proposed in this chapter lies at the intersection of two research avenues: knowledge injection into word embeddings, and textual data augmentation.

4.1.1 Related Work on Knowledge Injection

This family of approaches incorporates information from external human-constructed knowledge bases into a statistical learning process. The knowledge-augmented distributional learning technique I propose in this chapter is related to this family in that it incorporates external sources of linguistic knowledge into the training objective. The existing approaches to integrate knowledge into the distributional learning of word embeddings can be broadly categorised into two main families:

Semantic Specialisation [2] Given a set of pre-trained word embeddings, this approach transforms (or *fine-tunes*[3]) the embeddings to reflect specific relations extracted from a knowledge base or ontology. The transformations performed on these embeddings are commonly constrained to prevent *catastrophic forgetting* (Good-

[2]Term taken from Mrkšić et al. (2017)

[3]This family of approaches has also been called *model fine-tuning* (Vulić et al., 2018), however this term might be confused with the concept of fine-tuning used in the BERT models (Devlin et al., 2018), where the model learns a general-purpose text embedding which is subsequently specialised to solve a specific NLP task.

fellow et al., 2015), a phenomenon whereby subsequent training of a model results in washing away the distributional information contained in the original embeddings. Faruqui et al. (2015) first proposed the *retrofitting* post-processing approach, which pulls pre-trained vectors closer together in embedding space if their corresponding words appear in a relationship in the knowledge base, while minimising the distance they move from their original position. Mrkšic et al. (2016) and Mrkšić et al. (2017) build upon the ideas from retrofitting with a framework they call *Attract-Repel*, where they use synonyms extracted from a knowledge base to bring word embeddings closer together (attract), but they also introduce antonym relations which are used to push word embeddings apart (repel), all of this while maintaining a vector space preservation (regularisation) term. More recent work by Vulić and Mrkšic (2018) extends the Attract-Repel framework to consider lexical entailment, or *hyponymy-hypernymy* (is-a) relations, which unlike synonymy and antonymy is an asymmetric relation that is enforced by setting a hierarchy over word vector norms (e.g. the norm for the more general concept of `animal` is larger than the norm of the vector for `dog`, which in turn is larger than the norm of `terrier`). Vulić et al. (2018) continue this line of work by learning a general mapping function from an Attract-Repel specialisation model in order to apply the transformations learned from the "seen subspace" to *unseen* words (i.e. not appearing in the knowledge base). Kamath et al. (2019) and Glavaš and Vulić (2019) apply this idea of a general specialisation function to the lexical entailment (is-a) relation. Similarly to these fine-tuning approaches, the knowledge-augmented approach I propose extracts semantic relationships, specifically synonyms, from a lexical knowledge base and enforces said relationships in the learned word embeddings. Nevertheless, in my approach this enforcing occurs during the training phase, and not as a post-training stage.

Knowledge-Constrained Objective Functions This approach consists of incorporating an additional (*regularisation*) term to an existing training objective, where the added term enforces specific semantic relationships gathered from a knowledge base. The prototypical knowledge-constrained function takes the form:

$$\mathscr{L} = \mathscr{L}_{\text{distributional}} + \lambda \mathscr{L}_{\text{knowledge}}$$

where $\mathscr{L}$ is the total loss term, $\mathscr{L}_{\text{distributional}}$ is a distributional loss term, and $\mathscr{L}_{\text{knowledge}}$ is a loss term dependent on the incorporation of external knowledge. The trade-off between the distributional information and external knowledge is managed through a weighting coefficient λ that controls the relative importance of the loss terms. Yu and Dredze (2014) use the CBOW Word2Vec training objective (Mikolov et al., 2013a) as the *distributional loss* and define an additional *relation constrain* objective as their *knowledge loss*, which is defined in terms of the probability that two words are related with respect to a lexical resource. Bian et al. (2014) also use the CBOW objective and incorporate *auxiliary objectives* based on a word's syntactic and semantic information such that, given a context word, the model needs to learn to predict the focus word as well as its synonyms, POS tags, categories, hyponyms, etc. Fried and Duh (2015) propose a loss function that combines the *neural language model* by Collobert et al. (2011) with a relational distance term which is defined as the distance between words in a knowledge base, where the embeddings for the distributional and knowledge-constrained objectives can vary independently but are forced to remain minimally different by an additional Lagrangian penalty term. Liu et al. (2015) combine a Skip-gram objective with a penalty term based on a set of ranked semantic inequalities, which are cast in terms of a triplet of word embeddings $\mathbf{w}_i, \mathbf{w}_j, \mathbf{w}_k$ and a similarity function such that $\text{sim}(\mathbf{w}_i, \mathbf{w}_j) > \text{sim}(\mathbf{w}_i, \mathbf{w}_k)$ if the words i and j are more closely related (e.g. synonyms) than words i and k. Nguyen et al. (2017) learn hierarchical embeddings by combining the Skip-gram objective with two terms that minimise the distance between embeddings for pairs of words in a hypernym relationship. Jiang et al. (2018) use the Skip-gram objective for both the distributional and knowledge-constrained terms, where the negative samples in the knowledge-constrained objective are not randomly sampled from the text corpus but are rather selected in terms of their "word reading difficulty" score (from a pre-specified knowledge graph), where selecting pairs that are deemed to have significantly different scores enforces words of similar reading difficulties to have similar embeddings. Lauscher et al. (2019) seek to provide information on "true semantic similarity" to the BERT model by introducing an additional *lexical relation prediction* term to BERT's two language model objectives, which incorporates synonym and hyponym pairs as lexical constraints on the model. The addition of synthetically constructed synonyms in my knowledge-augmentation approach can be thought of as a knowledge-dependent term $\mathscr{L}_{\text{knowledge}}$, and the augmentation ratio plays a similar role to the knowledge weighting coefficient λ. The main difference between these models and my approach, however, is that my incorporation of linguistic knowledge is achieved by modifying the training data itself, while leaving the original Skip-gram objective unchanged. The intention behind my approach is to artificially surface these linguistic patterns in the data without having to explicitly modify the probabilistic modelling objective.

4.1.2 Related Work on Data Augmentation

The term "data augmentation" was initially used in the field of computer vision, where it remains a widely used technique (Ciresan et al., 2010; Krizhevsky et al., 2012; Wong et al., 2016). In general terms, it refers to the application of some label-preserving transformation to the data such as rotations (Rowley et al., 1998), image scaling (Wang et al., 2018b), or *elastic distortions* (Simard et al., 2003) (i.e. parameterised deformations of an image to emulate oscillations of hand muscles in handwritten digits). These transformations seek to increase the size and diversity of the training data while preserving its semantic content. The main goal of data augmentation is to improve the robustness of a model and help prevent overfitting (Simard et al., 2003). Due to the syntactic and semantic rules of composition that accompany language generation, the concept of a "label-preserving transformation" as a parametric function is not clearly defined for text data.

Given that text data augmentation strategies should abide by the intrinsic rules of language and accommodate the semantic information of its constituting elements (e.g. word meanings, n-grams, phrasal verbs, etc.), one of the most common augmentation strategies consists of performing word or phrase replacement operations. These replacements are typically guided by a lexical resource (e.g.

dictionaries, paraphrase datasets, lexical knowledge base, etc.). Synonym swapping is a commonly used replacement technique, since replacing a word with its synonym usually preserves the original meaning of the text. This *semantic invariance* is a fundamental property of data augmentation, as mentioned by Zhang et al. (2016). *Synonym-augmentation* has been exploited by Zhang et al. (2016) and Coulombe (2018) to produce more label-preserving examples for text classification tasks. The *contextual augmentation* proposed by Kobayashi (2018) extends synonym replacement to the more general idea of *paradigmatic relations* (e.g. `actors` and `performances` are not synonyms, but they can be substituted in the context of the phrase `the actors/performances were fantastic`[4]), where words are replaced by a predicted word from an RNN language model. The type of knowledge-augmentation technique I use for the experiments in this chapter is, in essence, a synonym replacement strategy, with the caveat that this replacement is performed not on raw text, but directly on Skip-gram word pairs. This aims to preserve the Skip-gram modelling process and minimise the noise introduced by this augmentation strategy.

Numerous other augmentation techniques for text data have been developed in recent years. These techniques are not as closely related to my knowledge-augmentation approach, but are included here for the sake of completeness. Coulombe (2018) proposes two methods of text data augmentation through paraphrasing based on regular expressions and syntax tree transformations. In the context of question answering, Wei Yu et al. (2018) augment the size of their question answering dataset through *back-translation*, a technique developed by Sennrich et al. (2016a) which produces paraphrases by translating an input text into a *pivot* language and subsequently translating it back to the original language. Dong et al. (2017) also apply back-translation paraphrasing to augment the data in a question answering task, together with two other paraphrasing methods: lexical and phrasal paraphrases extracted from the Paraphrase Database (PPDB) (Pavlick et al., 2015), and harvesting paraphrase rules (e.g. mapping `the average size of __`

[4]Example taken from Kobayashi (2018).

to `what be __ average size`) from a corpus of question clusters. Raiman and Miller (2017) propose a *TypeSwap* augmentation process which adds wrong question answering examples by replacing named entities with other entities of the same type, with which they aim to improve the model's ability to prune wrong answers. Xie et al. (2019) augment a text classification dataset with back-translation paraphrasing and a word replacing regime informed by a word's *informativeness* in terms of its term frequency-inverse document frequency score. A less common approach in text data augmentation is the addition of noise as described by Coulombe (2018), which amounts to modifying letters of words, changing the case of a letter, adding or removing punctuation, or adding frequently misspelled words.

4.1.3 Positioning

The knowledge augmentation approach I propose in this chapter combines the two families of approaches presented in this section, i.e. knowledge injection and data augmentation, and aims to exploit the main advantages of each. Through a process of linguistically-informed data augmentation, namely augmenting the text data used to train a word embedding model through a process of synonym replacement, my approach seeks to exploit the knowledge contained in linguistic knowledge bases in a data augmentation framework that mostly preserves word distributions. This approach constitutes an attempt to further narrow the gap between knowledge-based representations and distributional learning.

4.2 Background

The *distributional hypothesis* (Harris, 1954) states that the meaning of a word can be inferred from the **context**[5] in which it appears. Harris (1954) defines the *distributional regularities* of a word in terms of the words appearing in its context, where the "difference of meaning correlates with difference of distribution". This postulates that words that are semantically similar will occur in similar contexts, an idea more commonly associated with the *dictum of Firth*: "You shall know a word by the

[5]Harris (1954) talks about a word's *environment* or *neighbourhood*, i.e. the set of words appearing directly before and after that word. However, in line with more recent work, I will refer to these concepts as *context* for the remainder of this thesis.

company it keeps!" (Firth, 1962). Word co-occurrence models exploit this idea to learn vector representations of words by employing an *abstraction mechanism*[6] that maps word co-occurrence statistics into a vector space, such that words that have similar contexts are assigned vector representations that are close to each other.

The Word2Vec models proposed by Mikolov et al. (2013b) are two of the most widely used word co-occurrence language models in NLP. The experiments in this chapter explore the effects of my proposed *knowledge-augmentation* approach on the Skip-gram model, a variant of Word2Vec that learns word embeddings by optimising a probabilistic objective which, given a *focus* word f, attempts to predict its *context words* $c \in \mathscr{C}$, where context $\mathscr{C}$ is defined as a context window consisting of the C words appearing directly before and after the focus word. The Skip-gram language model aims to maximise an objective[7] that approximates the average probability $p(c \,|\, f)$ over all focus-context word pairs in a dataset $\mathscr{D}$, equivalent to maximising the following quantity:

$$J = \sum_{(f,c)\in\mathscr{D}} \log p(c \,|\, f) \tag{4.1}$$

where $\mathscr{D}$ refers to the dataset containing all preprocessed focus-context word pairs.

For a sample sentence `For many of the farmers involved, the news is devastating.`, the context window for a context size of $C = 3$ when `farmers` is the focus word looks as follows:

$$\texttt{For}\ \underbrace{\texttt{many of the}}_{\text{Left context}}\ \overbrace{\texttt{farmers}}^{\text{focus word}}\ \underbrace{\texttt{involved, the news}}_{\text{Right context}}\ \texttt{is devastating.}$$

where this example would contribute the following word pairs to the Skip-gram objective:

[6]Term taken from Boleda (2020).

[7]Mikolov et al. (2013b) describe the Skip-gram model's probabilistic objective as $\frac{1}{|\mathscr{F}|}\sum_{f\in\mathscr{F}}\sum_{c\in\mathscr{C}} \log p(c \,|\, f)$, a near average (the normalising term $\frac{1}{|\mathscr{F}|}$ should be $\frac{1}{|\mathscr{F}|\times|\mathscr{C}|}$ for this to be an average) log probability of context words given a focus word. Since maximising this term is proportional to maximising $p(c \,|\, f)$, the latter is used in this discussion for ease of exposition.

$$J = \ldots \log p(\texttt{many} \mid \texttt{farmers}) + \log p(\texttt{of} \mid \texttt{farmers}) + \log p(\texttt{the} \mid \texttt{farmers}) \ldots$$

Due to the properties of discrete distributions, generalising this language model to previously unseen data can become complex and require large numbers of parameters to provide accurate estimates, as described by Bengio et al. (2003). A possible alternative is to use neural networks as smoothing functions, which can simplify the estimation of these probabilities by distributing probability mass across neighbourhoods of similar words. **Neural (probabilistic) language models**, as first proposed by Bengio et al. (2003), are an approach to language modelling that employ neural networks to learn distributed representations of the words in a sequence. The probabilistic model is then cast in terms of these learned representations. Bengio et al. (2003) describe this for an n-gram language model:

$$\hat{p}(w_t \mid w_{t-1}, \ldots, w_{t-N}) = f(w_t, \mathbf{M}(w_{t-1}), \ldots, \mathbf{M}(w_{t-N})) \tag{4.2}$$

where $\mathbf{M} : \mathbb{R}^V \mapsto \mathbb{R}^M$ is an embedding function that maps the one-hot encoding of a word $w_i \in \mathscr{V}$ (of dimensionality $V = |\mathscr{V}|$) to a dense vector of dimensionality M, and f is a function that produces a probability distribution over the words in a vocabulary $\mathscr{V}$.

Skip-gram is a special case of a neural language model that sets its prediction task $p(c \mid f)$ as a **log-bilinear language model**, meaning that it defines the probability of a context word given a focus word in terms of two interacting linear units. Mikolov et al. (2013b) term these *input*, $\mathbf{u}$, and *output*, $\mathbf{v}$, embeddings, which can be thought of as single rows of the corresponding embedding matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{V \times M}$, such that the input and output embeddings for a word w are $\mathbf{u}_w = [\mathbf{U}]_w$ and $\mathbf{v}_w = [\mathbf{V}]_w$, respectively. The resulting probability distribution for a context word c given a focus word f is therefore:

$$p(c \mid f) = \frac{\exp(\mathbf{v}_c{}^\top \mathbf{u}_f)}{\sum_{v \in \mathscr{V}} \exp(\mathbf{v}_v{}^\top \mathbf{u}_f)} \tag{4.3}$$

where v is a word from vocabulary $\mathscr{V}$.

The intended interaction between these two embedding units is that, given two words, the dot product between the input embedding of the first and the output embedding of the second should be large if they frequently appear in the same context, and small if they do not usually co-occur. The normalisation term then ensures that this interaction has the characteristics of a probability mass function:

$$0 \leq p(c \mid f) \leq 1$$
$$\sum_{v \in \mathscr{V}} p(v \mid f) = 1$$

An issue with this setup is that, for every prediction made by the model, the **normalisation constant** or *partition function* $\sum_{v \in \mathscr{V}} \exp(\mathbf{v}_v{}^\top \mathbf{u}_f)$ requires a sum over all elements in the vocabulary. Since vocabularies can consist of tens or hundreds of thousands, or even millions of words, this operation can become prohibitively expensive to compute. To produce an efficient estimate of this normalisation constant, Mikolov et al. (2013b) implement a concept they call **negative sampling**, which is closely related to the idea of **noise-contrastive estimation (NCE)** proposed by Gutmann and Hyvärinen (2010).

The main idea behind NCE is that the normalisation constant can be approximated by learning to distinguish the data from some noise signal. Given an unnormalised probability density function $p^0_{\mathscr{D}}(\cdot;\alpha)$, the normalised version of the log of that distribution can be defined as $\log p_{\mathscr{D}}(\cdot;\alpha) = \log p^0_{\mathscr{D}}(\cdot;\alpha) + c$, where $c = -\log Z(\alpha)$ and $Z(\alpha)$ is the normalisation constant. The parameters for the normalised distribution $\Theta = \{\alpha, c\}$ are obtained by maximising the following objective function over the full set of observations in the dataset:

$$J(\Theta) = \frac{1}{2|\mathscr{D}|} \sum_{\mathbf{x} \in \mathscr{D}, \mathbf{s} \sim p_s(\cdot)} \log \sigma[G(\mathbf{x};\Theta)] + \log(1 - \sigma[G(\mathbf{s};\Theta)]) \tag{4.4}$$

where $\mathbf{x}$ is a sample from the dataset $\mathscr{D}$, $\mathbf{s}$ is a noise sample from the noise distribution $p_s(\cdot)$, $\sigma(x) = 1/(1+\exp(-x))$ is the sigmoid function, and $G(\mathbf{m};\Theta) =$

$\log p_{\mathscr{D}}(\mathbf{m};\Theta) - \log p_s(\mathbf{m};\Theta)$. To maximise this objective, $G(\mathbf{x};\Theta)$ should be large and $G(\mathbf{s};\Theta)$ should be small, which happens when there is a high probability that $\mathbf{x}$ comes from the data ($\frac{p_{\mathscr{D}}(\mathbf{x};\Theta)}{p_s(\mathbf{x};\Theta)} > 1$) and $\mathbf{s}$ comes from the noise distribution ($\frac{p_{\mathscr{D}}(\mathbf{s};\Theta)}{p_s(\mathbf{s};\Theta)} < 1$).

As described by Mikolov et al. (2013b), the aim of the Skip-gram model is not to closely model the softmax probabilities, but rather to learn high-quality vector representations, which justifies using a simplified version of the NCE as the normalisation constant estimator. This simplified estimator, which Mikolov et al. (2013b) call negative sampling, uses *K negative samples*[8] to contrast between focus-context word pairs that come from the dataset and *negative* word pairs which are artificially constructed. This idea comes from the approach by Mnih and Teh (2012) and Mnih and Kavukcuoglu (2013) where noise samples are K times more frequent than data samples. Negative word pairs are made up of the focus word provided as input and a word r that is randomly sampled from a noise distribution $p_s(\cdot)$. Mikolov et al. (2013b) set the noise distribution used for the negative samples to be the unigram distribution $p_{\text{uni}}(\cdot)$ (i.e. the frequency of a word in the dataset) raised to the $3/4$th power, $p_s(\cdot) = p_{\text{uni}}(\cdot)^{3/4}$. The resulting negative sampling objective for a single focus-context word pair (f,c) is:

$$J_{\text{NEG}}(f,c) = \log\sigma(\mathbf{v}_c{}^{\top}\mathbf{u}_f) + \sum_{i=1}^{K} \mathbb{E}_{r\sim p_s(\cdot)}\left[\log\sigma(-\mathbf{v}_r{}^{\top}\mathbf{u}_f)\right] \tag{4.5}$$

In line with the distributional hypothesis, this objective enforces the learning of word embeddings that place words with similar contexts in nearby regions of the learned embedding space. The distributional learning process of the Skip-gram model fits into the **structuralist** tradition of linguistics, which posits that a word's meaning is determined by its relations with other words within a *lexical field* or syntactic structure (Gasparri and Marconi, 2019). However, the number and linguistic diversity of said relations can be limited in training datasets. Since Skip-gram word embeddings are learned entirely on distributional information, their quality depends

[8]The choice of K is a hyperparameter, where a larger value reportedly results in a more accurate estimate at the expense of longer computational times (Gutmann and Hyvarinen, 2012).

on the contexts they observe for every particular word in the training data, which can make the learned embeddings **domain specific**, meaning that they only capture the regularities of a limited use of language.

In linguistics, the *relational* approach to lexical semantics offers a possible solution to the problem of **domain dependence**. **Relational semantics** is a sub-field of structuralist semantics[9] that describes word meaning in terms of a *structured hierarchical network* of semantic relations, otherwise known as a **lexical knowledge base**. The relations between words in these lexical knowledge bases are hard-coded from agreed upon *word senses*. The rigidity of these networks implies that they contain little to no information about a word's stylistic usage, yet they do provide a fixed set of relations that hold regardless of the linguistic domain from which a particular word is extracted. An instance of one such network is the **WordNet** knowledge base (George A. Miller, 1995), which is used for the experiments in this chapter. WordNet represents words as tuples consisting of a *form* (i.e. the string for the word), and a *sense* (i.e. a distinct meaning), where the connections between words are given by linguistic relations. For the relation of *polysemy*, a single form can take two or more senses, while in *synonymy* two or more forms share a single sense. Other relations in WordNet include *hyponymy* (and *hypernymy*) which are hierarchical broader-narrower (or supertype-subtype) relations, and *meronymy* (and *holonymy*) which are part-of relations, among others. It is important to note that WordNet, as well as other existing lexical knowledge bases, is created and maintained by humans and is therefore prone to contain errors or incomplete information, where very rare, specialised, or neologistic words might not be correctly documented or not appear at all.

In the context of training word embeddings, a combination of the distributional and relational approaches has the potential to produce embeddings that capture information about a word's usage as well as semantic relations for that word that transcend the specific linguistic style of the training corpus. This process can potentially produce more transferable word embeddings that are useful across linguistic

[9]According to the taxonomy provided by (Gasparri and Marconi, 2019).

domains and applications.

The integration of the distributional and relational approaches, however, implies a trade-off between preserving the distributional information and incorporating domain-invariant linguistic relations. The **knowledge-augmented distributional learning** approach I propose in this chapter aims to minimise this ensuing informational trade-off by incorporating information about semantic relations between words extracted from a knowledge base into the Skip-gram's distributional learning process. The main idea behind this approach is that, by following the assumption made by Harris (1954) that "difference of meaning correlates with difference of distribution", **words that share a meaning or *sense* should also have similar contextual distributions**. For instance, a training corpus sourced from a small selection of linguistic domains might not contain enough examples of `neglect` to robustly predict its context, i.e. $p(\cdot\,|\,\texttt{neglect})$, but the word `ignore` might appear frequently enough in this corpus to accurately model its context, $p(\cdot\,|\,\texttt{ignore})$. Since the words `neglect` and `ignore` have shared senses (i.e. are related by synonymy) in a lexical knowledge base like WordNet, the contextual information for `ignore` can be used to complement the contextual modelling of `neglect` under the assumption that the contextual distributions for both words should be similar. In this way, this corpus can be **augmented** with the relations sourced from a lexical knowledge base to explicitly enforce the semantic similarity of that pair of words. Augmentation from lexical knowledge bases can provide information for very rare words that might not be sufficiently represented even in large corpora. Additionally, augmentation will not boost the relevance of frequent misspellings, which could come as a byproduct of increasing the size of the training corpus.

A simple augmentation procedure for the sentence:

```
We ignore the majority of sounds present in the world.
```

would replace `ignore` with its synonym `neglect` and add the resulting sentence to the dataset:

```
We neglect the majority of sounds present in the world.
```

This, however, introduces a series of artifacts to the Skip-gram learning algorithm that might unnecessarily increase the noise in the augmented dataset and bias the data's co-occurrence statistics. One issue with this procedure is that it duplicates word pairs that are not related to the augmentation process, e.g. $p(\texttt{sounds} \mid \texttt{present})$ for focus word `present`. Another unwanted artifact is that, in the new sentence, the substituted word now appears in the context of other words that it did not appear with in the original dataset, e.g. when the focus word is `majority`, `neglect` becomes part of its context in the new sentence. This is an issue because the aim of the augmentation process is to bring the contextual distribution of the two synonyms, e.g. $p(\cdot \mid \texttt{neglect})$ and $p(\cdot \mid \texttt{ignore})$, closer together without needlessly modifying the context distribution of other words, e.g. $p(\cdot \mid \texttt{majority})$. To minimise the noisiness of the augmentation procedure, I only replace words after the Skip-gram word pairs have been constructed, and the substitution is only performed on focus words. For the example sentence `We ignore the majority of sounds present in the world.`, a context size of $C = 3$, and focus word `ignore`, the resulting word pairs would be:

Original pairs

(`ignore`, `We`)

(`ignore`, `the`)

(`ignore`, `majority`)

(`ignore`, `of`)

Augmented pairs

(`neglect`, `We`)

(`neglect`, `the`)

(`neglect`, `majority`)

(`neglect`, `of`)

Word pairs where the focus word is not augmented with a synonym do not generate any additional word pairs. This augmentation mechanism is designed to exploit the assumption that semantically similar words should have similar contextual distributions. Defining a contextual distribution as the probability that a context word c appears in the context of a word w, such that $p(c \mid w)$, this assumption that synonyms, like ignore and neglect, should have similar contextual distributions can be described as $p(w \mid \texttt{neglect}) \approx p(w \mid \texttt{ignore})$ for any w. The augmentation process I propose here seeks to produce an augmented contextual distribution, e.g. $\tilde{p}(\cdot \mid \texttt{neglect})$, that is closer to the contextual distribution of a synonymous word, which can be expressed in terms of Kullback-Leibler divergences, $KL[\tilde{p}(\cdot \mid \texttt{neglect}) || p(\cdot \mid \texttt{ignore})] < KL[p(\cdot \mid \texttt{neglect}) || p(\cdot \mid \texttt{neglect})]$.

With respect to the Skip-gram model, the decision to only replace focus words forces the input embeddings of the synonym $\mathbf{u}_{\texttt{neglect}}$ and source words $\mathbf{u}_{\texttt{ignore}}$ to come closer together in embedding space, while it avoids the noisy interactions that would come with modifying the output embedding of the synonym $\mathbf{v}_{\texttt{neglect}}$. For instance, following equation 4.3, the objective for the augmented word pair made up of neglect as the focus and majority as the context word would be as follows:

$$p(\texttt{majority} \mid \texttt{neglect}) = \frac{\exp(\mathbf{v}_{\texttt{majority}}^{\top} \mathbf{u}_{\texttt{neglect}})}{\sum_{v \in \mathscr{V}} \exp(\mathbf{v}_v^{\top} \mathbf{u}_{\texttt{neglect}})} \tag{4.6}$$

Even though this augmentation regime attempts to preserve the distributional information that is captured during the learning process, augmenting the data introduces a certain level of noise by adding examples that do not naturally occur. The synonym replacement scheme is also inherently noisy, since for a word with multiple synonyms there is no clear way to automatically decide which synonym is most appropriate. For example, given the sentence `We ignore the majority of sounds present in the world.`, possible synonym replacements for the word world, include `...sounds present in the globe` and `...sounds present in the existence`, where the second example would very rarely occur in the real world. One more consideration is that, while the context distributions for synonyms should in theory be similar, making them too similar can wash away semantic and stylistic information about the words being represented. For instance, while world and globe can share senses, they can appear in different contexts and carry different connotations. The experiments in this chapter explore different ratios of augmented to *natural* (i.e.

coming from the original dataset) to limit the amount of noise that the augmentation regime contributes to the distributional learning process.

As an additional measure to preserve distributional information when training word embeddings on an augmented dataset, I propose an **embedding space partitioning** technique, which ensures that the augmentation regime only affects a *subspace* of the full embedding space. This partitioning technique consists of constraining the error propagation of augmented examples to only a subset of all the available dimensions of the *input embedding* space. In practice, this is achieved by replacing the gradient of the Skip-gram negative sampling objective with respect to the input embedding **for a synonym**, $\nabla_{\mathbf{u}_{\text{syn}}} J_{\text{NEG}}$, with a modified gradient $\tilde{\nabla}_{\mathbf{u}_{\text{syn}}} J_{\text{NEG}}$ where the gradients calculated during back-propagation of the *unaugmented* dimensions are set to zero, while the gradients of unaugmented word pairs remain unchanged, $\nabla_{\mathbf{u}_{\text{word}}} J_{\text{NEG}}$. The modified gradient is shown in the following equation:

$$\tilde{\nabla}_{\mathbf{u}_{\text{syn}}} J_{\text{NEG}} = \begin{bmatrix} \frac{\partial J_{\text{NEG}}}{\partial [\mathbf{u}_{\text{syn}}]_1} \\ \frac{\partial J_{\text{NEG}}}{\partial [\mathbf{u}_{\text{syn}}]_2} \\ \vdots \\ \frac{\partial J_{\text{NEG}}}{\partial [\mathbf{u}_{\text{syn}}]_B} \\ \frac{\partial J_{\text{NEG}}}{\partial [\mathbf{u}_{\text{syn}}]_{B+1}} = 0 \\ \frac{\partial J_{\text{NEG}}}{\partial [\mathbf{u}_{\text{syn}}]_{B+2}} = 0 \\ \vdots \\ \frac{\partial J_{\text{NEG}}}{\partial [\mathbf{u}_{\text{syn}}]_M} = 0 \end{bmatrix}$$

where $[\mathbf{u}_{\text{syn}}]_i$ is the ith dimension of the input embedding, B is the dimensionality of the *augmented subspace*, and the augmented dimensions are coloured in green.

Embedding space partitioning is explored as an addition to the knowledge-augmented distributional learning of word embeddings. This approach seeks to retain more distributional information in the model by localising the effect of the data augmentation, such that the dimensions of the learned embedding that are unaffected by the augmentation will only contain information about the original dataset.

The experiments in this chapter analyse the effects of this informational constraint by varying the dimensionality of the augmented subspace, i.e. the value of B.

4.3 Learning Algorithm

This section delineates the general experimental framework used to train and evaluate the knowledge-augmented word embedding models proposed in this chapter.

4.3.1 Task, dataset, and architecture definition

While the overarching goal of these experiments is to learn distributed word embeddings, the main **learning task** is based on a Skip-gram language model (Mikolov et al., 2013a), where the aim is to predict a **context word** $c \in \mathscr{C}$ for a given **focus word** f with context window $\mathscr{C}$:

$$p(c \mid f) \tag{4.7}$$

The **dataset** used for this task is the BNC dataset described in section 3.4.1. Due to computational constraints, the experiments in this chapter are carried out on a subset of 10% of the sentences randomly sampled from the full dataset (the subset data is identical in all experiments). The construction of this subset is also detailed in section 3.4.1. The **architecture** for these experiments is the Skip-gram model with negative sampling as described by Mikolov et al. (2013b).

4.3.2 Preprocessing

The Skip-gram training regime works with word pairs, which requires the raw dataset to be fully tokenised up to the word-level. Aside from basic tokenisation, I perform a series of additional steps to shuffle the data, reduce variability among the tokens (e.g. removing uppercases), and remove unnecessary tokens (e.g. punctuation marks). The resulting preprocessing pipeline consists of the following steps:

1. **Concatenate all BNC documents** into a single continuous raw text corpus
2. **Tokenise sentences** following BNC XML tags
3. **Remove XML information** since BNC tags are not used in these experiments

4. **Shuffle sentences** (since the BNC data is organised into topics, shuffling ensures that this training corpus has interspersed sentences from all topics) and **retain a subset** of 10% of the sentences in the data (601,818 sentences)

5. **Tokenisation and POS tagging** of words for every sentence in the dataset using the spaCy NLP library (Honnibal and Montani, 2017), where POS tags will be used to detect punctuation marks and will also inform the data augmentation process (i.e. only certain types of words, like adjectives or verbs, will be replaced)

6. **Convert to lower case** to minimise token variations, which helps reduce the size of the vocabulary by, for instance, mapping `Her` and `her` to the same token

7. **Remove punctuation** after tokenising and tagging in order to avoid removing punctuation symbols from acronyms or numbers

8. **Convert numeric tokens** to a generic format, e.g. `12.45` is converted to `##.##`, which is done to preserve the structure of numeric tokens (e.g. differentiate between amounts and dates) while reducing their variability, since storing every numeric token as a distinct token can cause the vocabulary to grow substantially

9. **Prune sentences** containing a single word since they have no contextual information (37,201 sentences removed)

Listing 4.1 shows a sample sentence after preprocessing.

Listing 4.1: Sentence `In fact, we ignore the majority of sounds present in the world.` after preprocessing

```
[
      ["in", "ADP"],
      ["fact", "NOUN"],
      ["we", "PRON"],
      ["ignore", "VERB"],
        ["the", "DET"],
        ["majority", "NOUN"],
        ["of", "ADP"],
        ["sounds", "VERB"],
        ["present", "ADJ"],
        ["in", "ADP"],
        ["the", "DET"],
        ["world", "NOUN"]
]
```

4.3.3 Vocabulary construction

The vocabulary used in these experiments is based on the 10% subset of the full BNC dataset (hereon referred to simply as the *dataset*) and follows the pruning procedure described in section 3.4.2 to remove very infrequent words for which there might not be enough information or which might include misspellings or overly specific terminology. The previously mentioned pruning uses a **frequency threshold of 5**, meaning that any word that appears fewer than 5 times in the dataset gets pruned from the vocabulary. This choice of frequency threshold is motivated by its reduction of vocabulary size, where the resulting vocabulary is left with **45,769 unique tokens**, which amounts to 32.45% of the 141,044 unique tokens in the full dataset. This reduction in vocabulary is achieved while maintaining an acceptable coverage over the training data, since the pruned vocabulary covers 98.48% of the 9,879,226 non-distinct tokens in the full dataset. The vocabulary, hereafter denoted by $\mathscr{V}$, is stored as a table containing the word together with its index, counts, and normalised frequency.

4.3.4 Datapoint construction

Given a dataset tokenised into sentences, and sentences tokenised into words, where words are tagged with their POS tags, the next step is to construct Skip-gram word pairs. These word pairs are constructed by setting a focus word f and fixing a context window of size C around it (i.e. the C words appearing directly before and after the focus word). Any word falling within that window is a context word c, and word pairs are made up of every combination of focus-context words found. So for the sentence `In fact, we ignore the majority of sounds present in the world.`, and context size $C = 3$, the context windows for the (focus) word ignore look as follows:

$$\underbrace{\texttt{In fact, we}}_{\text{Left context}}\ \overbrace{\texttt{ignore}}^{\text{focus word}}\ \underbrace{\texttt{the majority of}}_{\text{Right context}}\ \texttt{sounds present...}$$

which produces the following word pairs:

(`ignore`, `In`)

(`ignore`, `fact`)

(`ignore`, `we`)

(`ignore`, `the`)

(`ignore`, `majority`)

(`ignore`, `of`)

These word pairs are added to the Skip-gram dataset. Every datapoint contains the focus and context words, with their respective POS tags, as well as their source information: sentence number from the original dataset, position of the focus word in that sentence, and position of the context word relative to the focus word. The datapoints for the focus word `ignore` described above are shown in listing 4.2.

Listing 4.2: Skip-gram dataset

```
[["focus_word", "context_word", "sent_num", "
   focus_index", "ctx_position"],
[["ignore", "VERB"],["in", "ADP"],  5,   3,     -3],
[["ignore", "VERB"],["fact", "NOUN"],  5,   3,     -2]
   ,
[["ignore", "VERB"],["we", "PRON"],  5,   3,     -1],
[["ignore", "VERB"],["the", "DET"],  5,   3,      1],
[["ignore", "VERB"],["majority", "NOUN"],5,    3,      2
   ],
[["ignore", "VERB"],["of", "ADP"],  5,    3,      3],
```

This process then continues by going through every word in the text, treating every word as the focus word and sliding the context windows accordingly. In these experiments, context windows are constrained to the sentence in which the focus word appears, meaning that there are no inter-sentence context windows:

$$\texttt{...ignore the majority of } \underbrace{\texttt{sounds present in}}_{\text{Left context}} \ \overbrace{\texttt{the}}^{\text{focus word}} \ \underbrace{\texttt{world.}}_{\text{Right context}}$$

Since context windows are constrained to be intra-sentential, sentences that are made up of a single token do not provide any context and are therefore skipped. For the experiments in this chapter, the context size is kept at a constant value of $C = 5$, meaning the maximum context distance for a word pair in the dataset is either 5 positions before or 5 positions after the focus word. This context size matches the one reported by Mikolov et al. (2013b) in one of the original Skip-gram papers. Even though the Skip-gram paper by Mikolov et al. (2013a) reports a larger context size of $C = 10$, for these experiments I opted for the smaller context size since it favours the smaller context windows that naturally occur when context is constrained to a single sentence.

Mikolov et al. (2013a) argue that context words that appear closer to a focus word are usually more closely related to the focus word. To reduce the impact of context words that appear farther away, they propose setting C as the maximum context size and randomly sampling a context size ψ for every focus word f, i.e. $\psi_f \sim U(1,C)$, where $U(a,b)$ is the discrete uniform distribution over values in the closed interval $[a,b]$. For these experiments, I implement a context sampling procedure that achieves a similar result[10] by randomly sampling a context

[10]In both context size sampling techniques, the probability of including a context word c that is x positions away from a focus word is defined by $p(x) = (1 + \frac{1}{C}) - CDF[U(1,x)]$, where C is the maximum context window size, and $CDF[U(\cdot)]$ is the cumulative distribution function of the

size for every word pair, $\psi_{(f,c)} \sim U(1,C)$. If the (absolute) context position of the context word with respect to the focus word is larger than the sampled context size, the word pair is dropped. From the sample datapoints in listing 4.2, a sampled context size $\psi_{(\texttt{ignore},\texttt{in})} = 2$ would result in dropping the first datapoint, while $\psi_{(\texttt{ignore},\texttt{fact})} = 2$ would preserve the second datapoint. Listing 4.3 shows a context-sampled version of the datapoints from listing 4.2. The justification for using this context sampling regime, as opposed to that presented in the original paper by Mikolov et al. (2013a), is that this sampling method can more naturally accommodate the synonym replacement method that I propose here.

Listing 4.3: Sampled Skip-gram dataset

```
[["focus_word",  "context_word",   "sent_num",  "
   focus_index",  "ctx_position"],
[["ignore", "VERB"],["we", "PRON"],  5,   3,     -1],
[["ignore", "VERB"],["the", "DET"],  5,   3,      1],
[["ignore", "VERB"],["majority", "NOUN"],5,   3,      2
   ],
[["ignore", "VERB"],["of", "ADP"],  5,   3,      3],
```

The **synonym augmentation regime** occurs after the full Skip-gram dataset has been constructed and the contexts have been sampled. The first step in this process consists of selecting viable *candidates* for replacement, which amounts to choosing focus words that are tagged as nouns (`"NOUN"`), adjectives (`"ADJ"`), adverbs (`"ADV"`), or verbs (`"VERB"`). Every candidate word generates a query to WordNet to obtain its *synset*, or set of available synonyms, from which a single synonym is selected. There are different possible synonym selection strategies, such as weighting by the synonym's frequency. When applied to the data, the frequency-weighted synonym selection strategy was seen to select the most frequent synonym in the majority of cases, which is most likely due to the nature of word frequency distributions in text, as described by Zipf's law. Even though this strategy reinforces the naturally occurring frequencies of words, the synonym augmentation regime is

uniform distribution.

intended to artificially enforce semantic relations that are not frequently observed in the data. For this reason, in these experiments I opted for a random selection strategy where all synonyms in the synset are equally likely to be selected. Listing 4.4 shows the synonym replacement process on the context-sampled datapoints from listing 4.3.

Listing 4.4: Synonym replacement in Skip-gram dataset

```
[["synonym", "context_word", "sent_num",  "focus_index
   ", "ctx_position", "focus_word"],
["dismiss",  ["we", "PRON"],  5,   3,     -1,    ["
   ignore", "VERB"]],
["neglect",  ["the", "DET"],  5,   3,      1,     ["
   ignore", "VERB"]],
["snub",    ["majority", "NOUN"],5,   3,      2,     ["
   ignore", "VERB"]],
["dismiss",  ["of", "ADP"],  5,   3,      3,     ["
   ignore", "VERB"]],
```

The resulting *context-sampled* and *synonym-replaced* datasets are stored in separate files.

4.3.5 Input representation

In preparation for processing, datapoints are first simplified to only contain focus-context word pairs, with all other information (e.g. tags, context position, etc.) being discarded. An example of this is shown in listing 4.5.

Listing 4.5: Skip-gram word pair dataset, the first column corresponds to the focus word, and the second to the context word.

```
[["ignore", "we"],
["ignore", "the"],
["ignore", "majority"],
["ignore", "of"],
```

These word pairs are then numericalised according to the word indices defined in the vocabulary file, with out-of-vocabulary words being replaced with the index of the generic *unknown* token `<unk>`. An example of this numericalisation applied to the dataset in listing 4.5 is provided in listing 4.6.

Listing 4.6: Skip-gram numericalised dataset, the first column corresponds to the focus word, and the second to the context word.

```
[[4171, 37],
[4171, 4],
[4171, 1040],
[4171, 5],
```

This process is carried out for both the *context-sampled* and the *synonym-replaced* datasets. The resulting context-sampled and numericalised datasets are called *training*, *training-synonyms*, and *validation*.

4.3.6 Dataset split

The tokenised sentences in the preprocessed dataset are shuffled before the dataset is split into a **training set** made up of **90%** of the sentences in the full dataset, which amounts to 508,326 sentences, and a **validation set** containing the remaining 56,291 sentences that make up the other **10%** of the dataset. This choice of training-validation split follows a widely used heuristic and also allows for an efficient use of training data when working with moderately sized datasets.

4.3.7 Model training

4.3.7.1 Pre-epoch processing

Before training begins, a model's parameters can be initialised to pre-specified values. In these experiments, I explore random initialisation for the two input $\mathbf{U}$ and output $\mathbf{V}$ embedding matrices, as well as initialising the input embeddings to the publicly available Skip-gram embeddings which had been pre-trained on the Google News dataset (one billion words). In this process, which I refer to as Word2Vec initialisation, any words from the vocabulary that do not appear in the pre-trained Skip-gram embeddings are initialised to the zero vector $\mathbf{0}$. Since I was unable to find pre-trained output Skip-gram embeddings, this Word2Vec initialisation is only applied to the input embeddings,

All three datasets, *training*, *training-synonyms*, and *validation*, are **shuffled** at the beginning of every epoch. Training is carried out for exactly **10 epochs**, so early stopping checks are omitted. The decision to train for a fixed number of epochs is part of an attempt to keep experimental conditions fixed across models, and all models displayed certain degree of convergence after 10 epochs. To speed up computation, data is processed in **batches of 20 datapoints**, after initial tests showed that a batch size of 20 provided a significant speedup in training while achieving good validation performance. A batch is constructed by popping 20 word pairs from the (shuffled) *training* and *training-synonyms* datasets, where the probability that a word pair in the batch came from the *training-synonyms* dataset is determined by the **augmentation ratio**. Three different augmentation ratios are tested in these experiments: 12%, 25%, 37%, where, for example, an augmentation ratio of 25% means that, in expectation, 5 out of the 20 word pairs in a batch will be synonym replacements. Finally, for every word pair, $K = 5$ **negative samples**[11] are constructed by sampling from the unigram distribution p_{uni} raised to the $3/4$th power, as described in Mikolov et al. (2013b).

4.3.7.2 Data processing

Every batch provided as input to the model is made up of 20 index pairs representing the focus and context words, and 5×20 indices for the negative samples. Every focus word index f gets replaced with an **input embedding** $\mathbf{u}_f \in \mathbb{R}^M$ corresponding to the fth row of the input embedding matrix $\mathbf{U} \in \mathbb{R}^{V \times M}$, where M is the dimensionality of the word embeddings and V is the size of the vocabulary $\mathscr{V}$. Similarly, every context word index c and negative sample index r are converted to **output embeddings** $\mathbf{v}_c \in \mathbb{R}^M$ and $\mathbf{v}_r \in \mathbb{R}^M$, respectively, which correspond to rows of the output embedding matrix $\mathbf{V} \in \mathbb{R}^{V \times M}$. The full batch is processed with matrix and tensor products, which are more computationally efficient than performing vector dot products separately. An example of the Skip-gram architecture processing a

[11]Mikolov et al. (2013b) report values of K (number of negative samples) between 5 and 20, or between 2 and 5 for larger datasets, are effective in practice.

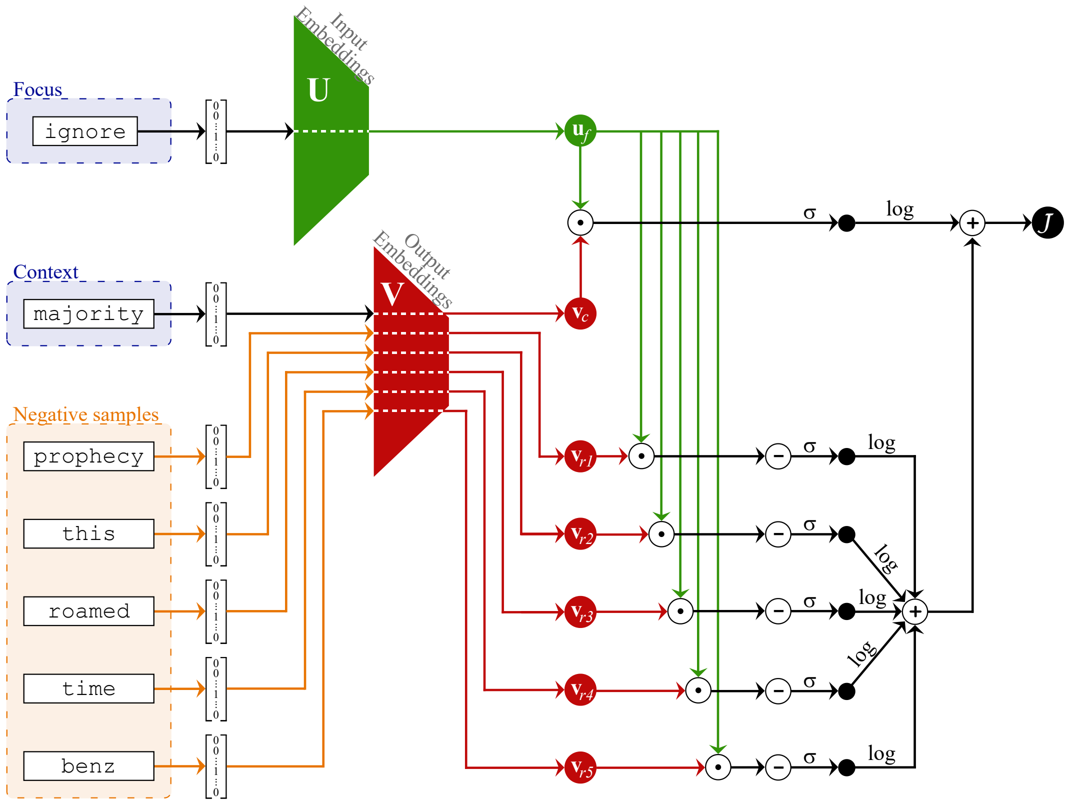


Figure 4.1: Skip-gram architecture processing a word pair made up of `ignore` (focus word) and `majority` (context word), and 5 negative samples

word pair with negative sampling is shown in figure 4.1.

4.3.7.3 Error calculation

The Skip-gram language model is set up as a function of the dot product $\mathbf{v}_c{}^\top \mathbf{u}_f$ between the output embedding of the context word and the input embedding of the focus word, and a noise contrastive term $-\mathbf{v}_r{}^\top \mathbf{u}_f$ defined as the negative dot product between the output embedding of the negative samples and the input embedding of the focus word. The full Skip-gram negative sampling objective is as described in section 4.2:

$$J_{\text{NEG}}(f,c) = \log \sigma(\mathbf{v}_c{}^\top \mathbf{u}_f) + \sum_{i=1}^{K} \mathbb{E}_{r \sim p_s(\cdot)} \left[\log \sigma(-\mathbf{v}_r{}^\top \mathbf{u}_f) \right]$$

4.3.7.4 Error propagation and optimisation

The error is backpropagated through the input and output embedding matrices, where the only contribution to the error for a datapoint is localised in the row of the input embedding matrix that corresponds to the focus word $\mathbf{u}_f = [\mathbf{U}]_f$, and the rows of the output embedding matrix that correspond to the context word $\mathbf{v}_c = [\mathbf{V}]_c$ and the negative samples $\mathbf{v}_r = [\mathbf{V}]_r$. Given an embedding dimension of $M = 300$ and a vocabulary size of $V = 45,769$, the full set of trainable parameters Θ in the model consists of the input and output embedding matrices:

$$\Theta = \{\mathbf{U}, \mathbf{V} \in \mathbb{R}^{45,769 \times 300}\}$$

After backpropagating the error for the full batch, the embedding matrices are updated using a SGD optimiser with a learning rate of $\eta = 0.1$. To further explore the effects of knowledge augmentation, I train a set of models using the *embedding space partitioning* technique described earlier, where the augmented data only affects a subspace of dimension B of the embedding space. More concretely, this amounts to restricting the error propagation to a specific subset of the dimensions in the input embedding matrix $\mathbf{U}$ when processing a synonym-replaced word pair. When processing an augmented example, the restricted error propagation is implemented by setting the gradients of the unaugmented dimensions to zero, which is meant to ignore the augmentation. Models are trained on three different partitioned proportions of augmented-unaugmented subspaces: 25%-75% (i.e. 75-225 dimensions), 50%-50% (i.e. 150 dimensions), 75%-25% (i.e. 225-75 dimensions).

4.3.7.5 Validation phase

The validation phase consists of repeating the data processing step on the validation set to produce a single **validation score**. This helps keep track of the generalisation capabilities of the model after training for an epoch, which can help avoid any overfitting, as well as confirm that the model is converging to a local optimum of the objective function.

4.3.8 Post-training

After model training is concluded, the resulting word embeddings are evaluated on a set of metrics to probe for semantic and syntactic information in the learned embedding space. Only the trained input embeddings are evaluated. For these experiments, all models are evaluated on the evaluation metrics described in section 3.4.4, namely the **WMD kNN document classification** task for the extrinsic evaluation, and the **similarity-distance correlation** scores and **word pair distance distributions** as intrinsic evaluation metrics. All evaluation conditions from section 3.4.4 remain constant for these experiments. The pre-trained Skip-gram model cropped to the BNC vocabulary, as described in section 3.4.3, is used as a baseline model to get a better sense of how the embeddings in these experiments measure up to a well-established model. Despite the evidence that retrofitting and semantic specialisation methods improve word embedding performance on downstream tasks (Faruqui et al., 2015; Mrkšić et al., 2017; Vulić and Mrkšic, 2018; Vulić et al., 2018; Kamath et al., 2019; Glavaš and Vulić, 2019), they are not used in the empirical comparisons in this chapter. These methods were not evaluated since the focus of this chapter is on the effects of the knowledge-augmentation approach with respect to the original model they augment. A thorough comparison between knowledge-augmented embeddings and semantic specialisation methods is left as future work.

4.4 Results

The synonym-augmented Skip-gram models presented in this chapter are trained with different combinations of the following training conditions:

- *Initialisation method*, where **Word2Vec init** is used to denote models where the input embeddings are initialised to pre-trained Skip-gram embeddings, and **Rand init** indicates both input and output embeddings are initialised to random values.

- *Augmentation ratio*, which refers to the proportion of synonym augmented examples in the training data and can take a value of **12% syns**, **25% syns**, **37% syns**, or **no-syns** for unaugmented models.

- *Augmented partition size*, or the percentage of embedding dimensions that are updated when processing synonym-augmented examples, where the possible values are **25% part** (75/300 augmented dimensions), **50% part** (150/300 augmented dimensions), **75% part** (225/300 augmented dimensions. When no partition size is indicated, augmented examples affect all embedding dimensions.

The models trained with these variations are referred to as *trained models* for the remainder of this chapter. The original pre-trained Skip-gram word embeddings (described in section 3.4.3), referred to here simply as **Word2Vec**, is used as a baseline for these experiments. The models in this chapter trained with random initialisation (Rand init) and no augmentation (no-syns) correspond to training embeddings following the original Skip-gram formulation. However, it is important to note that, due to differences in hyperparameters and, more importantly, differences between the dataset and vocabulary used for training, these models are not directly comparable to the original Skip-gram embeddings.

WMD Document Classification accuracies with corresponding error bounds are presented in table 4.1 for models with different initialisation methods and augmentation ratios (scores for partitioned models are deferred to table 4.4). The pre-trained **Word2Vec** model achieves the highest accuracy by a considerable margin. The classification accuracies of all the models I trained are within statistical bounds of each other, regardless of augmentation ratio and initialisation method. What is meant by *within statistical bounds* in this context is whether the models have overlapping confidence intervals (measured at a 95% confidence), which is indicative of whether there is a statistical difference between the two measures. While this method is not a formal measure of statistical significance (Schenker and Gentleman, 2001), it is used here as a simpler proxy to evaluate the significance of experimental results.

Similarity-Distance Correlation scores (described in detail in section 3.4.4) are shown in tables 4.2 and 4.3, which contain the correlation scores for cosine and Euclidean distances, respectively. The shading in each cell is proportional to the degree of correlation, i.e. a lighter shade indicates less correlation (closer to zero),

Table 4.1: WMD kNN document classification accuracies on the 20 Newsgroups dataset for unaugmented and augmented models with different initialisation methods

Model	Accuracy
Word2Vec	**62.75% (± 0.89)**
Word2Vec init no-syns	55.01% (± 1.12)
Word2Vec init 12% syns	55.60% (± 1.12)
Word2Vec init 25% syns	54.75% (± 1.12)
Word2Vec init 37% syns	54.95% (± 1.12)
Rand init no-syns	54.51% (± 1.12)
Rand init 12% syns	54.12% (± 1.13)
Rand init 25% syns	54.45% (± 1.12)
Rand init 37% syns	54.82% (± 1.12)

Table 4.2: Similarity-distance correlation results with **cosine** distance for unaugmented and augmented models with different initialisation methods

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.5807	-0.4200	-0.3241	-0.2490
Word2Vec init no-syns	-0.1453	-0.0959	-0.0966	-0.0693
Word2Vec init 12% syns	-0.1229	-0.1790	-0.0953	-0.0471
Word2Vec init 25% syns	-0.0900	-0.0402	-0.1044	-0.0456
Word2Vec init 37% syns	-0.1242	0.0075	-0.0582	-0.0430
Rand init no-syns	-0.0486	-0.0315	-0.0320	-0.0393
Rand init 12% syns	-0.0986	-0.0409	-0.0961	-0.0470
Rand init 25% syns	-0.1149	-0.0580	-0.0571	-0.0745
Rand init 37% syns	-0.1271	-0.0464	-0.0494	-0.0387

while a darker shade indicates a stronger correlation (larger positive or negative value). The correlation scores for cosine distances are significantly lower across all models than the corresponding scores for Euclidean distances. While the correlation scores for these two distance metrics exhibit similar patterns, these patterns are more evident for Euclidean distances. For this reason, and to use results that more closely align with the extrinsic evaluation (the WMD is based on Euclidean distances), the intrinsic evaluation analysis focuses on Euclidean distance. The rest of the results for cosine distances can be found in appendix A.

The similarity-distance correlation scores in table 4.3 show that the pre-trained

Table 4.3: Similarity-distance correlation results with **Euclidean** distance for unaugmented and augmented models with different initialisation methods

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.7706	-0.6150	-0.4426	-0.3565
Word2Vec init no-syns	-0.5144	-0.3207	-0.1957	-0.0952
Word2Vec init 12% syns	-0.4577	-0.4151	-0.1953	-0.0821
Word2Vec init 25% syns	-0.4365	-0.2589	-0.1951	-0.0467
Word2Vec init 37% syns	-0.4225	-0.2008	-0.1595	-0.0854
Rand init no-syns	-0.3694	-0.2347	-0.1206	-0.0817
Rand init 12% syns	-0.4459	-0.2570	-0.1616	-0.0631
Rand init 25% syns	-0.4956	-0.2472	-0.1601	-0.0984
Rand init 37% syns	-0.5211	-0.2405	-0.1387	-0.0834

Word2Vec model outperforms the trained models. The effect of the augmentation process is not clear for Word2Vec initialised models, where increasing the augmentation ratio from 0% (*Word2Vec init no-syns*) to 37% (*Word2Vec init 37% syns*) seems to consistently weaken correlation, with the exception of the *WordSim353 Rel* dataset, where the highest correlation score is obtained by the *Word2Vec init 12% syns* model. In randomly initialised models (*Rand init*), knowledge-augmentation does seem to improve correlation across all correlation datasets up to an augmentation ratio of between 12% and 25%, after which point scores seem to start declining, with the exception of the *WordSim353 Sim* dataset, where correlation seems to consistently improve as the augmentation ratio increases.

Word Pair Distance Distributions (Euclidean) are displayed in figure 4.2 for the unaugmented models, namely *Word2Vec*, *Word2Vec init no-syns*, and *Rand init no-syns*. It is worth noting that the synsets used for this evaluation metric come from a different subset of the BNC dataset than the one used to train and validate the models presented in this chapter. Even though there is no overlap in the data used in this evaluation and the training data, the similarity of the linguistic domains must be pointed out. The most noteworthy aspect of this plot is the difference in the shape of the distance ditributions across all groups of word pairs, where the word-pair distance distributions for *Word2Vec* and *Word2Vec init no-syns* appear more normally distributed, in stark contrast with the *Rand init no-syns* model's asymmetric and long-tailed distance distributions. Despite displaying similar distribution shapes, the mean distances for all word pair groups in the pre-trained *Word2Vec* are significantly smaller than those of the *Word2Vec init no-syns*, indicating that the training regime followed by the *Word2Vec init no-syns* model causes the original *Word2Vec* embeddings to move away from each other. Regarding the relative central tendency between word pair groups (the white dots in the violin plots represent the mean of the distribution), the three models in figure 4.2 display a larger mean distance for the random word pairs when compared to contextual and synonym pairs, which helps validate that embeddings for related words are indeed closer than those of unrelated words. Another desirable property for these embeddings is for synonyms to be closer together than contextual word pairs, since synonyms are meant to be more semantically similar. This property is reflected in the pre-trained *Word2Vec*, but is not as clear in the *Word2Vec init no-syns* and *Rand init no-syns* models.

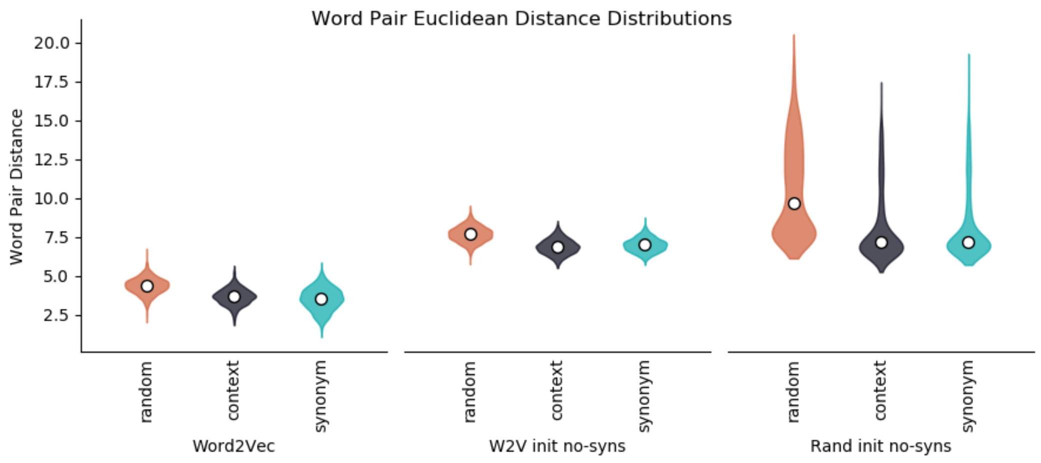


Figure 4.2: Euclidean distance distributions between random, contextual, and synonymous word pair sets for unaugmented models: *Word2Vec*, *Word2Vec init no-syns*, and *Rand init no-syns*

Figures 4.3 and 4.4 show the word pair distance distributions for Word2Vec initialised and randomly initialised models, respectively. As with the unaugmented models in figure 4.2, the most salient difference between these two plots is the shape of the distance distributions, where the word pair distances in the Word2Vec initialised models appear to be more normally distributed. Another important aspect that is consistent in these two plots is that both the Word2Vec initialised and randomly initialised models' distances seem to remain largely unaffected by the degree of augmentation.

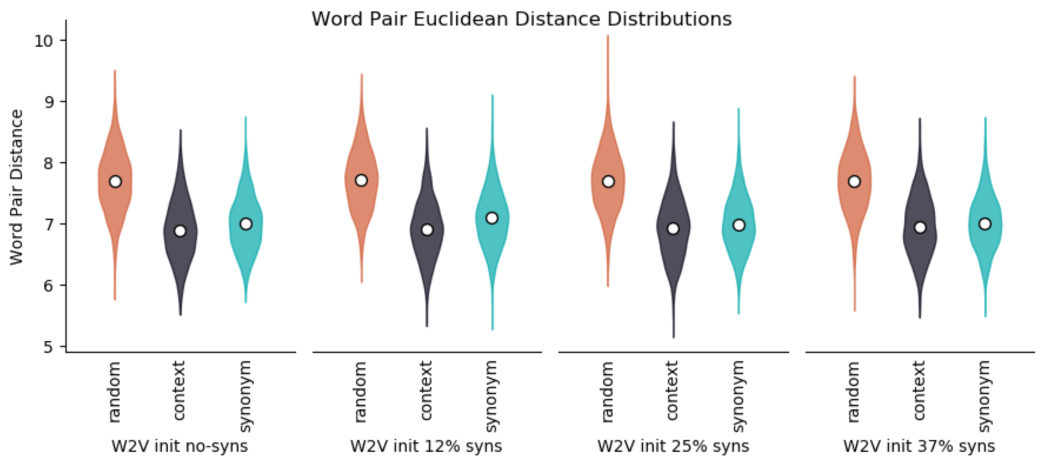


Figure 4.3: Euclidean distance distributions between random, contextual, and synonymous word pair sets for Word2Vec initialised models with different augmentation ratios

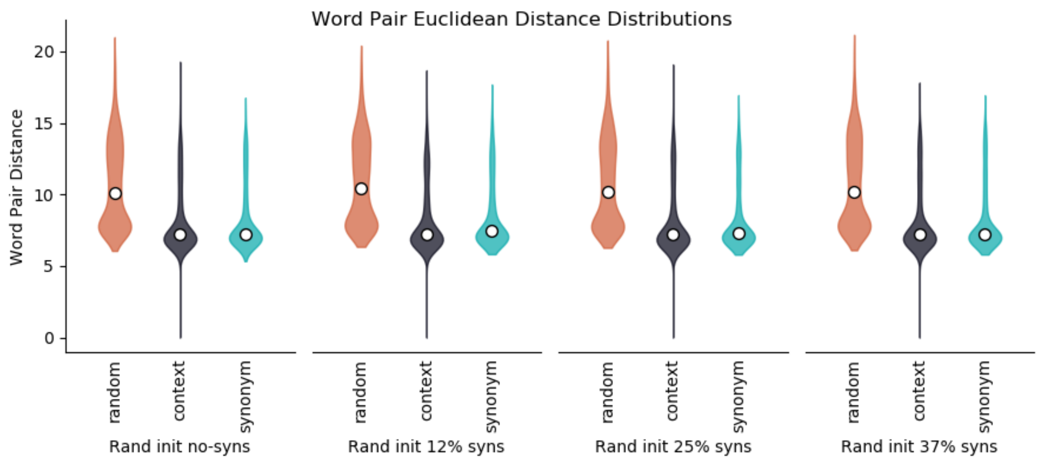


Figure 4.4: Euclidean distance distributions between random, contextual, and synonymous word pair sets for randomly initialised models with different augmentation ratios

Partitioned Knowledge-Augmented Models

WMD Document Classification results for the partitioned knowledge-augmented models are shown in table 4.4. Partitioning is performed on the models with an augmentation ratio of 25%, i.e. *Word2Vec init 25% syns* and *Rand init 25% syns*,

Table 4.4: WMD kNN document classification accuracies on the 20 Newsgroups dataset for partitioned knowledge-augmented models

Model	Accuracy
Word2Vec	**62.75% (± 0.89)**
Word2Vec init no-syns	55.01% (± 1.12)
Word2Vec init 25% syns	54.75% (± 1.12)
Word2Vec init 25% syns 25% part	54.78% (± 1.12)
Word2Vec init 25% syns 50% part	55.31% (± 1.12)
Word2Vec init 25% syns 75% part	54.83% (± 1.12)
Rand init no-syns	54.51% (± 1.12)
Rand init 25% syns	54.45% (± 1.12)
Rand init 25% syns 25% part	53.27% (± 1.13)
Rand init 25% syns 50% part	54.45% (± 1.12)
Rand init 25% syns 75% part	54.37% (± 1.12)

since this augmentation ratio is the midpoint of the augmentation ratios explored in these experiments. For the Word2Vec initialised models, these extrinsic evaluation scores seem largely unaffected by the partitioning process since all scores are within statistical bounds of each other. Randomly initialised models seem to be slightly more sensitive to the partitioning process, with the *Rand init 25% syns 25% part* partitioned model achieving slightly lower accuracies than the other randomly initialised models.

Similarity-Distance Correlation scores, presented in table 4.5, show a more evident effect of the partitioning process. In both Word2Vec initialised and randomly initialised models, the 50% partitioning achieves the weakest correlation scores, with the exception of the *WordSim353 Sim* score for the *Rand init 25% syns 50% part* model. In the Word2Vec initialised models, the 75% partition size achieves stronger correlation scores than the other partitioned models across all datasets, and outperforms the non-partitioned *Word2Vec init 25% syns* model in three of the four datasets. In randomly initialised models, there is no single partition size that clearly outperforms the rest. The best performing models per initialisation method are non-partitioned models, i.e. *Word2Vec init no-syns* and *Rand init 25% syns*.

Word Pair Distance Distributions for partitioned models, shown in figures 4.5 and 4.6, do not vary significantly for the different partition sizes within each ini-

Table 4.5: Similarity-distance correlation results with **Euclidean** distance for knowledge-augmented partitioned word embeddings

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.7706	-0.6150	-0.4426	-0.3565
Word2Vec init no-syns	-0.5144	-0.3207	-0.1957	-0.0952
Word2Vec init 25% syns	-0.4365	-0.2589	-0.1951	-0.0467
Word2Vec init 25% syns 25% part	-0.4323	-0.2703	-0.1175	-0.0586
Word2Vec init 25% syns 50% part	-0.3496	-0.2562	-0.0595	-0.0630
Word2Vec init 25% syns 75% part	-0.4378	-0.2808	-0.1500	-0.0924
Rand init no-syns	-0.3694	-0.2347	-0.1206	-0.0817
Rand init 25% syns	-0.4956	-0.2472	-0.1601	-0.0984
Rand init 25% syns 25% part	-0.3666	-0.2113	-0.1366	-0.0599
Rand init 25% syns 50% part	-0.4139	-0.1919	-0.1019	-0.0438
Rand init 25% syns 75% part	-0.3500	-0.2277	-0.1682	-0.0534

tialisation method. In both Word2Vec initialised and randomly initialised models, the biggest difference in word pair distance distributions is observed between the non-partitioned model and the partitioned models. Partitioned Word2Vec initialised models display wider distance distributions with longer tails when compared to the non-partitioned *Word2Vec init 25% syns* model. Word pair distance distributions in partitioned randomly initialised models exhibit similar shapes per word pair group, regardless of partition size, but these shapes differ substantially from the distribution shapes observed in the non-partitioned *Rand init 25% syns*.

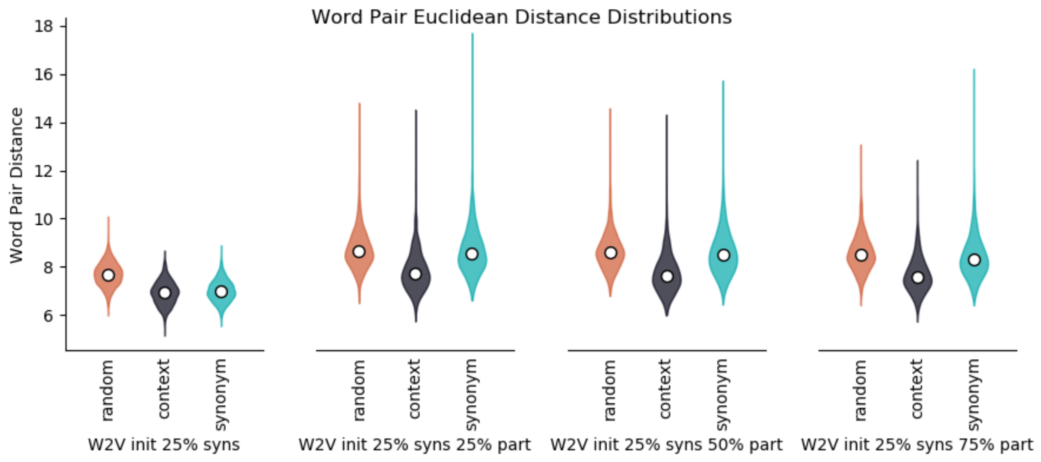


Figure 4.5: Euclidean distance distributions between random, contextual, and synonymous word pair sets for Word2Vec initialised models with a 25% augmentation ratio and different partition sizes

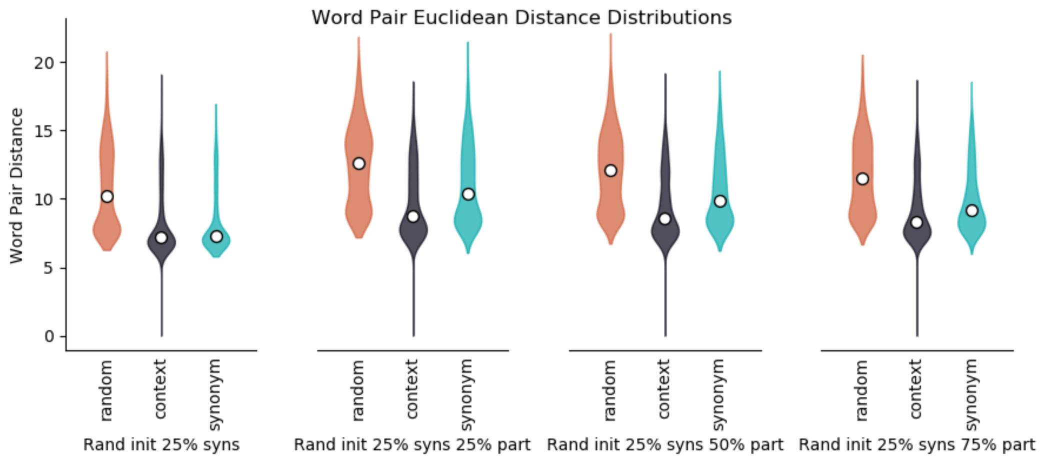


Figure 4.6: Euclidean distance distributions between random, contextual, and synonymous word pair sets for randomly initialised models with a 25% augmentation ratio and different partition sizes

To gain a better insight into the properties of the two partitioned subspaces, the word pair distance distributions are calculated for the augmented and unaugmented partitions separately, i.e. the two subspaces that make up the partitioned embedding. To avoid any artifacts that result from comparing partitions of different sizes (e.g. for 300-dimensional embeddings and a partition size of 25%, the augmented partition will have 75 dimensions and the unaugmented partition will have 225), I focus this exploration on 50% partitions which produces augmented and unaugmented partitions of the same dimensionality, i.e. 150 dimensions per partition. The distance distributions for the partitions of the *Word2Vec init 25% syns 50% part* and *Rand init 25% syns 50% part* models are shown in figures 4.7 and 4.8, respectively. The most significant aspect that is observed is that synonym pair distances seem to be smaller for the augmented partitions, which agrees with my intuition that the augmentation process should bring synonyms closer in embedding space and that this effect should be more accentuated in the augmented subspace. Additionally, both the augmented and unaugmented partitions in these figures preserve the pattern from their full embeddings whereby context pair distances have a smaller mean than synonym pair distances.

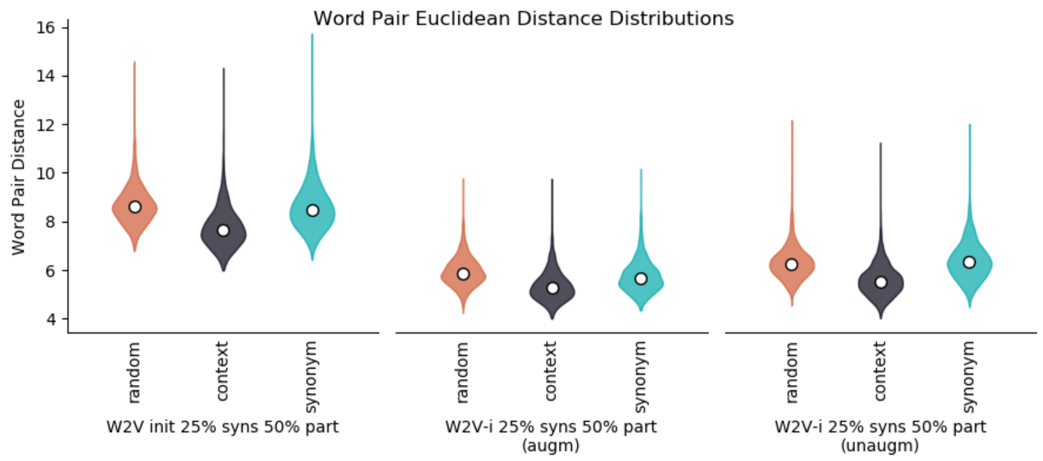


Figure 4.7: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 50% partitioned embeddings

Table 4.6: WMD kNN document classification accuracies on the 20 Newsgroups dataset for *Rand init 25% syns* word embeddings trained with different learning rates (η)

Model	Accuracy
Rand init 25% syns $\eta = 0.1$	54.45% (± 1.12)
Rand init 25% syns $\eta = 0.04$	**56.86% (± 1.12)**
Rand init 25% syns $\eta = 0.003$	55.40% (± 1.12)

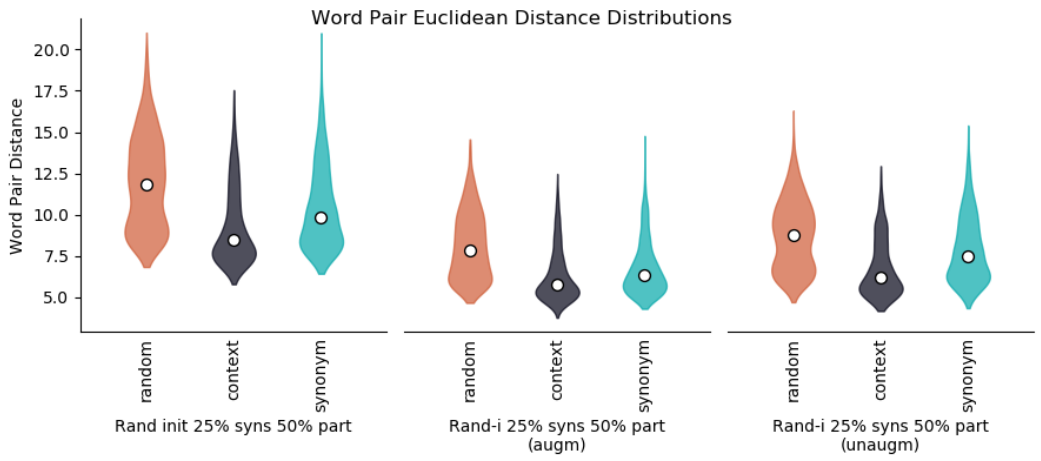


Figure 4.8: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 50% partitioned embeddings

Learning Rates (η)

To better understand how training conditions can affect these word embedding models, I retrained a *Rand init 25% syns* using three different learning rates: $\eta = 0.1$ (used when training all other models in this chapter), $\eta = 0.04$, and $\eta = 0.003$, where all other training conditions remain constant. Table 4.6 shows the WMD document classification scores for these models, where the models with a learning rate of $\eta = 0.04$ achieve a significantly better accuracy than the other two models. The correlation results for these models, presented in table 4.7, show that the model trained with a learning rate of $\eta = 0.04$ also achieves substantially stronger correlation scores across all datasets.

Figures 4.9 and 4.10 further demonstrate the impact that the learning rate can have on the resulting word embeddings. This divergence in model performance for

Table 4.7: Similarity-distance correlation results with **Euclidean** distance for *Rand init 25% syns* word embeddings trained with different learning rates (η)

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Rand init 25% syns $\eta = 0.1$	-0.4956	-0.2472	-0.1601	-0.0984
Rand init 25% syns $\eta = 0.04$	-0.5238	-0.3840	-0.2967	-0.1907
Rand init 25% syns $\eta = 0.003$	-0.0569	-0.0597	-0.0253	-0.0120

different learning rates happens despite observing convergence during the 10 epochs of training, as evidenced in figure 4.11. Even though all models converge, the values for training and validation loss they converge to vary, with $\eta = 0.1$ converging to $\mathscr{L}_{\text{Train}} \approx 2.90$ and $\mathscr{L}_{\text{Val}} \approx 2.93$, $\eta = 0.04$ converging to $\mathscr{L}_{\text{Train}} \approx 2.36$ and $\mathscr{L}_{\text{Val}} \approx 2.50$, and $\eta = 0.003$ converging to $\mathscr{L}_{\text{Train}} \approx 2.51$ and $\mathscr{L}_{\text{Val}} \approx 2.67$, where $\mathscr{L}_{\text{Train}}$ and $\mathscr{L}_{\text{Val}}$ refer to the values of the training and validation loss, respectively. Zooming into the last three epochs of training, shown in figure 4.12, $\eta = 0.003$ seems to exhibit a steeper descent than $\eta = 0.1$ and $\eta = 0.04$, suggesting that that training regime might have benefitted from training for an additional number of epochs.

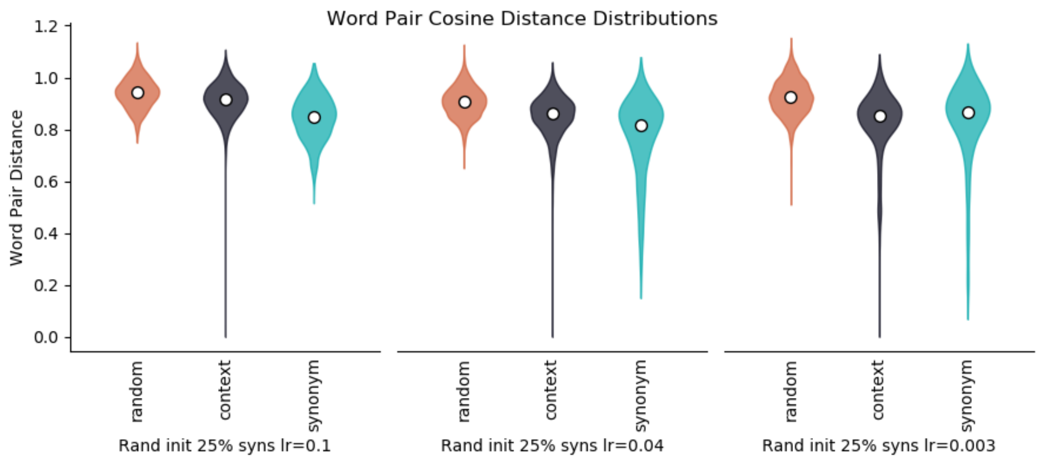


Figure 4.9: Cosine distance distributions between random, contextual, and synonymous word pair sets for *Rand init 25% syns* word embeddings trained with different learning rates (η)

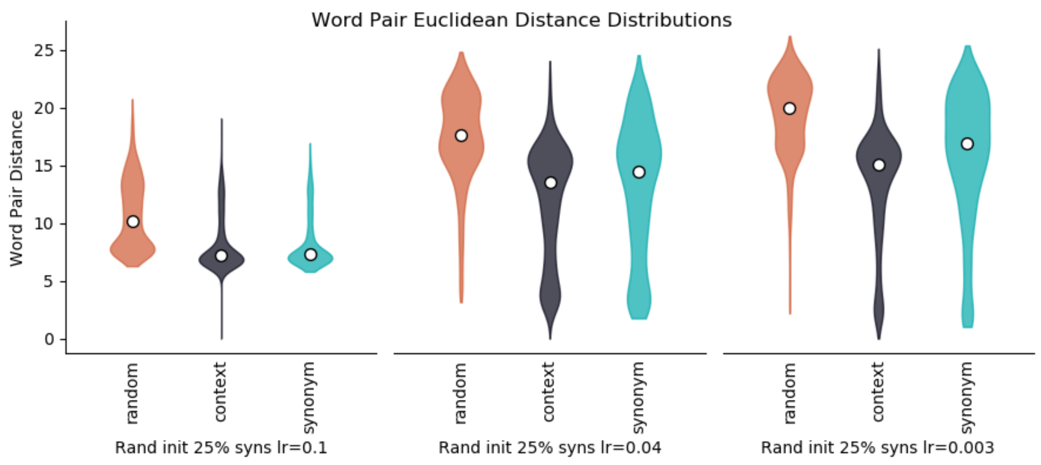


Figure 4.10: Euclidean distance distributions between random, contextual, and synonymous word pair sets for *Rand init 25% syns* word embeddings trained with different learning rates (η)

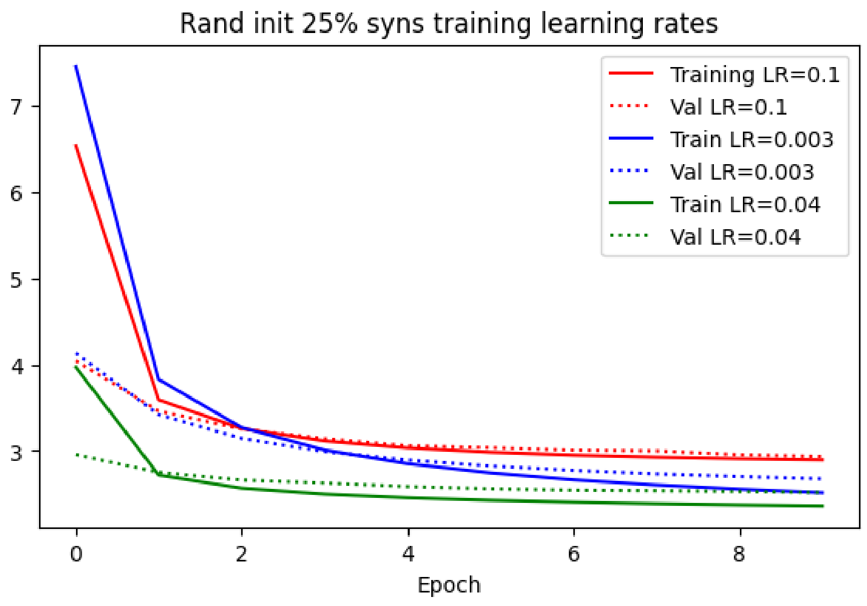


Figure 4.11: Training and validation losses for *Rand init 25% syns* word embeddings trained with different learning rates (η)

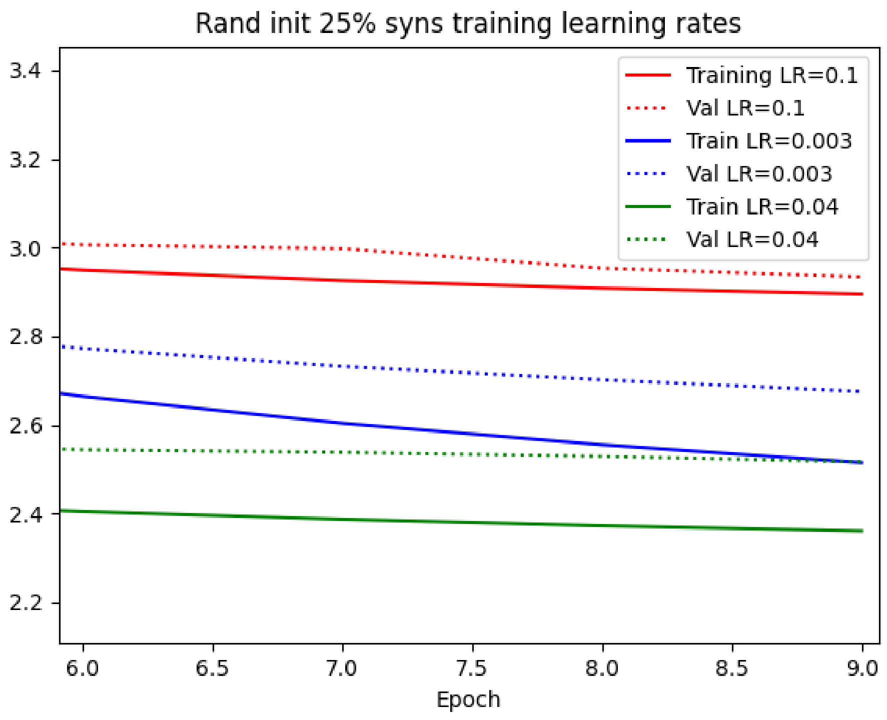


Figure 4.12: Training and validation losses for randomly initialised 25% augmented word embeddings trained with different learning rates (η) zoomed into the last three epochs of training

Analysis

These experiments provide evidence that the synonym-augmentation process during randomly initialised training can potentially help produce *higher quality word embeddings*, defined here as the word embedding models that obtain higher accuracies in the WMD classification task and stronger similarity-distance correlation scores. However, the augmentation process only appears to be beneficial when using the right augmentation ratio, as evidenced in these experiments, where too little augmentation can have a negligible effect and too much augmentation can be detrimental to the quality of the word embeddings. Given that the augmentation process can be interpreted as a noising operation that artificially introduces examples that do not naturally occur in the data, adding too many augmented examples can blur some of the patterns in the original (i.e. unaugmented) data. It is important to note that, in my original formulation of the knowledge-augmentation method, one of the main advantages that I envisioned was that the approach augmented text in a way that respected the natural distribution of words in language and therefore did not rely on additional hyperparameters. However, as evidenced by the results of the experiments in this chapter, the augmentation ratio ended up becoming a de facto hyperparameter that has an important influence on the effectiveness of the augmentation process.

The augmentation process has a different effect on the performance of the Word2Vec initialised models, since none of the augmentation ratios used in this chapter were seen to positively impact them. A possible explanation for this lies in the fact that the pre-trained Word2Vec word embeddings used to initialise these models are already optimised, meaning that they likely represent a specific region of the gradient landscape, e.g. local minima. These initial parameters would therefore act as a strong bias that hampers the model's ability to freely explore the parameter space during training.

Regarding the partitioning process, these experiments do not present any evidence that it can positively impact the learning of synonym-augmented word embeddings. The partitioning process possibly fails to constrain the information flow to only the desired subspace; one possible explanation is the complex interaction between the two embedding matrices (i.e. input and output) that make up the Skip-gram architecture. The partitioning process also appears to have a detrimental effect on synonym distances, pushing embeddings of synonyms further away from each other than in their respective non-partitioned counterparts.

As shown most clearly in the comparison between the word pair distance distributions in figures 4.3 and 4.4, parameter initialisation is the training condition that most dramatically affects the geometry of the learned embeddings, out of the three varying training conditions tested in these experiments. However, considering how similarly the trained models studied in this chapter perform, and the large differences in performance with the pre-trained Word2Vec model, it becomes apparent that the biggest impact on these word embeddings is not the augmentation, partitioning, or initialisation method, but rather other experimental conditions which are kept fixed for all models trained here, such as dataset selection, vocabulary size, number of training epochs, optimisation method, hyperparameter choices, etc.

The impact of training conditions on the learned word embeddings is further confirmed when comparing the performance of the same model trained with three different learning rates, which exhibits a much greater difference in evaluation scores than any of the techniques and approaches explored in this chapter. These observed differences suggest that an accurate comparison between different word embedding models should be carried out on models that are trained under the same training conditions. Finally, to fully understand the impact of the knowledge-augmentation and partitioning techniques proposed in this chapter, hyperparameters should be optimised for the specific variation of the model that is being studied since, for instance, the optimal learning rate for unaugmented models might be different than for augmented models. The results up to this point do not provide clear evidence that the knowledge augmentation process presented in this chapter offers empirical advantages over other existing methods in the literature. A separate set of experiments that focus on optimisation and empirical performance must be conducted in order to better understand how this augmentation process can benefit word embedding models. However, the knowledge augmentation method presented here is representative of a family of approaches that might be useful to practitioners, particularly when other approaches are not suitable.

Chapter 5

Structure-Aware Word Embeddings

Each word in a sentence is not isolated as it is in the dictionary. The mind perceives connections between a word and its neighbors. The totality of these connections forms the scaffold of the sentence.

Lucien Tesniére, *Éléments de syntaxe structurale*

Word co-occurrence models, such as the ones discussed in chapter 4, are efficient to train given the fixed, and typically small, size of contexts they work with. These models work on raw text and require no additional supervision to learn relevant semantic information. Nevertheless, word co-occurrence statistics using fixed-size context windows can fail to capture the fine-grained semantic information found in the syntactic structure and long-range dependencies that are only present in full linguistic structures such as sentences. Shifting from the truncated contexts of word co-occurrence to full sentences has a number of theoretical guarantees. Sentences are natural syntactic units, referred to by Allerton (1969) as the *minimum linguistic units of structural independence*. Sentences can also be considered to be *semantically complete* and the "proper means of expression for a thought", as posed by Gottlob Frege (May, 2006). Within full sentences, words can be analysed in terms of their functional or grammatical roles (e.g. POS tags) or relations (e.g. dependency parse trees).

In this chapter, I explore the potential of structural linguistic information in the learning of *structure-aware word embeddings* by investigating sequential representation learning models. I analyse the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) and Robustly optimised BERT approach (RoBERTa) (Liu et al., 2019) models, two closely related large-scale Transformer models for general-purpose sentence representations. Models like BERT and RoBERTa extract the compositional structure of sentences from their surface form, i.e. they are trained on vast amounts of raw sentences. To examine the effect of explicitly using a sentence's underlying structure to train a representation learning model, rather than relying only on its surface form, I designed **Struct2Seq**, a novel encoder-decoder architecture that takes a **dependency parse tree** as input and attempts to reconstruct the original sentence that produced it. During training, the Struct2Seq model optimises the semantic information that is captured in the learned representations (i.e. word and dependency parse tree embeddings) by learning a mapping between two semantically equivalent structures, since a dependency parse tree constitutes the parsing of a sentence that retains the same underlying semantics.

These three models, BERT, RoBERTa, and Struct2Seq, are trained to learn sentence representations compositionally from sequences of words. At the base of these architectures lies a trainable embedding unit used to represent the input words. The resulting word embeddings can be thought of as being *incidental*, since these models are not explicitly trained to produce optimal word representations. The hypothesis of this chapter is that, as an effect of the learning process, these word embeddings are instilled with significant information about a word's functional and structural linguistic relations.

5.1 Related work

The use of **encoder-decoder** architectures for NLP tasks became widespread with the sequence-to-sequence (Seq2Seq) architecture proposed by Sutskever et al. (2014) to solve a machine translation task. In their setup, a sentence in a source language is processed by an LSTM encoder, which outputs a context vector that is used by an LSTM decoder to predict the translation of that sentence in the target language in an autoregressive fashion, i.e. producing one token at a time based on the output at the last timestep. Seq2Seq architectures have also been employed for other NLP tasks like POS tagging (Zhang et al., 2018), dependency parsing (Kiperwasser and Goldberg, 2016; Li et al., 2018), and NER tagging (Lample et al., 2016).

In the context of representation learning, Kiros et al. (2015) exploit the representational capabilities of Seq2Seq architectures to learn sentence embeddings by setting up a **Skip-Thought** prediction task, where a full sentence is processed by an RNN encoder, and two separate decoders attempt to reconstruct the previous and next sentences, respectively. **Context Vectors (CoVe)**, proposed by McCann et al. (2017), produce context-aware word embeddings by training a Seq2Seq machine translation model to translate from English to German, and using the trained Bidirectional LSTM (BiLSTM) encoder to embed words that are aware of their semantic context. Wieting and Gimpel (2017) take a similar approach in which sentence embeddings are trained by maximising the cosine similarity between the embeddings of paraphrase pairs. The **Sequential (Denoising) Autoencoder** proposed by Hill et al. (2016), a Seq2Seq denoising autoencoder where the input is a corrupted sentence (i.e. some words are randomly removed or shuffled), is the approach most similar to the experiments described in this chapter, since the input dependency parse tree can be thought of as a different type of noising operation applied to the input sentence.

Sentence representation learning approaches based on the Transformer architecture have produced transferable sentence embeddings that have helped achieve[1] state-of-the-art results across several NLP tasks. These transferable sentence embeddings have been commonly trained by learning a language model, such as predicting a missing word from a sentence (Devlin et al., 2018), or learning *inter-sentence* dependencies for sentence and document embeddings (Liu and Lapata, 2018). Some of these approaches complement language modelling with a fine-tuning stage (Cer et al., 2018; Radford et al., 2018, 2019). All of these sentence embedding methods learn word embeddings as a byproduct of both the RNN and

[1]Some of these models are reported to have achieved state-of-the-art results in and of themselves. However, in many of the cases, they have achieved these results when used as input representations for task-specific models.

Transformer architectures they employ. As evidenced by the empirical results obtained by Embeddings from Language Models (ELMo) representations (Peters et al., 2018), the internal token and context representations of an LSTM can produce context-aware word embeddings that are useful for downstream NLP tasks.

LSTMs and Transformers are designed to capture interactions between elements in a sequence. However, the specific interactions they capture are neither entirely interpretable nor well-understood. Recurrent units, even ones with more complex memory cells, emphasise sequentially proximal information, while (simple) attention units that assign probabilities to positional elements are not fully able to capture structural dependencies, as described by Liu and Lapata (2018). Even though in theory these architectures could learn to model structural relations, as argued by Kim et al. (2017), McCoy et al. (2020) found that Seq2Seq architectures with different variants of RNNs and attention mechanisms had a bias towards *linear order*, while tree-based architectures, such as the Tree-LSTM, had a *hierarchical-syntactic* bias, which they describe as underlying Chomsky's 1965 model of human language acquisition.

The strong structural bias of tree-based architectures, together with their ability to handle tree-structured input, has sparked a renewed interest in tree structures for NLP for tasks such as dependency parsing (Kiperwasser and Goldberg, 2016), NLI (Mou et al., 2016b; Chen et al., 2017), or text generation (Guo et al., 2018). Tree-based architectures have also been amply used in encoder-decoder setups. **Tree-to-tree** architectures have been applied to the problem of translating computer programs from one language to another (Chen et al., 2018), molecular graph generation (Jin et al., 2018), or generating programs from natural language (Alvarez-Melis and Jaakkola, 2017), among others. **Tree-to-sequence**, and the closely related **graph-to-sequence**, architectures have been used in text generation (Damonte and Cohen, 2019; Beck et al., 2018), constituency parsing (Guo et al., 2018), and machine translation (Eriguchi et al., 2016; Beck et al., 2018).

Lastly, Vashishth et al. (2019) propose two methods that use Graph Convolutional Networks (GCN) to train word embeddings by exploiting the syntactic (*Syn-*

GCN), or semantic (*SemGCN*) information. Their SynGCN variant is similar to the approach I propose in this chapter in that they both exploit dependency parse trees to learn word embeddings that are aware of their syntactic context, as opposed to the *sequential context* captured by fixed context window approaches. Unlike my approach, which is based on autoencoding, the objective of SynGCN is based on predicting a target word given its context, where the context is a learned representation of the dependency parse tree provided by the GCN.

5.1.1 Positioning

The structure-aware word embedding model I propose in this chapter seeks to harness the information extraction potential of tree-structured RNN architectures. While many of the models presented here use tree-based architectures to learn sentence-level representations, my approach focuses on atomic word representations that are instilled with information about the syntactic structures in which they are used. Additionally, unlike some of the tree- and graph-based methods presented in this section which focus exclusively on syntactic structures, the training objective of my approach is designed to jointly learn a word's syntactic and sequential contexts. By exploiting both word co-occurrence statistics and linguistically-informed contexts, the model I propose seeks to learn word representations that capture more syntactic information, even when trained on smaller datasets.

5.2 Background

5.2.1 Background on Transformer Models

Transformers (Vaswani et al., 2017), which have recently become one of the most widely used architectures in NLP, consist of stacks of *multi-head self-attention* layers. Self-attention (Lin et al., 2017) refers to an attention mechanism applied within a single sequence, i.e. a set of attention weights that relate the elements of a sequence to the elements of the same sequence. BERT and RoBERTa leverage the expressibility of a multi-layer Transformer architecture to model the syntactic information contained in sentences without the need for any explicit structural information. The intuition behind these models is that, when applied to natural language,

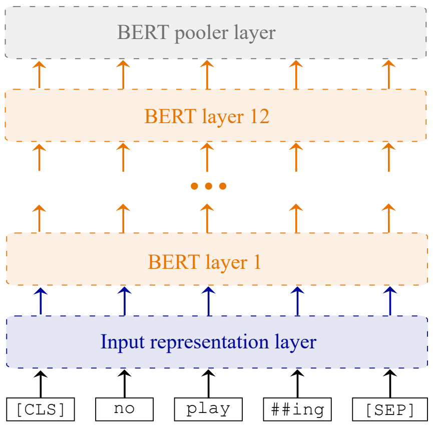


Figure 5.1: General BERT architecture, where BERT layers are made up of Transformer encoders, and the BERT pooler is a fully connected layer that outputs token probabilities.

Transformers have the capacity to model intrasentential interactions, i.e. functional relations between the words in a sentence, and can therefore produce sentence representations that capture the inner structure of the represented sentence.

Both the BERT and RoBERTa models share the same underlying architecture, shown in figure 5.1. BERT and RoBERTa train different model sizes, but these experiments are based on their *BASE* architecture, made up of 12 layers with 768-dimensional hidden layers and 12 self-attention heads, adding up to a total of 110 million trainable parameters. BERT is trained on two learning tasks: the first is a *masked language modelling* task, where a random word in a sentence is masked and the model is tasked with predicting the masked word; and the second task is a *next sentence prediction* binary classification task, where given two sentences, the model must predict whether they are contiguous or come from different parts of the corpus. The masked language modelling task forces the model to consider both past and future contexts concurrently when building its sentence representations, while the next sentence prediction task is meant to provide the model with inter-sentence information. RoBERTa is trained only on the masked language modelling task, as Liu et al. (2019) found in their experiments that the next sentence prediction task was not improving, and in some cases was hurting, the performance of the model.

In order to keep a reduced vocabulary that has high coverage of the text corpus, these models use **subword tokenisation**, which creates a vocabulary of the

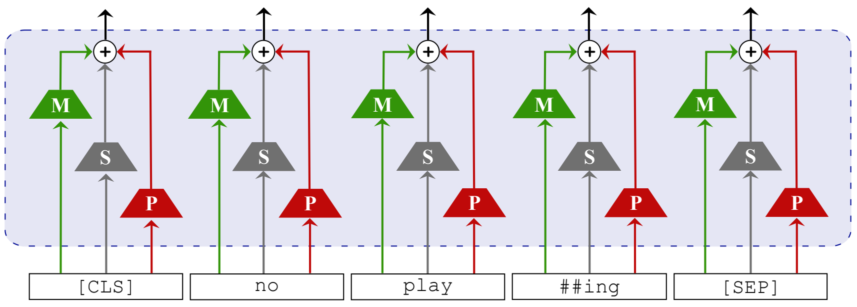


Figure 5.2: Input representation layer of the BERT architecture. Every token receives a token embedding (**M**), a segment embedding (**S**), and a positional embedding (**P**)

most frequently appearing subwords. Both BERT and RoBERTa use slightly different subword tokenisation methods, BERT uses *WordPiece tokens*, while RoBERTa uses *byte-level BPE tokens* (both of these tokenisation methods are described in section 3.1). For instance, while a common word like `dog` might have a corresponding token in both the WordPiece and BPE subword vocabularies, a rare word like `zygomatic` might get broken down into different subword tokens, where the WordPiece tokenisation will produce the tokens `z`, `##y`, `##go`, `##matic`; and BPE will produce `_zy`, `g`, `omatic`. Aside from the learning task and the tokenisation method, RoBERTa mainly differs from BERT in the choice of *training conditions*, which include increasing the size of the mini-batches used during training, masking tokens dynamically (as opposed to masking the dataset once and cycling over the same masks), and using a single full sentence as input (instead of the concatenated pair of sentences BERT uses for its next sentence prediction task).

The two models use an input representation layer, which combines learned **token embeddings** (**M**) with learned **positional embeddings** (**P**) and, in the case of BERT, **segment embeddings** (**S**), i.e. embeddings for each of the two concatenated input sentences, as shown in figure 5.2. For the experiments in this chapter, I use the token embeddings **M** as the standalone structure-aware word embedding.

5.2.2 Background on the Struct2Seq Model

Learning word embeddings that contain information about their full semantic context requires an architecture that is capable of extracting semantic information from the full sentence in which the words occur. RNNs are well equipped for this task,

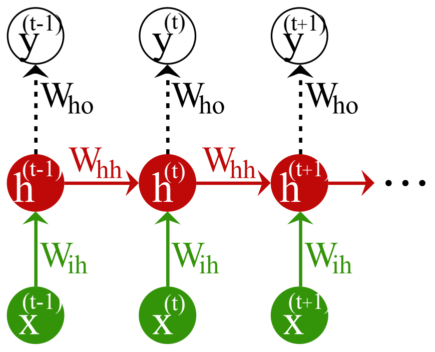


Figure 5.3: A basic RNN architecture, where $\mathbf{x}^{(t)}$, $\mathbf{h}^{(t)}$, and $\mathbf{y}^{(t)}$ denote the *input*, *hidden layer*, and *prediction*, respectively, at timestep t. The weight matrices are reused across all timesteps and represent connections between the input and hidden layers, $\mathbf{W}_{ih}$, the hidden layer and the prediction, $\mathbf{W}_{ho}$, and the hidden layers in consecutive timesteps, $\mathbf{W}_{hh}$.

since they were originally designed to process sequences of arbitrary length. A basic RNN consumes a sequence of ordered elements $S = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)} \rangle$ one element at a time, and reuses the same parameter matrices $\mathbf{W}_{ih}$ and $\mathbf{W}_{hh}$ at each timestep to produce a hidden state $\mathbf{h}^{(t)}$ for timestep t that factors in the input at that timestep $\mathbf{x}^{(t)}$ together with the hidden state at the previous timestep:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \Theta) \tag{5.1}$$

where the parameters $\Theta = \{\mathbf{W}_{ih}, \mathbf{W}_{hh}\}$ include connections from the input word $\mathbf{x}^{(t)}$ to the hidden state $\mathbf{h}^{(t)}$, $\mathbf{W}_{ih}$, and recurrent connections between contiguous hidden states $\mathbf{W}_{hh}$. Hidden state $\mathbf{h}^{(t)}$ contains a representation of the sequence up to timestep t. A simple RNN is depicted in figure 5.3; other variants of this architecture can produce output only at the last timestep, or include connections between outputs and hidden states (Goodfellow et al., 2016).

Like all other neural networks, RNNs (and variants) learn an internal representation that is (approximately) informationally optimal for their training objective, meaning that these learned representations efficiently store the information that is most relevant to solve their learning task. In text classification tasks, where the output is a single class, the hidden state is tasked with capturing the most salient features in the input with respect to the classification task. In that setting, the RNN's hidden state acts as a *feature detector over time*. This task requires the RNN to learn the features that are most indicative for the classification problem, but there is no guarantee as to the amount of information about the original input that will be preserved by its learned representation. In terms of learning transferable representations, this type of task-specific feature learning fails to capture the general linguistic patterns that might be relevant to solve other related tasks.

Sequence **encoder-decoders**, also known as **Seq2Seq** architectures (Sutskever et al., 2014), offer an alternative that is designed to retain as much information from the full sequence in ther hidden state as possible. In a **Seq2Seq** architecture, the input is processed by an **encoder** to produce a **context vector** that is passed as input into a **decoder** which, in turn, learns to produce an approximation of the **target sequence**. In machine translation, where Seq2Seq architectures have been widely used, the context vector must preserve as much information about the semantic structure of the input text as possible in order for the decoder to produce a close translation. The input sequence can be replicated as the target to convert this architecture into a **sequence autoencoder** (shown in figure 5.4) which, unlike the machine translation setup, is unencumbered by an additional grammar for a target language. This **reconstruction objective** exploits the information bottleneck[2] structure of the autoencoder since, by seeking to recover the orignal input from a compressed representation, the learned representation is forced to capture as much information from the original sequence as possible. However, this objective emphasises the sequential structure of the textual input, which is sensitive to linguistic domain and syntactic style. In order to maximise their transferability, word embeddings should, in principle, favour information about a word's semantics and its grammatical role. This information is more linguistically invariant and therefore useful in a wider set of linguistic domains.

In the context of sequential autoencoders, I hypothesise that dependency relations can be integrated into the autoencoding process to emphasise the capturing

[2]The *information bottleneck* term was first introduced by Tishby et al. (1999) and is broadly defined as the process of finding a *short code* $\hat{X}$ for an input X such that $\hat{X}$ preserves as much *meaningful information* about another variable Y as is possible given the availables bits in $\hat{X}$, where Y, also called the *relevance variable*, is dependent on X (i.e. the mutual information of X and Y is positive).

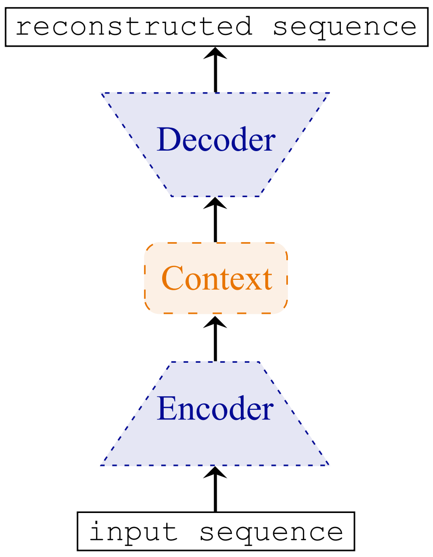


Figure 5.4: Generic sequence autoencoder architecture

of sentence-level syntactic information in the learned representations. A **dependency parse tree** is a representation of a sentence in terms of the (typed, binary, and directed) **grammatical relations** between its constituting words, and is a full description of the underlying syntactic structure of that sentence. Dependency parse trees are useful in this setting because they are entirely based on pairwise relations between words, unlike constituency parse trees, which deal with recursive compositional phrase structures. Additionally, dependency parse trees are less sensitive to order information, which allows them to focus on grammatical relations that hold irrespective of the linguistic domain in which they appear. To exemplify this, take the compound sentence `For many of the farmers involved, the news is devastating`, which is semantically equivalent to a rearrangement of its independent clauses: `The news is devastating for many of the farmers involved`. For this particular example, the semantic similarity between these two sentences is more closely captured by their dependency parse trees (figure 5.5) than by their constituency parse trees (figure 5.6). The temporal processing performed by a straightforward RNN architecture favours the order information captured by constituency parse trees, but incorporating the non-proximal relations contained in dependency parse trees can help these sequential models capture the underlying syntactic structure of a sentence.

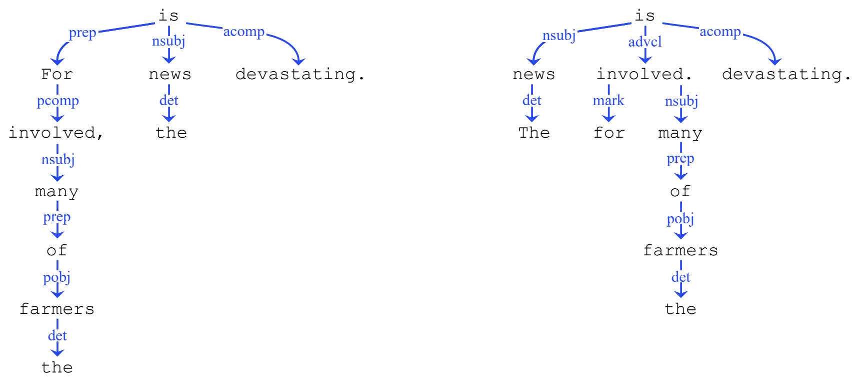


Figure 5.5: Dependency parse trees for the sentences `For many of the farmers involved, the news is devastating.` (left) and `The news is devastating for many of the farmers involved.` (right)

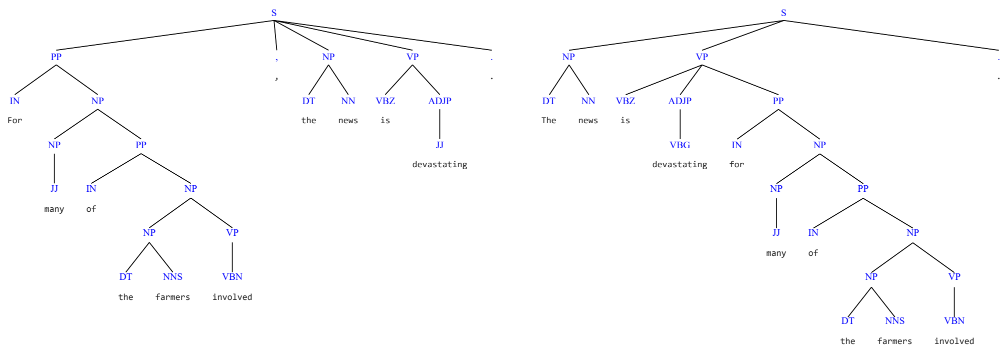


Figure 5.6: Constituency parse trees for the sentences `For many of the farmers involved, the news is devastating.` (left) and `The news is devastating for many of the farmers involved.` (right)

Even though dependency parse trees have the desirable property of (some degree of) stylistic invariance, they achieve it at the expense of word order information from the original sequence. Word order is relevant to the construction of word embeddings since it informs about word usage. Ideally, learning **transferable** and **structure-preserving** word embeddings should integrate their more invariant interactions (dependency relations) with their stylistic usage (positional information). To achieve this, the **Struct2Seq** architecture that I propose in this chapter incorporates the ideas from the Seq2Seq architecture and the additional grammatical information that is obtained from dependency parse trees. Struct2Seq is, in essence, the inverse formulation of an end-to-end neural dependency parser in that it takes a dependency parse tree as input and is tasked with recovering the original sentence for that dependency parse tree. Figure 5.7 shows the general Struct2Seq architecture for the sample sentence `He experimented on pea plants`.

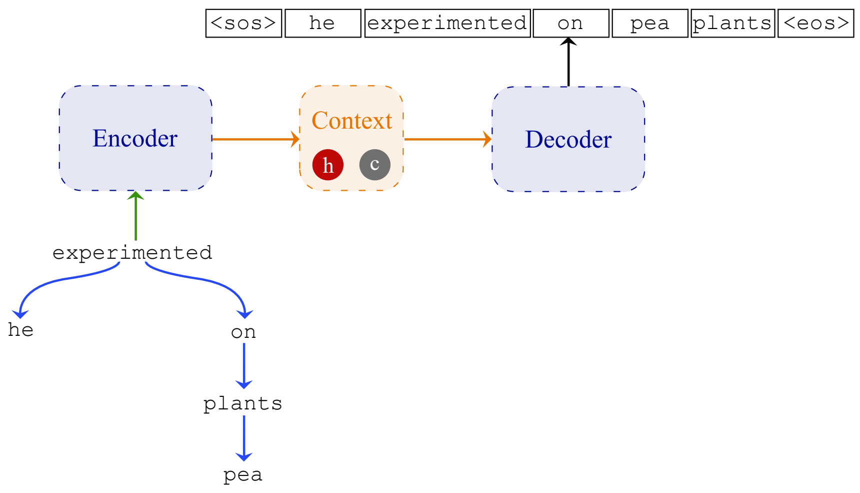


Figure 5.7: Struct2Seq architecture

Whereas the internal representations of a Seq2Seq autoencoder are employed as **memory units** that keep track of the words that appeared in the input to be reconstructed, the internal representations of the Struct2Seq architecture double as a **mapping from dependency relations to word order**. By inverting the order of a dependency parser, the expectation is that the learned representations of the encoder place a bigger emphasis on the dependency structure. Additionally, learning a tree decoder is significantly harder than learning a sequence decoder, so by simplifying the decoding task, the model can focus on the representation learning process. The main reasoning behind providing syntactic information as part of the input to the Struct2Seq autoencoding process is that, as the syntactic information flows through the tree-structured encoder, the token embeddings capture some of the information needed to model the syntactic relations. The expectation with this setup is that token embeddings will contain dependency information from their observed contexts during training. After training concludes, the learned token embeddings can be used as standalone word representations that contain information about a word's observed syntactic contexts without the need to provide syntactic information at the time of embedding.

The decoder unit of the Struct2Seq architecture is implemented with the **LSTM** (Hochreiter and Schmidhuber, 1997) RNN variant. The original LSTM implements a **memory gate** which allows it to selectively retain or *forget* information from the full history, which makes it better at modelling long-range dependencies than a standard RNN. This memory gate, shown in figure 5.8, is implemented with the following equations:

$$\begin{aligned}
\mathbf{i}^{(t)} &= \sigma(\mathbf{W}_i\mathbf{x}^{(t)} + \mathbf{U}_i\mathbf{h}^{(t-1)} + \mathbf{b}_i) \\
\mathbf{f}^{(t)} &= \sigma(\mathbf{W}_f\mathbf{x}^{(t)} + \mathbf{U}_f\mathbf{h}^{(t-1)} + \mathbf{b}_f) \\
\mathbf{o}^{(t)} &= \sigma(\mathbf{W}_o\mathbf{x}^{(t)} + \mathbf{U}_o\mathbf{h}^{(t-1)} + \mathbf{b}_o) \\
\mathbf{u}^{(t)} &= \tanh(\mathbf{W}_u\mathbf{x}^{(t)} + \mathbf{U}_u\mathbf{h}^{(t-1)} + \mathbf{b}_u) \\
\mathbf{c}^{(t)} &= \mathbf{i}^{(t)} \odot \mathbf{u}^{(t)} + \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} \\
\mathbf{h}^{(t)} &= \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)})
\end{aligned} \tag{5.2}$$

where $\mathbf{x}^{(t)} \in \mathbb{R}^D$ is the input vector at timestep t, $\mathbf{h}^{(t-1)}, \mathbf{c}^{(t-1)} \in \mathbb{R}^H$ are the hidden and cell vectors at the prevous timestep, $\mathbf{i}^{(t)}, \mathbf{f}^{(t)}, \mathbf{o}^{(t)}, \mathbf{u}^{(t)} \in \mathbb{R}^H$ are internal vectors, σ is a non-linear function, tanh is the hyperbolic tangent function, $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_u \in \mathbb{R}^{D\times H}$ and $\mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_o, \mathbf{U}_u \in \mathbb{R}^{H\times H}$ are trainable parameter matrices, and $\odot$ is the Hadamard or element-wise product.

The encoder of the Struct2Seq uses the Tree-LSTM architecture by Tai et al. (2015) to deal with its tree-structured input. The Tree-LSTM is an extension to the LSTM architecture designed to deal with trees. More specifically, the encoder uses a variant of this architecture, termed Child-Sum Tree-LSTM, that is capable of dealing with an arbitrary number of children for each node, which is necessary when processing dependency parse trees.[3] The transition equations for the Child-Sum Tree-LSTM, shown in figure 5.9, are similar to those of the LSTM, but where,

[3]The alternative formulation, the N-ary Tree-LSTM, is designed to deal with trees where nodes have at most N (ordered) children.

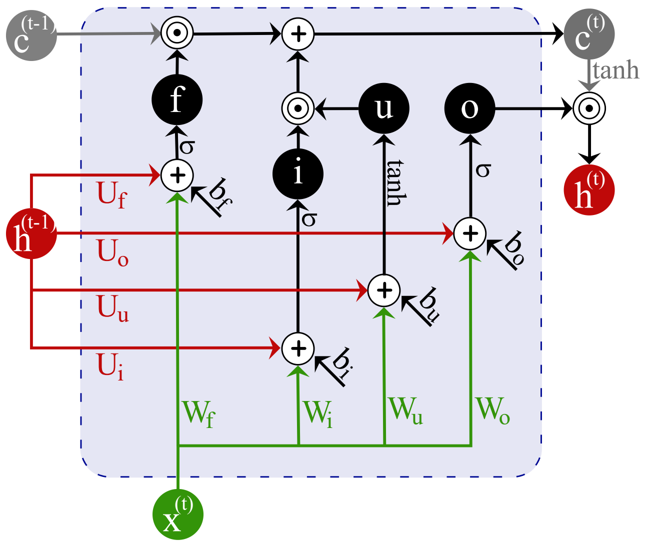


Figure 5.8: Architecture of a single LSTM cell, where green arrows denote the flow of the input signal, red arrows trace the flow of the previous hidden state, gray arrows show the flow of the previous cell state, and black arrows and circles correspond to the internal operations and variables of the cell.

instead of timestep t, processing follows the traversal of the nodes of a tree. These are the equations for node j, with children $\mathscr{C}(j)$:

$$\begin{aligned}
\tilde{\mathbf{h}}^{(j)} &= \sum_{k\in\mathscr{C}(j)} \mathbf{h}^{(k)} \\
\mathbf{i}_j &= \sigma(\mathbf{W}_i\mathbf{x}^{(j)} + \mathbf{U}_i\tilde{\mathbf{h}}^{(j)} + \mathbf{b}_i) \\
\mathbf{f}^{(jk)} &= \sigma(\mathbf{W}_f\mathbf{x}^{(j)} + \mathbf{U}_f\mathbf{h}^{(k)} + \mathbf{b}_f) \\
\mathbf{o}^{(j)} &= \sigma(\mathbf{W}_o\mathbf{x}^{(j)} + \mathbf{U}_o\tilde{\mathbf{h}}^{(j)} + \mathbf{b}_o) \\
\mathbf{u}^{(j)} &= \tanh(\mathbf{W}_u\mathbf{x}^{(j)} + \mathbf{U}_u\tilde{\mathbf{h}}^{(j)} + \mathbf{b}_u) \\
\mathbf{c}^{(j)} &= \mathbf{i}^{(j)} \odot \mathbf{u}^{(j)} + \sum_{k\in\mathscr{C}(j)} \mathbf{f}^{(jk)} \odot \mathbf{c}^{(k)} \\
\mathbf{h}^{(j)} &= \mathbf{o}^{(j)} \odot \tanh(\mathbf{c}^{(j)})
\end{aligned} \tag{5.3}$$

where most elements remain the same as in equation 5.2 with the exception of $\tilde{\mathbf{h}}^{(j)} \in \mathbb{R}^H$ which is the hidden state for the current node calculated as the sum of its child nodes (hence the name *Child-Sum* Tree-LSTM), and a $\mathbf{f}^{(jk)}$ vector calculated for every child k of node j.

The Tree-LSTM learns a representation for every node in the tree composition-

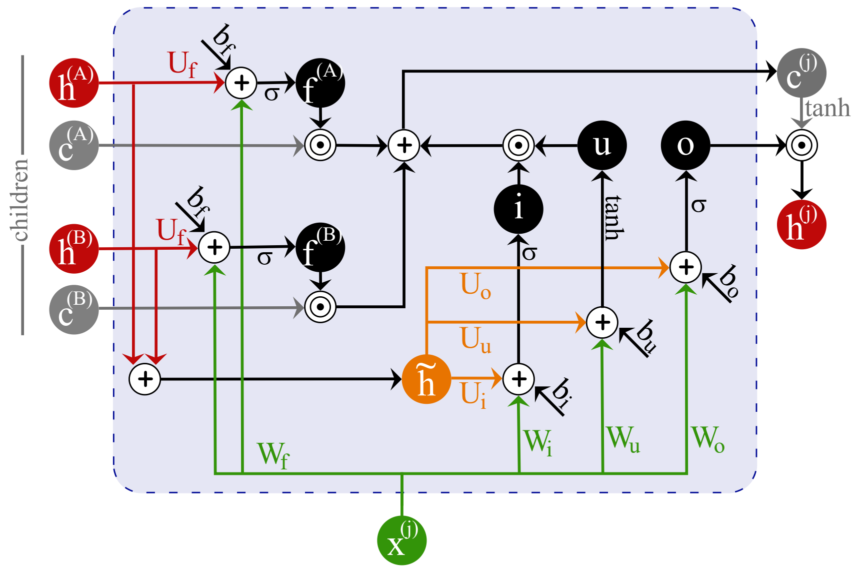


Figure 5.9: Architecture of a single Tree-LSTM cell, where green arrows denote the flow of the input signal, red arrows trace the flow of the hidden states of child nodes, gray arrows show the flow of the cell states of child nodes, orange arrows show the flow of the combined *parent* hidden state, and black arrows and circles correspond to the internal operations and variables of the cell.

ally, such that every node is represented as a function of its input and child nodes, and the representation of the root of the tree is also a representation of the full tree. To learn word embeddings that capture the general semantic and syntactic information from this pipeline, but are encapsulated from the rest of the words in the sentence, I prepend an additional trainable embedding layer **M** to the encoder that embeds the words before they are input into the Tree-LSTM.

5.3 Struct2Seq Learning Algorithm

In this section, I describe the full learning algorithm to train and evaluate structure-aware word embedding models.

5.3.1 Task, dataset, and architecture definition

The goal of these experiments is to learn structure-aware word embeddings. To achieve this, the **learning task** is that of reconstructing a sentence from its corresponding dependency parse tree. The general probabilistic model can be expressed as the probability of a sequence $S = \langle w_1, \ldots, w_T \rangle$ given a dependency parse tree G:

$$p(S|G)$$

As in the experiments in chapter 4, the **dataset** used for these experiments is the BNC dataset described in section 3.4.1. Due to computational constraints, and to be consistent with the experiments in chapters 3 and 4, the **Struct2Seq** models in this chapter are trained on the same subset of 10% of the sentences randomly sampled from the full dataset, as detailed in section 3.4.1.

5.3.2 Preprocessing

The Struct2Seq model training requires a set of sentence-dependency parse tree pairs. To construct a dataset that has this format, I add a dependency parsing step to the preprocessing pipeline described in section 4.3. Aside from the dependency parsing step, the rest of this preprocessing is essentially the same as that in section 4.3:

1. **Concatenate all BNC documents** into a single continuous raw text corpus
2. **Tokenise sentences** following BNC XML tags
3. **Remove XML information** since BNC tags are not used in these experiments
4. **Shuffle sentences** (since the BNC data is organised into topics, shuffling ensures that this training corpus has interspersed sentences from all topics) and **retain a subset** of 10% of the sentences in the data (601,818 sentences)
5. **Dependency parsing and word tokenisation** using the spaCy library (Honnibal and Montani, 2017) to produce the dependency parse trees for all tokenised sentences
6. **Convert text to lower case** to minimise token variations
7. **Remove punctuation** to focus the training on words and their relationships (this avoids having to encode the relations between words and punctuation marks)

8. **Convert numeric tokens** to a generic format, e.g. `12.45` is converted to `##.##`, to preserve the structure of numeric tokens while reducing their variability

9. **Prune sentences** containing a single word since they have no contextual information (37,201 sentences removed)

5.3.3 Vocabulary construction

For consistency with the rest of the experiments and baseline models used in this thesis, the vocabulary used in the experiments in this chapter is the same as the vocabulary described in chapters 3 and 4. To reiterate, the vocabulary used here is based on the 10% subset of the full BNC dataset, where words that appear fewer than 5 times in the dataset are pruned. After pruning, the resulting vocabulary is made up 45,769 unique tokens, which cover 32.45% of unique tokens and 98.48% of non-distinct tokens in the dataset.

5.3.4 Datapoint construction

The fully preprocessed dataset consists of dependency parse trees and tokenised sentences. From there, the construction of a datapoint is straightforward and consists of using the dependency parse tree as the input and the tokenised sentence as the target. Figure 5.10 shows the dependency parse tree for the sentence `He experimented on pea plants`, and listing 5.1 shows the corresponding JSON formatted datapoint for the sequence-dependency parse tree pair. The dependency parse tree used for the input is simplified for these experiments by removing the connection types.

Listing 5.1: Sequence-dependency parse tree datapoint in JSON format

```
{"seq": ["he", "experimented", "on", "pea", "plants"],
"tree":
  {"word": "experimented", "children": [
    {"word": "he", "children": []},
    {"word": "on", "children": [
```

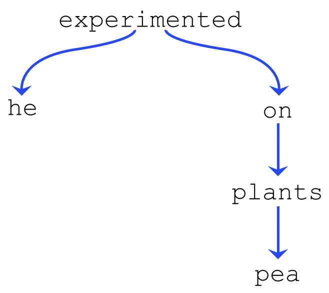


Figure 5.10: Dependency parse tree for the sentence He experimented on pea plants (dependency types are omitted)

```
      {"word": "plants", "children": [
        {"word": "pea", "children": []}
      ]}
    ]}
  ]}
}
```

5.3.5 Input representation

The Struct2Seq architecture requires the constructed datapoints, like the one shown in listing 5.1, to follow a particular numericalised format. The first formatting step consists of numericalising the words in the sequence and the tree by replacing each word with its index from the vocabulary. The sentence He experimented on pea plants would therefore be translated to the sequence of indices [18, 20270, 17, 19705, 1634], as shown in the "seq" entry of listing 5.2. The dependency parse tree is then decomposed into its constituting elements. The "features" element represents the tokens in the dependency parse tree in depth-first traversal order. In the example sentence and its dependency parse tree shown in figure 5.10, the resulting traversal order would be [experimented, he, on, plants, pea], and its corresponding numericalised representation, [20270, 18, 17, 1634, 19705], as it appears in ["tree"]["features"] in listing 5.2. The "levels" and "adjacency_list" features describe the structure of the tree in terms of the depth of a node in the tree (e.g. the root word experimented is at the 0th level) and the edges that connect the nodes in the tree (e.g. an adjacency of [0, 2] indicates a connection between the root node and the second node, which corresponds to on), respectively. Finally, "node_order" and "edge_order" describe the traversal order of the nodes and edges in the tree, from leaves to root. Every numericalised datapoint, i.e. sequence-dependency parse tree pair, is assigned an identifier, "id", that links it to its pre-numericalised form. The resulting datapoint in JSON format is shown in listing 5.2.

Listing 5.2: Input representation for sequence-dependency parse tree datapoint

```
{"id": 6,
"seq": [18,20270,17,19705,1634],
"tree": {
      "features": [20270,18,17,1634,19705],
      "levels": [0,1,1,2,3],
      "node_order": [3,0,2,1,0],
      "adjacency_list": [[0,1],[0,2],[2,3],[3,4]],
      "edge_order": [3,3,2,1]
      }
}
```

5.3.6 Dataset split

The tokenised sentences in the preprocessed dataset are shuffled before the dataset is split into a **training set** made up of **80%** of the sentences in the full dataset, which amounts to 450,314 sentences, and the remainder of the data is split evenly between a **validation set** (**10%**) and a **test set** (**10%**), containing 56,291 sentences each. This choice of training-validation-test split allows for an efficient use of training data when working with moderately sized datasets.

5.3.7 Model training

5.3.7.1 Pre-epoch processing

The *training* and *validation* datasets are **shuffled** at the beginning of every epoch. Training is carried out for an arbitrary number of epochs in order to understand the behaviour of the model's training, early stopping checks are omitted. Mini-batch processing was explored in preliminary checks and displayed a slower rate of convergence, so these models are trained on a **single datapoint** at a time.[4]

5.3.7.2 Data processing

A dependency parse tree is input into the Struct2Seq architecture, which initially embeds every word in the tree with a trainable embedding matrix $\mathbf{M}$. It then computes a representation for every node in the tree compositionally until it reaches the root. The root representation, or *context vector*, which contains information about the entire tree, is then decoded one step at a time into the output sequence. As a guide to the decoder, a start-of-sequence token `<sos>` is added at the beginning of the target sequence and an end-of-sequence token `<eos>` is added at the end. The decoder learns to predict the `<eos>` token, which indicates it has to stop producing new tokens, as a way of learning the length of the target sequence (which has the same number of elements as the input tree). The output sequence is produced one step at a time using the (updated) context vector and the generated output at the previous timestep. Given this dependence on the previous output, producing an incorrect token can increase the probability of producing wrong tokens for all future timesteps. **Teacher forcing** (Williams and Zipser, 1989) is used to decrease the effect of an incorrect token, which is a process that exchanges the predicted output at the previous timestep, $\mathbf{y}^{(t-1)}$, with the target for that timestep, $\tau^{(t-1)}$, with a given probability. For these experiments, I found the best performing teacher forcing ratio to be `tf=0.5`, meaning it will use the previous target $\tau^{(t-1)}$ instead of the predicted input $\mathbf{y}^{(t-1)}$ with a probability of 0.5. The full unfolded architecture for the sentence `He experimented on pea plants` is shown in figure 5.11.

An additional **maximum sequence length** parameter, `max_seq_len=60`, is set to skip longer sentences, which removes 30,384 sentences (around 5% of the full dataset). This reduces the memory requirement, since the BPTT algorithm requires

[4]Batch processing in this setting requires concatenating multiple dependency parse trees as a single sequential input, and the target consists of a sequence of concatenated target sequences. Since this architecture seeks to recover the words and order information, this concatenation significantly increases the complexity of the learning task, as evidenced by preliminary tests in which the reconstruction error increased substantially as the number of datapoints in each batch increased.

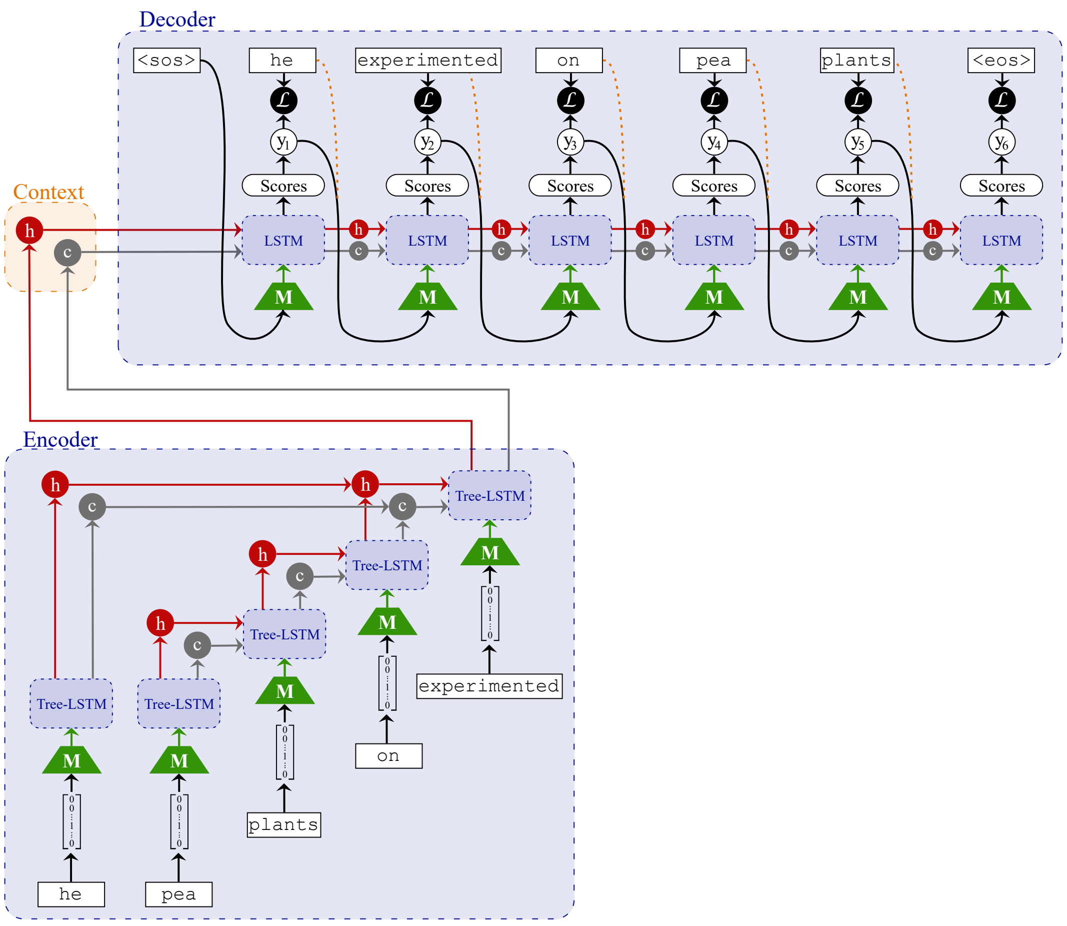


Figure 5.11: Unfolded Struct2Seq architecture to process the sentence `He experimented on pea plants`

the full computational graph to be unfolded to perform back-propagation over all of the timesteps in the sequence. Limiting the length of the sequence also reduces the complexity of the modelling process, which can speed up the convergence of the model's training.

5.3.7.3 Error calculation

The target sequence in these experiments is an ordered sequence of words $S_\tau = \langle w^{(0)}, w^{(1)}, \dots, w^{(T)} \rangle$, where $w^{(t)}$ corresponds to the target word at timestep t. The decoder of the Struct2Seq model outputs a vector of activations for each timestep $\mathbf{y}^{(t)} \in \mathbb{R}^V$ (where V is the size of vocabulary $\mathscr{V}$) which are raw, unnormalised scores for each word in the vocabulary, i.e. $[\mathbf{y}^{(t)}]_i$ (the ith element of the $\mathbf{y}^{(t)}$ vector) is the decoder's score for the ith word from the vocabulary happening at timestep t of the target sequence. Given that the target sequence is taken as a *ground truth*, the probability of a target word $w^{(t)}$ at timestep t is encoded as $p(w^{(t)}) = 1$, where all other words in the vocabulary receive a probability of zero, i.e. $p(\hat{w}^{(t)}) = 0, \forall \hat{w} \in \mathscr{V}, \hat{w}^{(t)} \neq w^{(t)}$. The cross-entropy loss, from the cross-entropy equation for discrete distributions p and q, i.e. $H(p,q) = -\sum_{w \in \mathscr{V}} p(w) \log q(w)$, is then calculated by focusing only on the score for the target word, since that will be the only non-zero term (given that $p(\hat{w}^{(t)}) = 0$ for non-target words). The resulting cross-entropy loss used in these experiments to calculate the loss at timestep t is:[5]

$$\mathscr{L}_t = -\log\left(\frac{\exp([\mathbf{y}^{(t)}]_i)}{\sum_j \exp([\mathbf{y}^{(t)}]_j)}\right) = -[\mathbf{y}^{(t)}]_i + \log\left(\sum_j \exp([\mathbf{y}^{(t)}]_j)\right) \tag{5.4}$$

where $\mathbf{y}^{(t)}$ is the vector of scores for all words at timestep t, $[\mathbf{y}^{(t)}]_i$ is the score for the target word $w^{(t)}$, i.e. $w^{(t)}$ is the ith word in the vocabulary, and $\sum_j$ is a sum over all the elements of $\mathbf{y}^{(t)}$.

To get a single error value for the full sequence comparison, irrespective of the length of sequences that are being compared, the loss function calculates the **average cross-entropy** of the probability of the predicted word for all timesteps. Using the average cross-entropy helps ensure that every datapoint contributes equally to the gradient, regardless of the length of its corresponding sentence. The resulting loss for a full sequence of length T takes the form:

$$\mathscr{L}_S = \frac{\sum_{t=0}^{T} \mathscr{L}_t}{T} \tag{5.5}$$

5.3.7.4 Error propagation and optimisation

The error of the generated sequence is backpropagated through the decoder first, where it unfolds the computation graph for all timesteps that were generated for the output sequence, following the BPTT procedure. It then backpropagates through the encoder in a similar fashion, but given that this is done through the structure of a tree, instead of through ordered timesteps, this unrolling of the Tree-LSTM is

[5]From the PyTorch implementation of cross-entropy loss described in `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

termed **back-propagation through structure (BPTS)** (Goller and Küchler, 1996). The full set of parameters that are tracked when backpropagating through the whole Struct2Seq architecture consists of the following:

$$\begin{aligned}
\Theta_{\text{Encoder}} &= \{\mathbf{M}_{\text{Enc}}, \mathbf{W}_i, \mathbf{U}_i, \mathbf{b}_i, \mathbf{W}_f, \mathbf{U}_f, \mathbf{b}_f, \mathbf{W}_o, \mathbf{U}_o, \mathbf{b}_o, \mathbf{W}_u, \mathbf{U}_u, \mathbf{b}_u\} \\
\Theta_{\text{Decoder}} &= \{\mathbf{M}_{\text{Dec}}, \mathbf{W}_{ih}, \mathbf{b}_{ih}, \mathbf{W}_{hh}, \mathbf{b}_{hh}, \mathbf{W}_{fc}, \mathbf{b}_{fc}\}
\end{aligned}$$

where $\mathbf{M}_{\text{Enc}}$ and $\mathbf{M}_{\text{Dec}}$ are embedding matrices that encode the words that are input into the cells of the encoder and decoder units, respectively. These embedding matrices are represented with green trapezoids in figure 5.11. Aside from the word embedding matrices, $\mathbf{M}_{\text{Enc}}$ and $\mathbf{M}_{\text{Enc}}$, the encoder parameters Θ_{Encoder} correspond to the Tree-LSTM cell parameters described in equation 5.3, and the decoder parameters Θ_{Decoder} correspond to the LSTM cell parameters described in equation 5.2. These experiments use the same vocabulary as in all previous experiments, which has a size of $V = 45,769$ tokens. Different values for the word embedding dimension $M \in \{300, 768\}$, and hidden dimensions $H \in \{512, 768, 1024, 1300\}$, are explored, giving the parameters the following dimensions:

$$\begin{aligned}
\mathbf{M}_{\text{Enc}} &\in \mathbb{R}^{45,769 \times M} \\
\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_u &\in \mathbb{R}^{M \times H} \\
\mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_o, \mathbf{U}_u &\in \mathbb{R}^{H \times H} \\
\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_u &\in \mathbb{R}^{H} \\
\mathbf{M}_{\text{Dec}} &\in \mathbb{R}^{45,769 \times H} \\
\mathbf{W}_{ih}, \mathbf{W}_{hh} &\in \mathbb{R}^{H \times H} \\
\mathbf{b}_{ih}, \mathbf{b}_{hh} &\in \mathbb{R}^{H} \\
\mathbf{W}_{fc} &\in \mathbb{R}^{H \times 45,769} \\
\mathbf{b}_{fc} &\in \mathbb{R}^{45,769}
\end{aligned}$$

Once the error is backpropagated to calculate the contribution of the parameters to the final error, different optimisers (Adam and SGD) with varying learning rates of $\eta \in \{0.0001, 0.0002, 0.001, 0.01, 0.04, 0.07, 0.1\}$ are used to update the model's parameters. The Adam optimiser is used with the default recommended values for the exponential decay rates for moment estimates $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

5.3.7.5 Validation phase

The data processing step is repeated on the validation set to produce a single **validation score**. This helps keep track of the state of the model after training for an epoch, which helps avoid any overfitting, measures the model's generalisation capabilities, and provides an insight into the convergence of the optimisation process.

5.3.8 Post-training

As in the experiments in chapter 4, after model training is concluded, the resulting word embeddings of the Struct2Seq encoder, $\mathbf{M}_{\text{Enc}}$, are evaluated on a set of metrics to probe for semantic and syntactic information in the learned embedding space. As with all previous experiments, the models in this chapter are evaluated on the evaluation metrics described in section 3.4.4, namely the **WMD kNN document classification** task for the extrinsic evaluation, and the **similarity-distance correlation** scores and **word pair distance distributions** as intrinsic evaluation metrics. All evaluation conditions from chapters 3 and 4 remain constant for these experiments. The Struct2Seq word embeddings are then compared with the BERT and RoBERTa word embeddings, as well as the pre-trained Skip-gram model, described in section 3.4.3, to provide a comparison between structure-aware word embeddings and word co-occurrence embedding models.

As described in section 5.2.1, the BERT and RoBERTa models use subword tokenisers to provide a wider coverage of text data, which implies that rare words can be tokenised into several more frequent subwords. To compare the different word embeddings, every word used in the evaluation must have a single vector representation. For this, I use two encoding strategies for BERT word embeddings: **single-token**, where I only evaluate words that have a single corresponding token

(25,766 of the 45,769 words in the vocabulary) and all missing words are encoded as a zero vector; and **average token**, which takes the embeddings of all subword tokens and averages them to produce a single embedding, e.g. $\mathbf{u}_{\texttt{zygomatic}} = (\mathbf{u}_{\texttt{z}} + \mathbf{u}_{\texttt{\#\#y}} + \mathbf{u}_{\texttt{\#\#go}} + \mathbf{u}_{\texttt{\#\#matic}})/4$. The subword vocabulary used by RoBERTa contains only 4,115 of the 45,769 words in the vocabulary as single tokens, so the RoBERTa word embeddings are only encoded as average token embeddings, to avoid having a majority of zero vectors.

5.4 Results

Before running a full training regime, I set out to verify that the Struct2Seq model I designed and implemented was in fact capable of learning from the data. To verify this, I forced the Struct2Seq model to overfit by training it over 200 epochs on a small subset of 0.1% of the data, which amounted to a training set of 450 sentences, and a validation set of 56 sentences. As shown in figure 5.12, the model was able to fit the training data perfectly after 40 epochs when using an Adam optimiser with a learning rate of $\eta = 0.0001$, which confirmed that there were no major issues with the model's architecture or training regime.

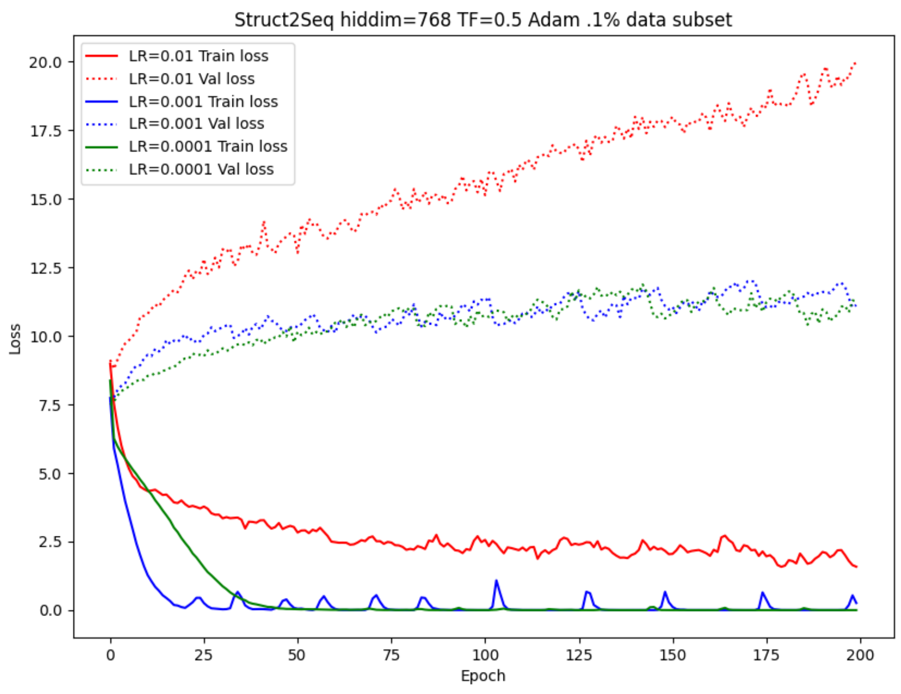


Figure 5.12: Training and validation losses for a Struct2Seq model with 768 hidden dimensions, teacher forcing of 0.5, and an Adam optimiser with different learning rates ($\eta = 0.01$, $\eta = 0.001$, and $\eta = 0.0001$) on .1% of the full dataset over 200 epochs

After validating the overfitting behaviour of the model, I ran additional tests on a larger subset of 10% of the data, i.e. 45,031 training and 5,628 validation sentences, to find a reasonable starting learning rate for training on the full dataset. As shown in figure 5.13, the model using a learning rate of $\eta = 0.0001$ exhibited a more convergent behaviour and obtained lower training and validation losses. This optimisation method and learning rate are consistent with the values used to train BERT and RoBERTa, which use an Adam optimiser with a peak learning rate value of $\eta = 0.0001$.

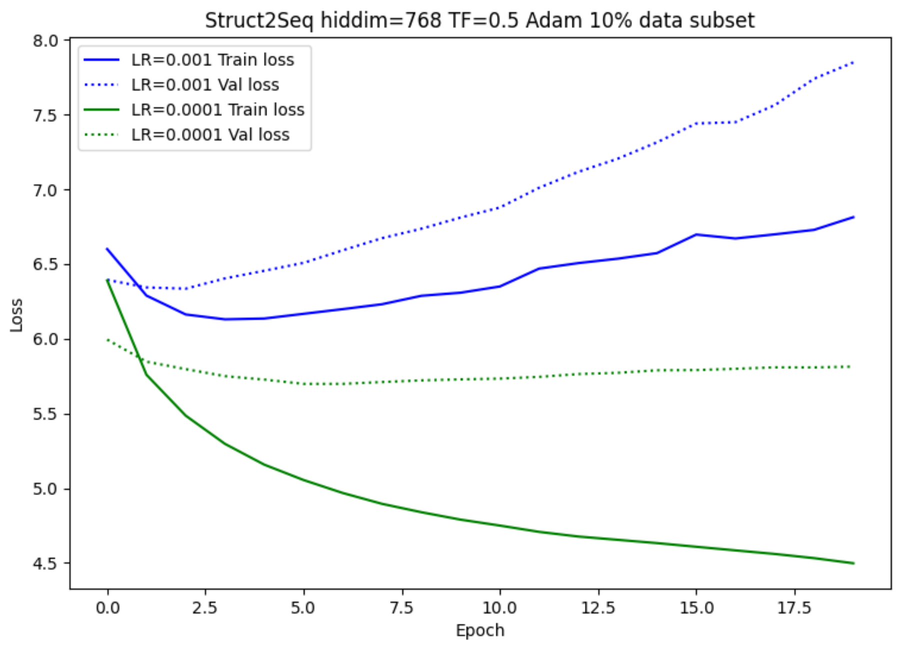


Figure 5.13: Training and validation losses for a Struct2Seq model with 768 hidden dimensions, teacher forcing of 0.5, and an Adam optimiser with different learning rates ($\eta = 0.001$, and $\eta = 0.0001$) on 10% of the full dataset over 20 epochs

When scaling the training to the full dataset, however, the initial optimiser and learning rate selection performed poorly. As described by Wilson et al. (2017), adaptive gradient optimisation methods like the Adam optimiser can oftentimes lead to worse generalisation capabilities than non-adaptive methods. For this reason, I subsequently experimented with a simpler SGD optimiser. As can be seen in figure 5.14, all models trained with an SGD optimiser outperformed the best model trained with an Adam optimiser.

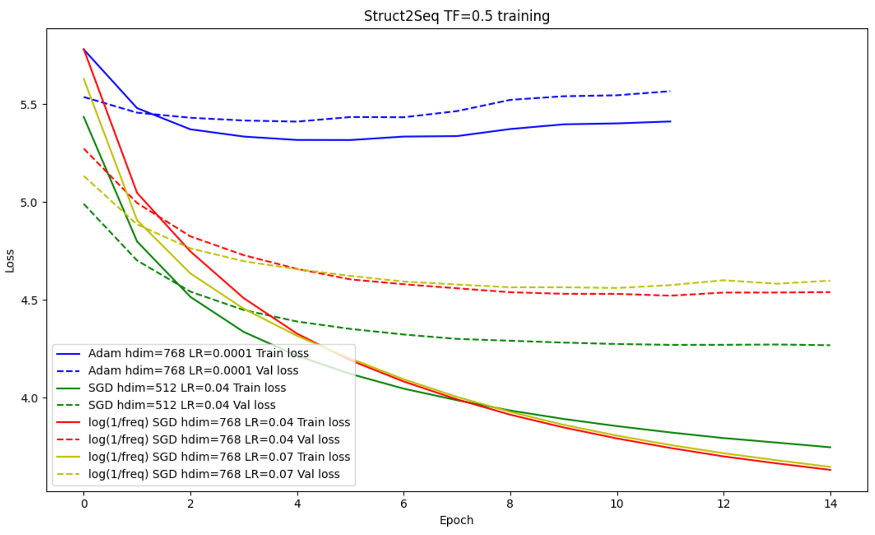


Figure 5.14: Training and validation losses for Struct2Seq models with different hidden dimensions, optimisers, learning rates, and word frequency weightings

An initial observation of the top predictions made by the model showed that the model developed a bias towards the most frequent words in the vocabulary, as shown in the sample validation predictions for each epoch shown in table 5.1. To compensate this bias, I explored adding a weighting of $\log\left(\frac{1}{\text{freq}}\right)$ to the words predicted by the model. This approach assigns a weight of 2.78 to the most frequent word, and 16.09 to the least frequent words, meaning that the contribution of rare words to the loss can be up to 5.75 times greater than that of frequent words. This weighting is meant to favour predicting rare words over frequent ones, which it qualitatively seems to do, as exemplified by the sample predictions presented in table 5.2. The resulting weighted cross-entropy loss is:

$$\mathscr{L}_t = -\phi_i \log\left(\frac{\exp([\mathbf{y}^{(t)}]_i)}{\sum_j \exp([\mathbf{y}^{(t)}]_j)}\right) = -\phi_i[\mathbf{y}^{(t)}]_i + \phi_i \log\left(\sum_j \exp([\mathbf{y}^{(t)}]_j)\right) \tag{5.6}$$

where $\phi_i = \log\left(\frac{1}{\text{freq}_i}\right)$ is the weight for word i.

I evaluate both the weighted and unweighted versions of the Struct2Seq model, namely *Struct2Seq* $H = 512$ $\eta = 0.04$ (hidden layer dimension of $H = 512$) and

Table 5.1: Sample validation predictions at epochs 1, 8, and 15 for Struct2Seq teacher forcing=0.5 hidden dimension=768 SGD optimiser with a learning rate of $\eta = 0.04$

Target	Prediction at epoch		
	1	8	15
if	if	whether	if
you	you	they	you
are	ask	are	ask
turned	the	not	turned
down	the	the	to
ask	the	the	the
the	the	the	the
bank	the	<eos>	the
whether	or	<eos>	or
or	the	the	they
not	the	the	ask
they	the	<eos>	to
have	are	<eos>	ask
used	a	used	the
a	a	to	or
credit	the	the	you
reference	<eos>	<eos>	for
agency	<eos>	<eos>	the
<eos>	<eos>	the	<eos>

Struct2Seq $H = 768$ $\eta = 0.04$ $\log\left(\frac{1}{\mathit{freq}}\right)$ (hidden layer dimension of $H = 768$). Both models were trained under the same conditions: 15 epochs of training, SGD optimiser with a learning rate of $\eta = 0.04$, teacher forcing ratio of `tf=0.5`, and word embedding dimension of 300.

WMD Document Classification results for structure-aware word embeddings are presented in table 5.3, where Word2Vec word embeddings are included for comparison with a non-sequential word co-occurrence model. Both Struct2Seq models perform similarly, indicating that the word frequency weighting strategy does not have a substantial impact on the learned word embeddings. The Struct2Seq models achieve significantly lower accuracies than the Word2Vec, BERT, and RoBERTa models. Word2Vec is outperformed by both BERT and RoBERTa models. The best performing model by a significant margin is BERT, both evaluated with single tokens and average tokens.

Table 5.2: Sample validation predictions at epochs 1, 8, and 15 for Struct2Seq teacher forcing=0.5 hidden dimension=768 SGD optimiser with a learning rate of $\eta = 0.04$ and word frequency weighting of $\log\left(\frac{1}{\text{freq}}\right)$

Target	Prediction at epoch		
	1	8	15
if	if	if	if
you	you	you	you
are	ask	ask	ask
turned	ask	ask	whether
down	or	ask	the
ask	the	the	the
the	the	the	whether
bank	the	bank	whether
whether	<eos>	<eos>	you
or	<eos>	<eos>	or
not	<eos>	not	not
they	<eos>	<eos>	you
have	<eos>	<eos>	have
used	not	<eos>	turned
a	<eos>	<eos>	to
credit	<eos>	<eos>	problem
reference	<eos>	<eos>	or
agency	<eos>	<eos>	<eos>
<eos>	<eos>	<eos>	<eos>

Table 5.3: WMD kNN document classification accuracies on the 20 Newsgroups dataset for *Word2Vec*, *Struct2Seq* (using SGD and a learning rate of $\eta = 0.04$), *BERT*, and *RoBERTa* word embeddings

Model	Accuracy
Struct2Seq $H = 512$	37.37% ($\pm$ 1.09)
Struct2Seq $H = 768$ $\log\left(\frac{1}{\text{freq}}\right)$	36.95% ($\pm$ 1.09)
Word2Vec	**62.75% ($\pm$ 0.89)**
BERT single token	**65.18% ($\pm$ 1.08)**
BERT average token	**65.79% ($\pm$ 1.08)**
RoBERTa average token	60.33% ($\pm$ 1.10)

Table 5.4: Similarity-distance correlation results with **cosine** distance for *Struct2Seq* (using SGD and a learning rate of $\eta = 0.04$), *BERT*, *RoBERTa*, and *Word2Vec* word embeddings

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Struct2Seq $H = 512$	-0.0004	0.0719	0.0162	-0.0099
Struct2Seq $H = 768$ $\log\left(\frac{1}{\text{freq}}\right)$	0.0601	-0.0336	-0.0769	0.0090
Word2Vec	-0.5807	-0.4200	-0.3241	-0.2490
BERT single token	-0.5513	-0.3864	-0.3617	-0.2099
BERT average token	-0.5789	-0.3938	-0.3621	-0.2367
RoBERTa average token	-0.1649	-0.0942	-0.2187	-0.0854

Table 5.5: Similarity-distance correlation results with **Euclidean** distance for *Struct2Seq* (using SGD and a learning rate of $\eta = 0.04$), *BERT*, *RoBERTa*, and *Word2Vec* word embeddings

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Struct2Seq $H = 512$	0.0872	0.0724	-0.0024	-0.0173
Struct2Seq $H = 768$ $\log\left(\frac{1}{\text{freq}}\right)$	0.0898	-0.0435	-0.0247	0.0130
Word2Vec	-0.7706	-0.6150	-0.4426	-0.3565
BERT single token	-0.6556	-0.4113	-0.4927	-0.2958
BERT average token	-0.6440	-0.4118	-0.4810	-0.2773
RoBERTa average token	-0.5231	-0.3389	-0.1801	-0.1041

Similarity-Distance Correlation results, presented in tables 5.4 and 5.5, show Word2Vec achieves the strongest correlation scores across most datasets, with the exception of the SimLex-999 dataset, where BERT word embeddings retain an advantage. Both Struct2Seq models achieve significantly weaker correlation scores than the rest of the models, and the frequency weighting strategy does not seem to substantially impact these scores. RoBERTa achieves considerably weaker correlation scores than the BERT word embeddings. These patterns are consistent for cosine and Euclidean distances, although the difference in correlation scores between RoBERTa and BERT and Word2Vec appears to be more pronounced with cosine distances.

Word Pair Distance Distributions for random, contextual, and synonym word pairs are shown in figures 5.15 and 5.16 for the *Word2Vec*, *Struct2Seq* $\eta = 0.04$ $H = 512$, *BERT single-token*, *BERT average-token*, and *RoBERTa average-token* word embeddings. The distance distributions for the *Struct2Seq* $\eta = 0.04$ $H = 512$ word embeddings are considerably larger than in all other models, which is especially evident in the Euclidean distance distributions (figure 5.16). *Struct2Seq* $\eta = 0.04$ $H = 512$ word embeddings also exhibit very similar distributions across all word pair groups, i.e. random, contextual, and synonym word pairs.

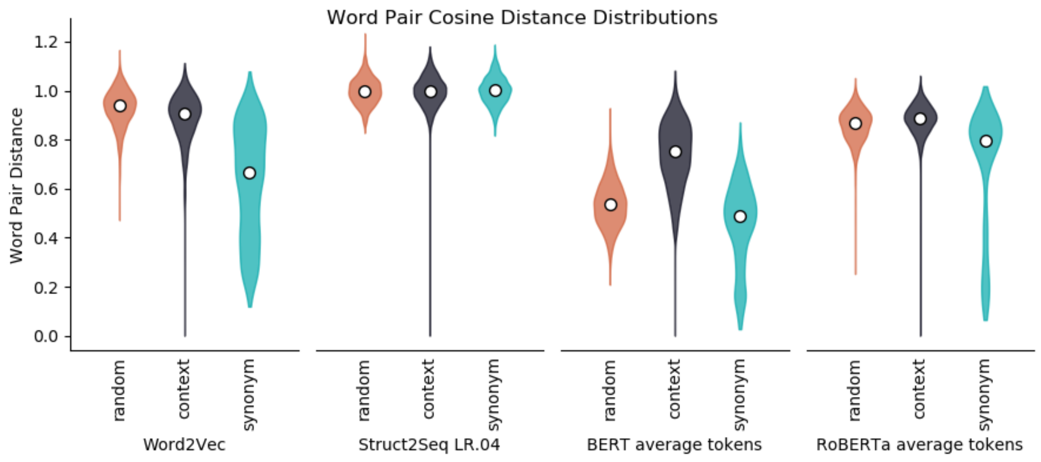


Figure 5.15: Cosine distance distributions between random, contextual, and synonymous word pair sets for *Word2Vec*, *Struct2Seq* $\eta = 0.04$ $H = 512$, *BERT single-token*, *BERT average-token*, and *RoBERTa average-token* word embeddings

As observed in the zoomed-in distance distributions for the *Struct2Seq* $\eta = 0.04$ $H = 512$ and *Struct2Seq* $\eta = 0.04$ $H = 768$ $\log\left(\frac{1}{freq}\right)$ word embeddings in figures 5.17 and 5.18, the distributions across all word pair groups, and even between the two models, bear a very close resemblance and exhibit near identical central tendencies (the white dots in the violin plots represent the mean of the distribution). This is especially noteworthy when compared to the variations in distribution shapes and central tendencies for the rest of the models observed in figures 5.15 and 5.16.

Lastly, figures 5.19 and 5.20 present only the pre-trained *Word2Vec*, *BERT average-token*, and *RoBERTa average-token* word embeddings to minimise the visual artifacts caused by the large distance distributions in Struct2Seq word embeddings. These plots show that the structure-aware word embeddings produced by

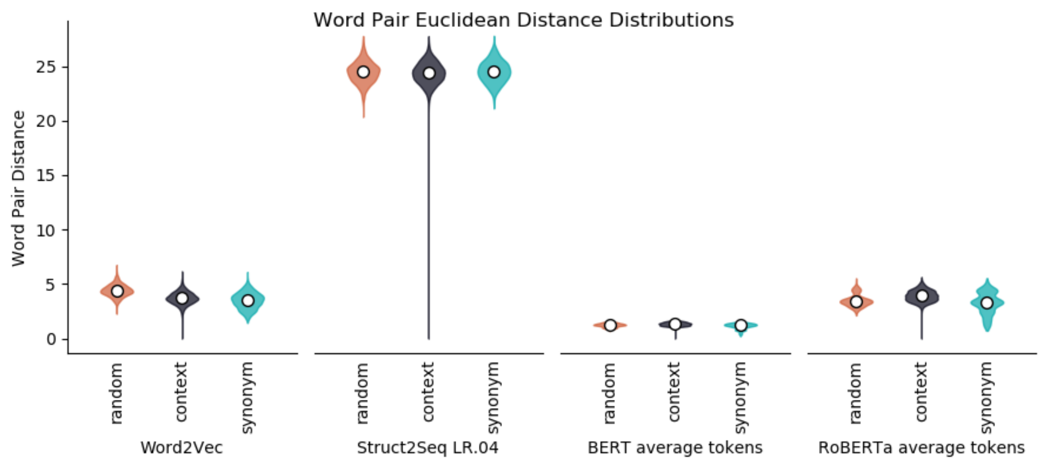


Figure 5.16: Euclidean distance distributions between random, contextual, and synonymous word pair sets for *Struct2Seq* $\eta = 0.04$ $H = 512$, *BERT single-token*, *BERT average-token*, and *RoBERTa average-token* word embeddings

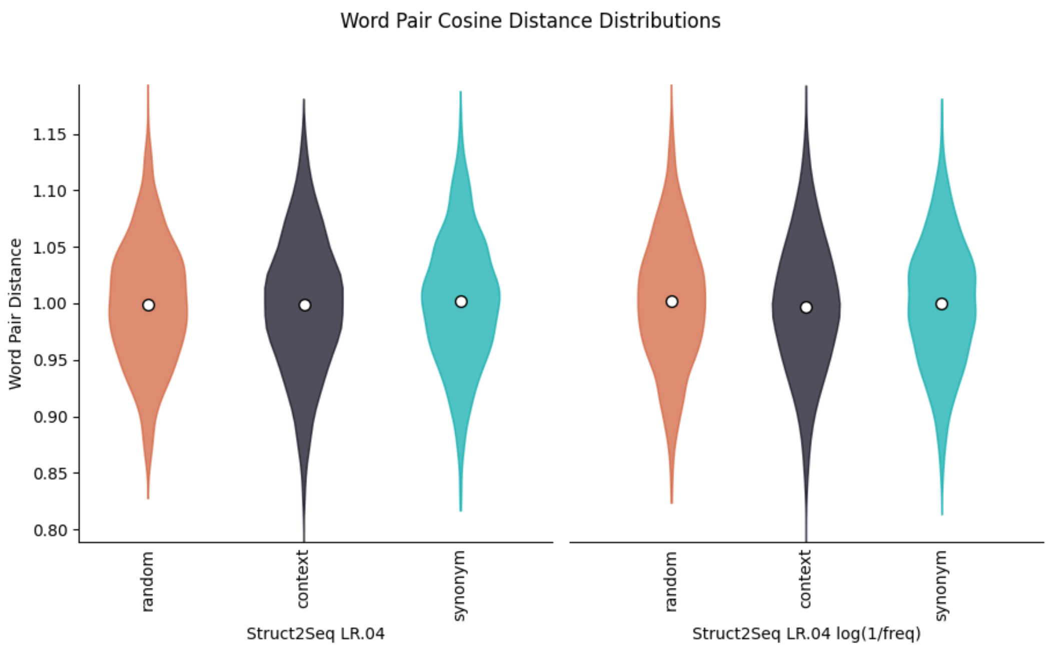


Figure 5.17: Zoomed-in cosine distance distributions between random, contextual, and synonymous word pair sets for *Struct2Seq* $\eta = 0.04$ $H = 512$ and *Struct2Seq* $\eta = 0.04$ $H = 768$ $\log\left(\frac{1}{freq}\right)$ word embeddings

BERT and RoBERTa have the unexpected pattern that contextual word pairs have larger distance distributions than random word pairs. This pattern is especially clear in the cosine distance distributions of BERT word embeddings. Despite this coun-

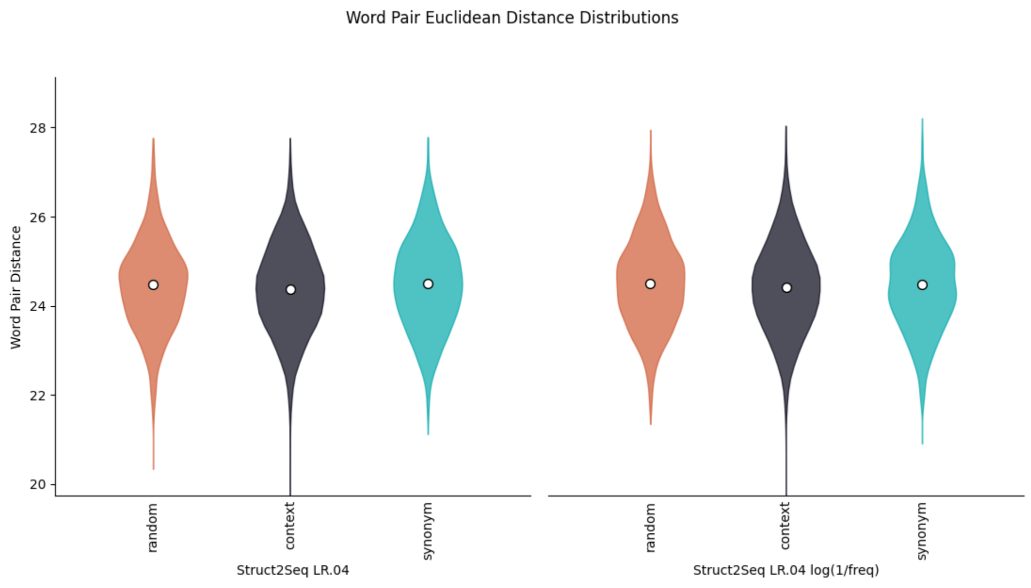


Figure 5.18: Zoomed-in Euclidean distance distributions between random, contextual, and synonymous word pair sets for *Struct2Seq* $\eta = 0.04$ $H = 512$ and *Struct2Seq* $\eta = 0.04$ $H = 768$ $\log\left(\frac{1}{freq}\right)$ word embeddings

terintuitive behaviour, BERT obtained the highest accuracies in the WMD extrinsic evaluation, which is a task based entirely on Euclidean distances and where, intuitively, contextual information would contribute significantly to document classification.

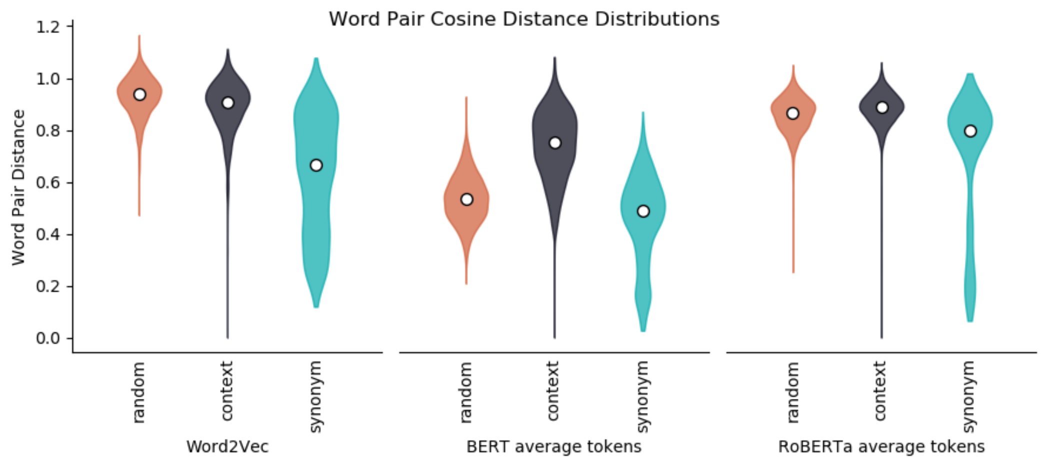


Figure 5.19: Cosine distance distributions between random, contextual, and synonymous word pair sets for *Word2Vec*, *BERT average-token*, and *RoBERTa average-token* word embeddings

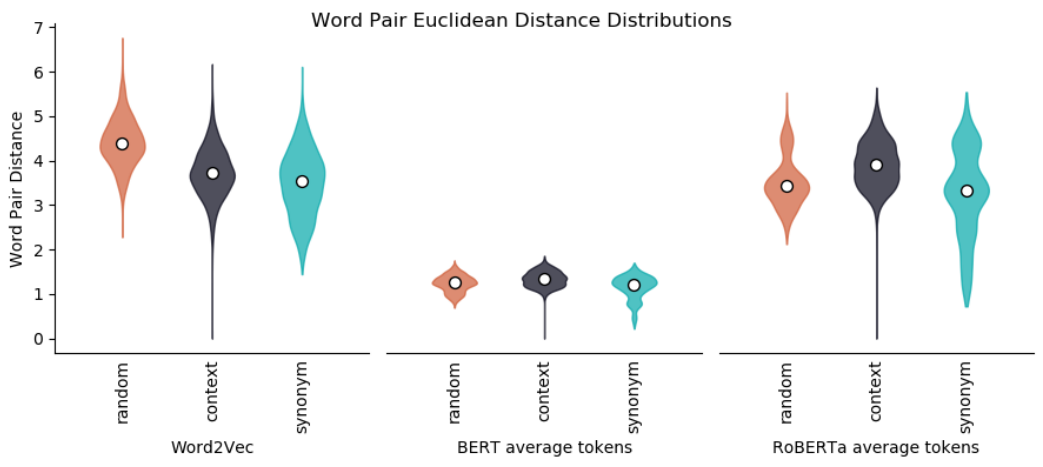


Figure 5.20: Euclidean distance distributions between random, contextual, and synonymous word pair sets for *Word2Vec*, *BERT average-token*, and *RoBERTa average-token* word embeddings

Analysis

The word embedding evaluations in this chapter were performed on sentence representation models which do not explicitly train their word embedding units, which opens the possibility that explicitly trained structure-aware word embeddings could potentially achieve even better performance. Surprisingly, Word2Vec obtains lower scores than BERT and RoBERTa in the WMD task, despite the fact that these are word-level tasks and Word2Vec is the only model that explicitly trains word embeddings.

There appears to be a relation between the pattern of word pair distances and the similarity-distance correlation results. Word2Vec, which outperformed structure-aware word embeddings on the WordSim353 and SimVerb-3500 datasets, exhibits word pair distance distributions that conform to my intuition that both semantically similar and contextually related words should be embedded close together. However, BERT's stronger correlation results on the SimLex-999 dataset, together with its extrinsic evaluation scores, challenge the assumption that the embeddings of contextually related words should be placed close together. Inspecting two examples from the SimLex-999 dataset with low similarity scores, `tiny` and `huge` obtain a similarity score of 0.6/10, and `dog` and `cat` obtain a similarity score of 1.75/10, these two pairs of words can be interpreted as being contextually related since they might appear in the same sentence more frequently than entirely unrelated words. These examples offer a possible explanation for why the larger contextual distances in BERT result in stronger correlation scores in the SimLex-999 dataset, which focuses exclusively on semantic similarity. The larger distances for contextual word pairs observed in BERT and RoBERTa can also be an effect of their architecture: multi-layer attention models have no need to explicitly capture contextual information in their word embedding layer, since this information will be captured by the attention layers in their architecture. Context is captured by the model's structure, so its word embeddings are free to focus on semantic similarity.

Even though the Struct2Seq exhibits a convergent training behaviour and good generalisation performance on its learning task, it would appear from the word pair distance distributions that the Struct2Seq word embeddings are not distinguishing random words from contextually or synonymically related words. This is further confirmed by the weak correlation scores achieved by these models on the similarity-distance correlation task. One possible explanation for this is that the learning task is inadequate for the learning of word embeddings, and these embeddings could potentially benefit from incorporating an auxiliary loss term that enforces semantically meaningful geometric properties on the word embeddings as part of the training objective. Another possible explanation for the poor performance of Struct2Seq word embeddings in these evaluations is that the current training regime is insufficient for the model to produce meaningful representations. One of the main differences between the pre-trained Word2Vec, BERT, and RoBERTa models and the Struct2Seq models presented here lies in the magnitude of training. While Struct2Seq models were trained on fewer than 10 million words, the Word2Vec models were trained on 6 billion words, and the BERT and RoBERTa models are trained on approximately 3.3 billion words.[6]

The patterns observed when training the Struct2Seq model with larger hidden dimensions of 1,024 and 1,300, as well as with a larger word embedding dimension

[6]Devlin et al. (2018) report training BERT on a dataset made up of the *BooksCorpus* (800 million words) and a Wikipedia dump (2.5 billion words). Liu et al. (2019), on the other hand, report the dataset used to train RoBERTa as consisting of 160GB of text, but make no mention of the number of words this refers to.

of 768, showed no clear performance gains, hinting that the poor performance cannot be easily improved by a simple increase in model size. Following the current trend in representation learning models for NLP, the potential of the Struct2Seq architecture might only be fully understood by scaling the complexity of the model and the size of the training data by several orders of magnitude.[7] However, as evidenced by the decay in performance exhibited by RoBERTa, despite having the same architecture and retaining most of the same training conditions as BERT, certain training conditions can have a large bearing on the quality of resulting model even when keeping the same architectural and dataset choices. For instance, the fact that RoBERTa achieves a lower document classification accuracy than BERT, can potentially be explained by the removal of the inter-sentence (next sentence prediction) task from its training regime, although as argued by Liu et al. (2019), this task has no effect on the scores obtained by the RoBERTa model on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018a).

[7]The models in this thesis are trained on 10^7 words, or an order of magnitude of 7, while dataset sizes for commercial NLP models are commonly over 10^9 words, or an order of magnitude of 9.

Chapter 6

Conclusions

In this thesis I explored the effect of integrating linguistic knowledge into representation learning approaches aimed at learning word embeddings. The main goal of these approaches is to contribute towards the creation of domain-invariant representations of text that can be accessible even for low-resource languages, i.e. representation learning models that can be robustly trained at reasonable computational and data costs. The expectation was that, by exploiting existing sources of linguistic knowledge, distributional representation learning models would be able to produce word embeddings that reflect more universal linguistic patterns and can therefore be used more effectively across linguistic domains and NLP applications.

This thesis provides a comprehensive analysis of multiple word representation techniques and approaches. The models evaluated in this research include publicly available pre-trained word embeddings, feature-engineered word representations, and my own word embedding models. For this comparison, I designed an automated pipeline to construct **feature-engineered word representations** that leverages various existing lexical resources to construct word representations from semantic, morphological, and lexical features (chapter 3). To provide a fair comparison of the different word representation approaches, I used a single word embedding evaluation framework comprised of a set of extrinsic and intrinsic evaluation metrics. As part of this framework, I proposed a new **word pair distance distribution** intrinsic evaluation (chapter 3) which gives an additional insight into the geometric properties of word representation models.

For the training of my own models, I defined a learning algorithm in order to ensure that as many of the experimental variables remained fixed or with minimal variation. Under these conditions, I trained a set of word embedding models using my proposed a **knowledge-augmentation technique** (chapter 4) which incorporates semantic relations extracted from lexical knowledge bases into the distributional learning of word embeddings through word co-occurrence models. To control the flow of augmented information while training these knowledge-augmented word embeddings, I implemented a novel **embedding partitioning** method (chapter 4). Additionally, I designed and trained a new **Struct2Seq model** (chapter 5) that produces word and sentence embeddings by learning a mapping between a dependency parse tree and its originating sentence.

The empirical results presented in this thesis suggest that training conditions, such as the choice of optimiser, the learning rate hyperparameter, or the vocabulary size, can have a large effect on the quality of a word representation model regardless of its underlying architecture or the characteristics of its training data. As evidenced by the experiments in this thesis, training conditions can have intricate and unpredictable interactions, and fully understanding the effect of varying a single training condition requires further work in this direction through meticulously controlled experiments. The work I presented here can help inform common heuristics in NLP research and applications, such as considerations regarding the choice of vocabulary and comparative advantages and disadvantages of commonly used word representation models.

Limitations and Future work

The models and approaches proposed in this thesis have contributed to the growing field of knowledge-aided representation learning for NLP. The experiments and analysis presented here show that it is possible to incorporate linguistic knowledge into the distributional learning of word embeddings, but the full potential of these approaches remains to be explored. Despite the known advantages that the incorporation of pre-existing linguistic knowledge contributes to the distributional learning of word embeddings (Vashishth et al., 2019; Glavaš and Vulić, 2019; Vulić and Mrkšic, 2018; Faruqui et al., 2015), the experiments conducted in this thesis present no conclusive evidence to further advance that claim. While the models and approaches presented in this research fall short of showing a benefit of this integration when compared against large-scale word embedding models, their applicability to low-resource languages remains to be explored in future work.

Even though keeping all experimental conditions fixed while training and evaluating all word representation models allowed for a clear comparison of these models and their individual strengths and shortcomings, this setup is restrictive when attempting to train more competitive models. The next step forward would therefore be to explore the full potential of a single one of the models presented here by training it on larger datasets while also increasing the model's complexity (e.g. number of layers or dimensions, adding attention layers). Specifically, the Struct2Seq model could obtain more competitive performance by training it on the full version of the BNC data, as well as by incorporating an attention layer that helps the decoder focus on specific nodes of the Tree-LSTM encoder. In addition, training a competitive version of these models would also require searching for the optimal hyperparameters for a given set of experimental conditions, as observed in the experiments in chapter 5 where the optimal learning rate and optimiser changed depending on the size of the training data. The experiments conducted for this research raise questions about the pertinence of restricting experimental conditions when comparing different NLP approaches. Given the complexity of the models and training regimes presented in this work, these restrictions can hamper the optimisation process and produce sub-optimal models as a result. These results suggest that it might be more productive to fully optimise training conditions and hyperparameters for each individual model and explore different alternatives that enable a like-for-like comparison of the fully optimised models.

Regarding possible avenues for future research, the potential of tree structures in NLP research still remains underexplored. Compositional architectures, such as Tree-LSTMs and Graph neural networks (GNNs) still have untapped potential, especially for processing and interpreting the structural information contained in language. This family of models has been explored in the context of Natural Language Generation (NLG) (Koncel-Kedziorski et al., 2019; Schmitt et al., 2020) and can potentially benefit the field of text representation learning.

The concept of partitioning embeddings can also be more thoroughly studied by applying it to existing models that are trained on a joint loss. The potential of partitioned embeddings in the decoupling of structural information from semantic similarity that I attempted in chapter 4 will be better understood after applying it to a wider variety of learning algorithms. The generality of this approach can also be analysed theoretically by running controlled tests that gauge the informational flow to each of the different subspaces in the embedding. Additionally, new evaluation metrics can be proposed that look into the geometric and semantic properties of different subspaces.

On the theoretical side, a more comprehensive analysis on the effect of the choice of vocabulary is still needed in the NLP community. A systematic review on the effect of vocabulary sizes and the advantages and disadvantages of word vs. subword vocabularies, tokenisation methods, and the impact of coverage in a model's performance can be very beneficial to the NLP research community. Furthermore, many empirical results in NLP rely on evaluations that use different distance metrics, while the specific impact of using one distance metric instead of another is still not well-understood in the context of word embeddings, especially given that the models that are being evaluated can be very high-dimensional. ~~An in-depth~~ To the best of my knowledge, a systematic analysis of the mathematical and geometric properties of different distance metrics ~~in high-dimensional spaces~~ with respect to their effects on word embedding learning and evaluation is still missing in the NLP literature. Such an analysis could help NLP researchers make more informed decisions when constructing their model evaluation frameworks.

The experiments in this thesis, as part of a growing body of research in the existing NLP literature, suggest that the integration of knowledge bases and statistical learning can benefit a wide variety of NLP approaches. Of special interest as a future work direction is the applicability of these approaches in low-resource languages. Linguistic knowledge has the potential to reduce the training data requirements for NLP models by providing shortcuts to generalisable linguistic patterns during training. If applied successfully, the incorporation of knowledge into statistical learning approaches could help democratise NLP by reducing the computational and data requirements for training competitive nlp models.

Appendix A

Knowledge-Augmented Word Embeddings - Additional Results

Additional results on knowledge-augmented word embeddings not presented in chapter 4.

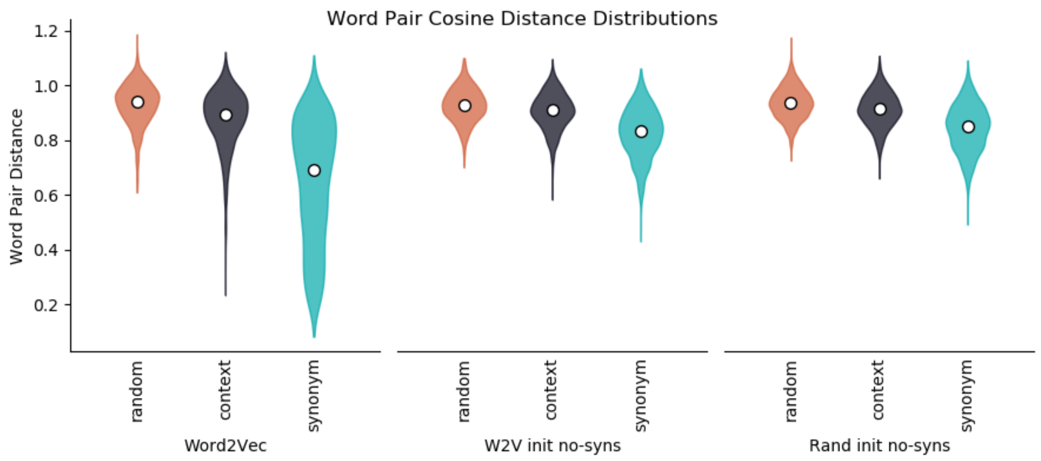


Figure A.1: Cosine distance distributions between random, contextual, and synonymous word pair sets for unaugmented models: *Word2Vec*, *Word2Vec init no-syns*, and *Rand init no-syns*

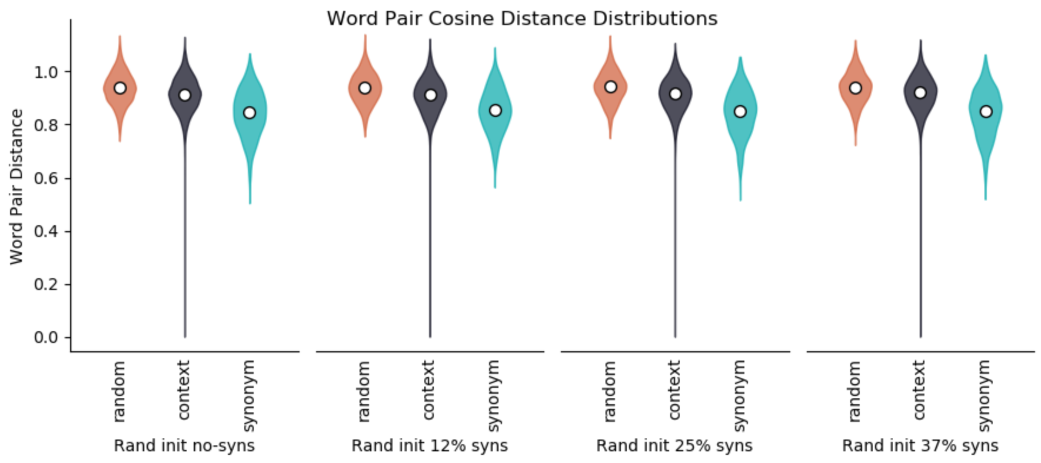


Figure A.2: Cosine distance distributions between random, contextual, and synonymous word pair sets for randomly initialised embeddings with varying augmentation ratios

Table A.1: Similarity-distance correlation results with **cosine** distance for knowledge-augmented partitioned word embeddings

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec init no-syns	-0.1453	-0.0959	-0.0966	-0.0693
Word2Vec init 25% syns 25% part	-0.1173	-0.1845	0.0445	0.0351
Word2Vec init 25% syns 50% part	-0.0267	-0.1160	0.0988	0.0282
Word2Vec init 25% syns 75% part	-0.0645	-0.1332	0.0143	0.0172
Word2Vec init 25% syns	-0.0900	-0.0402	-0.1044	-0.0456
Rand init no-syns	-0.0486	-0.0315	-0.0320	-0.0393
Rand init 25% syns 25% part	0.0203	-0.0842	0.0523	0.0546
Rand init 25% syns 50% part	-0.1054	-0.0451	0.0396	0.0507
Rand init 25% syns 75% part	-0.0017	-0.0758	-0.0180	0.0264
Rand init 25% syns	-0.0875	-0.0360	-0.1191	-0.0704

Table A.2: Similarity-distance correlation results with **cosine** distance for knowledge-augmented partitioned word embeddings (separate partitions)

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.5807	-0.4200	-0.3241	-0.2490
Word2Vec init no-syns	-0.1453	-0.0959	-0.0966	-0.0693
Word2Vec init 25% syns 25% part	-0.1173	-0.1845	0.0445	0.0351
Word2Vec init 25% syns 25% part (augm)	-0.1250	-0.1683	0.0155	0.0317
Word2Vec init 25% syns 25% part (unaugm)	-0.0944	-0.1605	0.0498	0.0358
Word2Vec init 25% syns 50% part	-0.0267	-0.1160	0.0988	0.0282
Word2Vec init 25% syns 50% part (augm)	-0.0643	-0.1340	0.0456	0.0065
Word2Vec init 25% syns 50% part (unaugm)	0.0066	-0.0798	0.1084	0.0361
Word2Vec init 25% syns 75% part	-0.0645	-0.1332	0.0143	0.0172
Word2Vec init 25% syns 75% part (augm)	-0.0084	-0.0798	-0.0109	0.0035
Word2Vec init 25% syns 75% part (unaugm)	-0.1264	-0.1994	0.0515	0.02976
Rand init no-syns	-0.0486	-0.0315	-0.0320	-0.0393
Rand init 25% syns 25% part	0.0203	-0.0842	0.0523	0.0546
Rand init 25% syns 25% part (augm)	-0.0198	0.0103	0.0075	0.0149
Rand init 25% syns 25% part (unaugm)	0.0335	-0.1013	0.0693	0.0655
Rand init 25% syns 50% part	-0.1054	-0.0451	0.0396	0.0507
Rand init 25% syns 50% part (augm)	-0.0318	-0.0077	0.0047	0.0383
Rand init 25% syns 50% part (unaugm)	-0.0980	-0.0219	0.0649	0.0568
Rand init 25% syns 75% part	-0.0017	-0.0758	-0.0180	0.0264
Rand init 25% syns 75% part (augm)	0.0197	-0.0286	-0.0362	0.0038
Rand init 25% syns 75% part (unaugm)	-0.0374	-0.1025	0.0162	0.0513

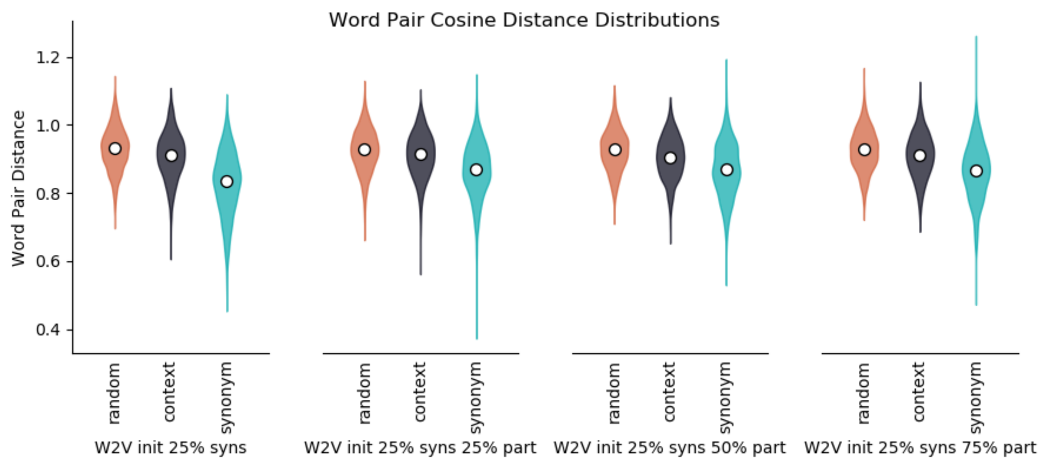


Figure A.3: Cosine distance distributions between random, contextual, and synonymous word pair sets for Word2Vec initialised 25% augmented partitioned embeddings

Table A.3: Similarity-distance correlation results with **Euclidean** distance for knowledge-augmented partitioned word embeddings (separate partitions)

	WordSim353			
	Sim	Rel	SimLex-999	SimVerb-3500
Word2Vec	-0.7706	-0.6150	-0.4426	-0.3565
Word2Vec init no-syns	-0.5144	-0.3207	-0.1957	-0.0952
Word2Vec init 25% syns 25% part	-0.4323	-0.2703	-0.1175	-0.0586
Word2Vec init 25% syns 25% part (augm)	-0.2316	-0.1282	-0.0998	-0.0550
Word2Vec init 25% syns 25% part (unaugm)	-0.3813	-0.2448	-0.0929	-0.0393
Word2Vec init 25% syns 50% part	-0.3496	-0.2562	-0.0595	-0.0630
Word2Vec init 25% syns 50% part (augm)	-0.2888	-0.2085	-0.0434	-0.0810
Word2Vec init 25% syns 50% part (unaugm)	-0.2327	-0.1682	-0.0419	-0.0277
Word2Vec init 25% syns 75% part	-0.4378	-0.2808	-0.1500	-0.0924
Word2Vec init 25% syns 75% part (augm)	-0.3764	-0.2370	-0.1539	-0.0909
Word2Vec init 25% syns 75% part (unaugm)	-0.3056	-0.2390	-0.0471	-0.0499
Rand init no-syns	-0.3694	-0.2347	-0.1206	-0.0817
Rand init 25% syns 25% part	-0.3666	-0.2113	-0.1366	-0.0599
Rand init 25% syns 25% part (augm)	-0.2960	-0.0871	-0.1367	-0.0877
Rand init 25% syns 25% part (unaugm)	-0.2423	-0.1736	-0.0840	-0.0280
Rand init 25% syns 50% part	-0.4139	-0.1919	-0.1019	-0.0438
Rand init 25% syns 50% part (augm)	-0.2803	-0.1086	-0.1036	-0.0507
Rand init 25% syns 50% part (unaugm)	-0.2691	-0.1284	-0.0568	-0.0134
Rand init 25% syns 75% part	-0.3500	-0.2277	-0.1682	-0.0534
Rand init 25% syns 75% part (augm)	-0.2861	-0.1663	-0.1767	-0.0764
Rand init 25% syns 75% part (unaugm)	-0.1816	-0.1638	-0.0661	-0.0054

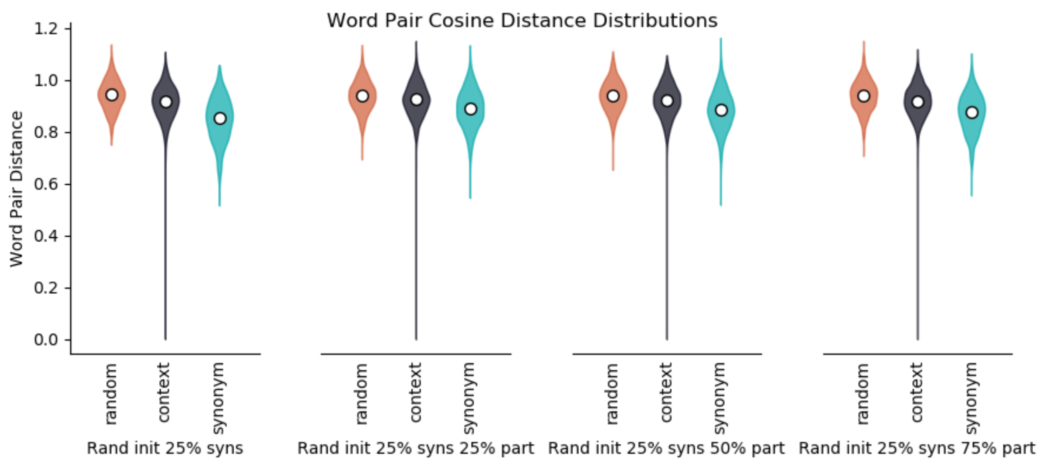


Figure A.4: Cosine distance distributions between random, contextual, and synonymous word pair sets for randomly initialised 25% augmented partitioned embeddings

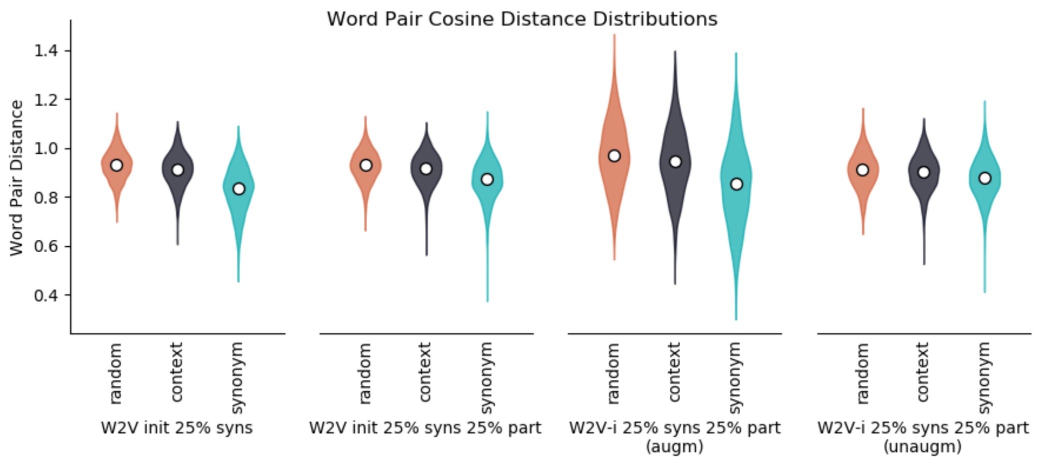


Figure A.5: Cosine distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 25% partitioned embeddings

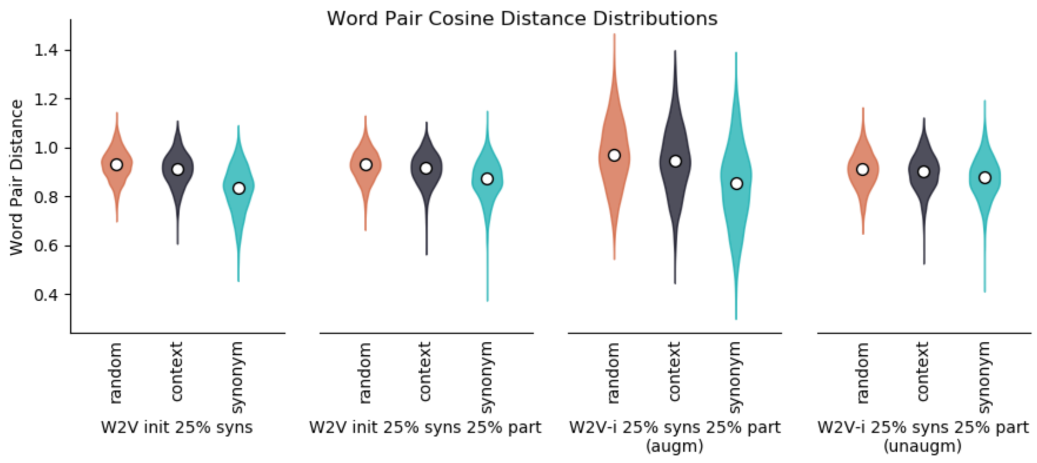


Figure A.6: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 25% partitioned embeddings

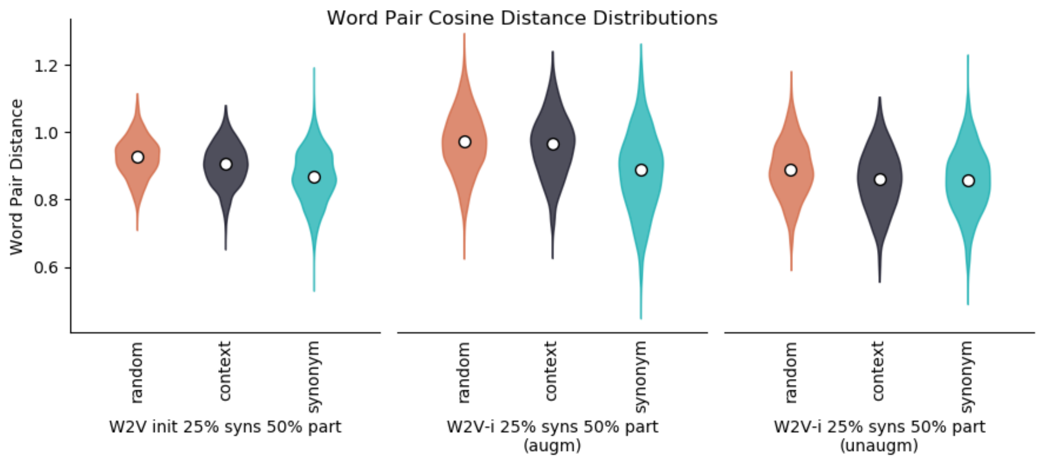


Figure A.7: Cosine distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 25% partitioned embeddings

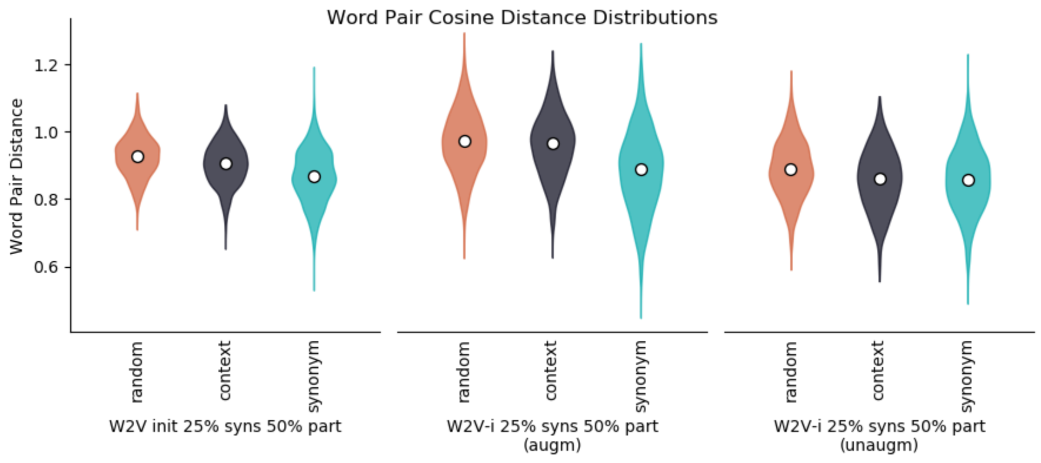


Figure A.8: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 25% partitioned embeddings

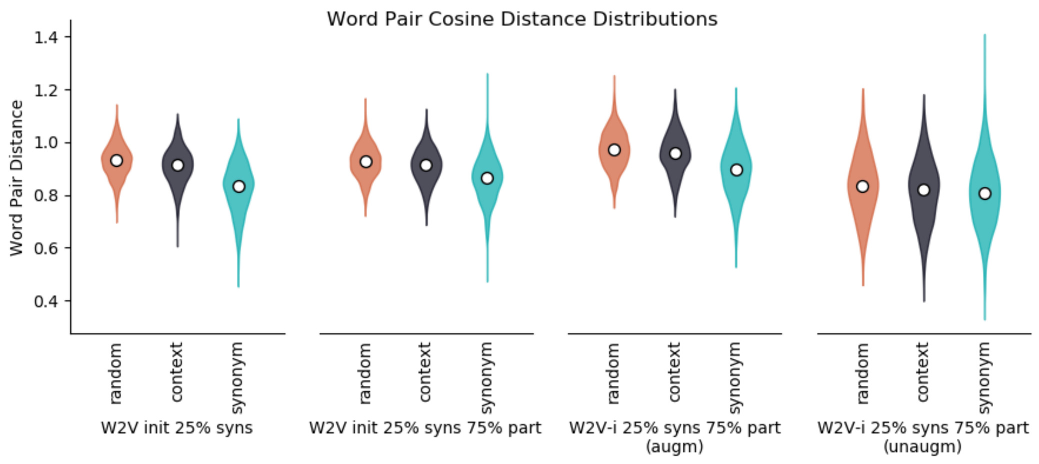


Figure A.9: Cosine distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 25% partitioned embeddings

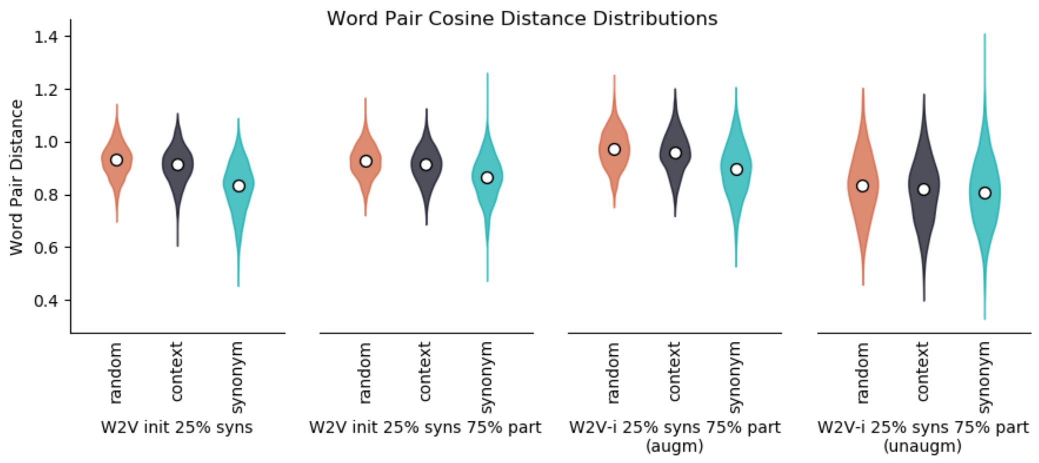


Figure A.10: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in Word2Vec initialised 25% augmented and 25% partitioned embeddings

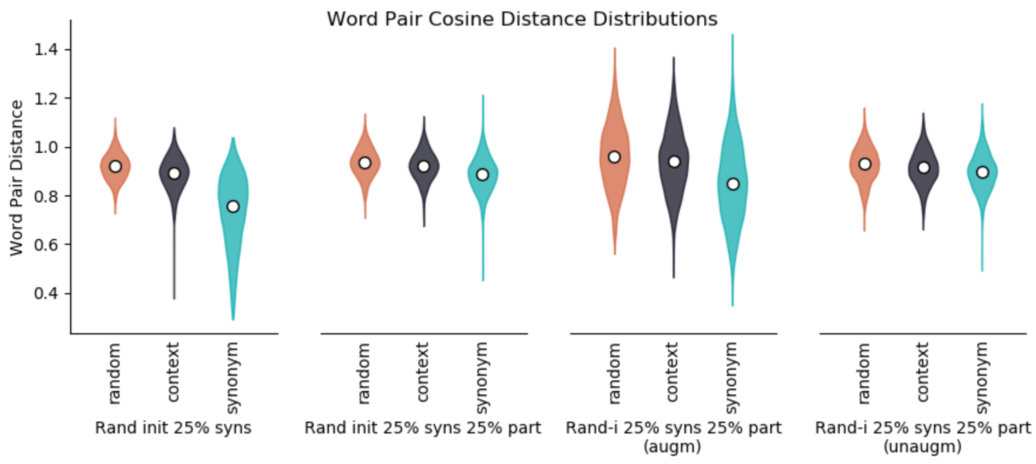


Figure A.11: Cosine distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 25% partitioned embeddings

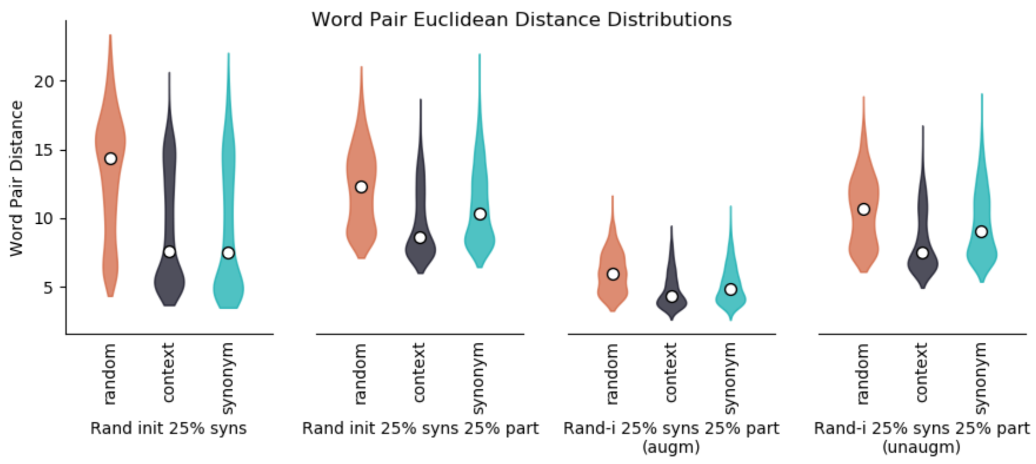


Figure A.12: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 25% partitioned embeddings

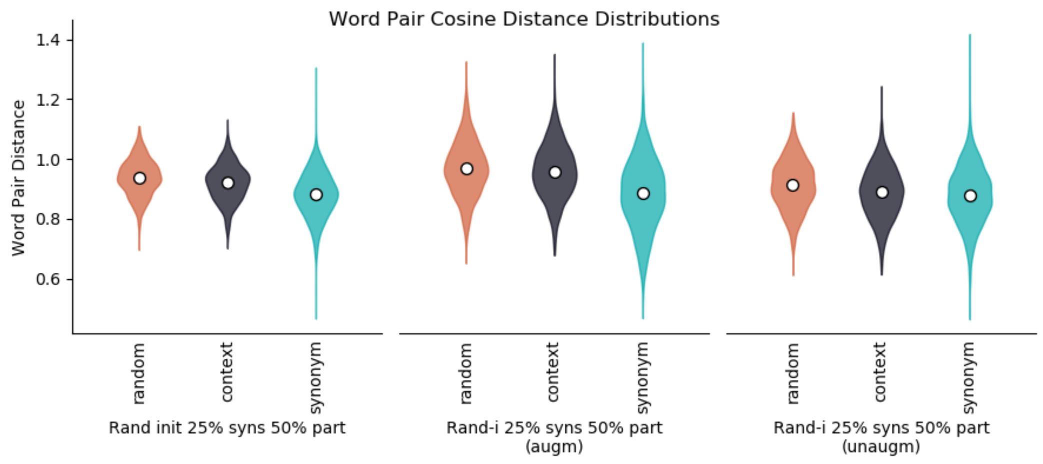


Figure A.13: Cosine distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 25% partitioned embeddings

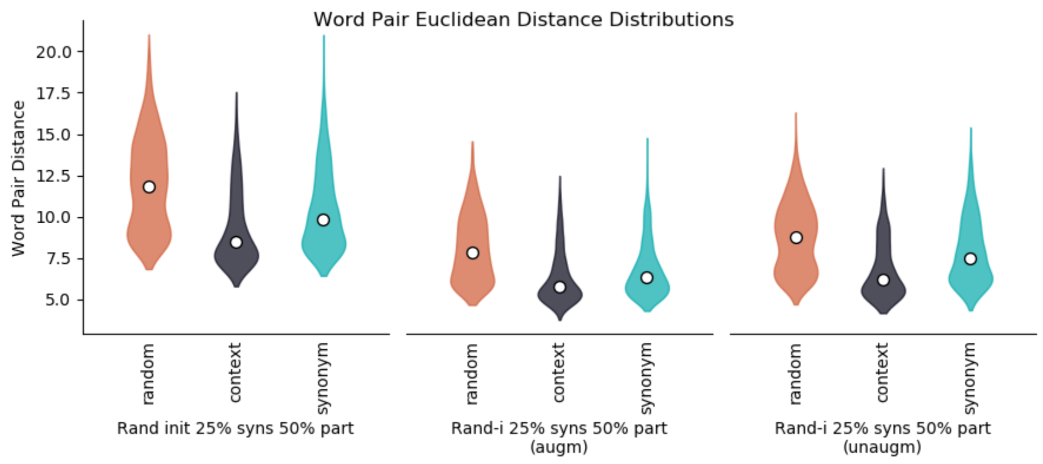


Figure A.14: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 50% partitioned embeddings

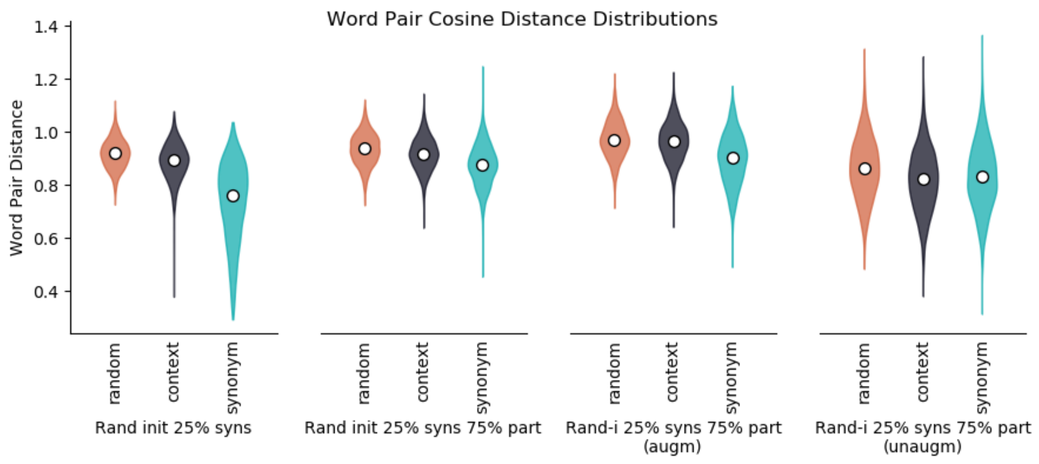


Figure A.15: Cosine distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 75% partitioned embeddings

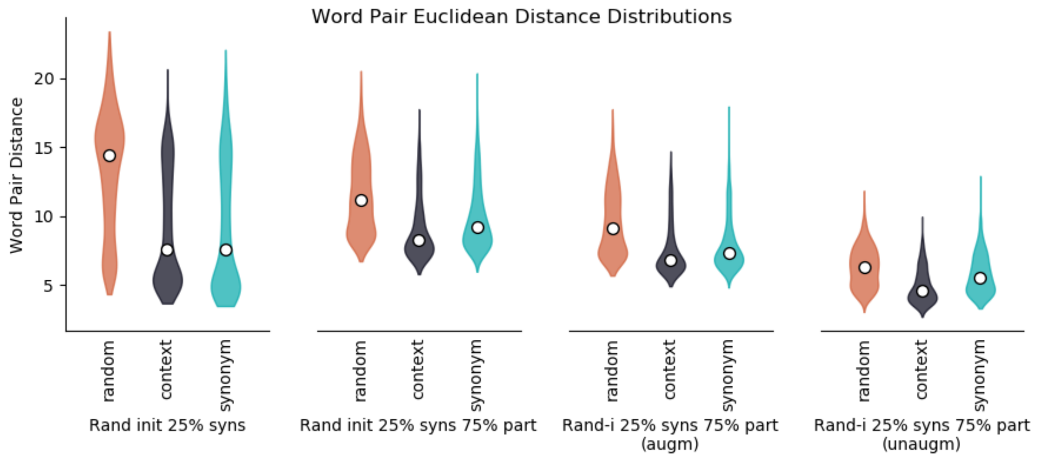


Figure A.16: Euclidean distance distributions between random, contextual, and synonymous word pair sets for partitions in randomly initialised 25% augmented and 75% partitioned embeddings

Bibliography

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pas¸capas¸ca, and Aitor Soroa. A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL*, pages 19–27, Boulder, CO, USA, jun 2009. ACL.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual String Embeddings for Sequence Labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, NM, USA, 2018. ACL.

D J Allerton. The Sentence as a Linguistic Unit. *Lingua*, pages 27–46, 1969.

David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. In *ICLR*. ICLR, 2017.

Mark Aronoff and Kirsten Fudeman. *What is Morphology*. Fundamentals of Linguistics. Wiley, 2nd edition, 2011. ISBN 978-1-405-19467-9.

Ben Athiwaratkun, Andrew Gordon Wilson, and Anima Anandkumar. Probabilistic FastText for Multi-Sense Word Embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pages 1–11, Melbourne, Australia, jul 2018. ACL.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*. ICLR, 2015.

Hareesh Bahuleyan, Lili Mou, Olga Vechtomova, and Pascal Poupart. Variational

Attention for Sequence-to-Sequence Models. In *Proceedings of COLING 2018*. COLING, 2018.

Amir Bakarov. A Survey of Word Embeddings Evaluation Methods. *arXiv*, 2018.

Collin F Baker, Charles J Fillmore, and John B Lowe. The Berkeley FrameNet Project. In *ACL '98/COLING '98: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 86–90, Montreal, Quebec, Canada, 1998. COLING.

Soumya Barikeri, Anne Lauscher, Ivan Vuli´cvuli´c, and Goran Glavaš. REDDITBIAS: A Real-World Resource for Bias Evaluation and Debiasing of Conversational Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, page 1955. ACL, aug 2021.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-Sequence Learning using Gated Graph Neural Networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pages 273–283, Melbourne, Australia, jul 2018. ACL.

Anya Belz. Quantifying Reproducibility in NLP and ML. *arXiv*, sep 2021.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin, Jaz Kandola, Thomas Hofmann, Tomaso Poggio, and John Shawe-Taylor. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Jiang Bian, Bin Gao, and Tie-Yan Liu. Knowledge-Powered Deep Learning for

Word Embedding. *Machine Learning and Knowledge Discovery in Databases*, pages 132–148, 2014.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

BNC Consortium. The British National Corpus, 2007.

Bernd Bohnet, Ryan Mcdonald, Gonçalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez. Morphosyntactic Tagging with a Meta-BiLSTM Model over Context Sensitive Token Encodings. *arXiv*, 2018a.

Bernd Bohnet, Ryan Mcdonald, Gonçalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez. Morphosyntactic Tagging with a Meta-BiLSTM Model over Context Sensitive Token Encodings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pages 2642–2652, Melbourne, Australia, jul 2018b. ACL.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. In *Transactions of the Association for Computational Linguistics*, pages 135–146. ACL, 2017.

Gemma Boleda. Distributional Semantics and Linguistic Theory. *Annual Review of Linguistics*, 6:213–234, 2020. doi: 10.1146/annurev-linguistics-011619-030303.

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Quantifying and Reducing Stereotypes in Word Embeddings. In *2016 ICML Workshop on #Data4Good: Machine Learning in Social Good Applications*, pages 41–45, New York, NY, USA, 2016a. ICML.

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, pages 4349–4357, Barcelona, Spain, 2016b. NeurIPS.

Léon Bottou. Online Learning and Stochastic Approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Siddhartha Brahma. Improved Sentence Modeling using Suffix Bidirectional LSTM. *arXiv*, 2018.

Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334): 183–186, apr 2017. ISSN 10959203. doi: 10.1126/SCIENCE.AAL4230.

Jose Camacho-Collados and Mohammad Taher Pilehvar. From Word to Sense Embeddings: A Survey on Vector Representations of Meaning. *Journal of Artificial Intelligence Research*, 63:743–788, 2018.

Augustine Cauchy. Méthode générale pour la résolution de systèmes d'équations simultanées. *Comptes Rendus Hebd. Seances Acad. Sci*, 25:536–538, 1847.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal Sentence Encoder. *arXiv*, 2018.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for Natural Language Inference. *arXiv preprint*, 2017.

Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree Neural Networks for Program Translation. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montréal, Canada, 2018. NeurIPS.

Zhou Cheng, Chun Yuan, Jiancheng Li, and Haiqin Yang. TreeNet: Learning Sentence Representations with Unconstrained Tree Structure. In *Proceedings of the*

Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), pages 4005–4011. IJCAI, 2018.

Billy Chiu, Anna Korhonen, and Sampo Pyysalo. Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance. In *Proceedings of the 1st Workshop on Evaluating Vector Space Representations for NLP*, pages 1–6, Berlin, Germany, aug 2016. ACL. ISBN 2003203,621.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. ACL, 2014.

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep Big Simple Neural Nets Excel on Hand-written Digit Recognition. *Neural computation*, 22(12):3207–3220, 2010.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

Alexis Conneau, Holger Schwenk, Yann Le Cun, and Lo¨ıc Lo¨ıc Barrault. Very Deep Convolutional Networks for Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 1107–1116, Valencia, Spain, apr 2017. ACL.

Claude Coulombe. Text Data Augmentation Made Simple By Leveraging NLP Cloud APIs. *arXiv*, 2018.

Ido Dagan, Oren Glickman, Bernardo Magnini, and Ramat Gan. The PASCAL Recognising Textual Entailment.pdf. *Machine Learning Challenges*, pages 177–190, 2006.

Marco Damonte and Shay B Cohen. Structural Neural Encoders for AMR-to-text Generation. *arXiv*, 2019.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34 th International Conference on Machine Learning*, Sydney, Australia, 2017. ICML.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova Google, and A I Language. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv*, 2018.

Li Dong, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. Learning to Paraphrase for Question Answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 875–886, Copenhagen, Denmark, sep 2017. EMNLP.

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

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding Back-Translation at Scale. *arXiv*, page 12, 2018.

Elozino Egonmwan and Yllias Chali. Transformer and seq2seq model for Paraphrase Generation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation (WNGT 2019)*, pages 249–255, Hong Kong, China, nov 2019. ACL.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 03640213. doi: 10.1016/0364-0213(90)90002-E.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. Tree-to-Sequence Attentional Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 823–833, Berlin, Germany, aug 2016. ACL.

Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. Retrofitting Word Vectors to Semantic Lexicons. In *Proceedings of NAACL*, 2015.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing Search in Context: The Concept Revisited. *ACM Transactions on information systems*, 20(1):116–131, 2002.

J R Firth. A Synopsis of Linguistic Theory, 1930-1955. In *Studies in Linguistic Analysis*. Basil Blackwell, 1962.

Daniel Fried and Kevin Duh. Incorporating Both Distributional and Relational Semantics in Word Representations. *arXiv*, 2015.

Luca Gasparri and Diego Marconi. Word Meaning, 2016.

Luca Gasparri and Diego Marconi. Word Meaning, 2019.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia, 2017. ICML.

George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995. ISSN 00010782. doi: 10.1145/219717.219748.

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2173–2182, Austin, TX, USA, nov 2016. EMNLP.

Anna Gladkova and Aleksandr Drozd. Intrinsic Evaluations of Word Embeddings: What Can We Do Better? In *Proceedings of the 1st Workshop on Evaluating Vector Space Representations for NLP*, pages 36–42, Berlin, Germany, aug 2016. ACL.

Goran Glavaš and Ivan Vulić. Generalized Tuning of Distributional Word Vectors for Monolingual and Cross-Lingual Lexical Entailment. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4824–4830, Florence, Italy, aug 2019. ACL.

Yoav Goldberg. *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers, Toronto, Canada, first edit edition, 2017. ISBN 9781627052955.

Christoph Goller and Andreas Küchler. Learning Task-Dependent Distributed Representations by Backpropagation Through Structure Learning Task-Dependent Distributed Representations by Backpropagation Through Structure. Technical report, 1996.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. *arXiv*, 2015.

Qipeng Guo, Xipeng Qiu, Xiangyang Xue, and Zheng Zhang. Top-Down Tree Structured Text Generation. *arXiv*, 2018.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 297–304, Sardinia, Italy, 2010. AISTATS.

Michael U Gutmann and Aapo Hyvarinen. Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics Aapo Hyvärinen. *Journal of Machine Learning Research*, 13:307–361, 2012.

Zellig S Harris. Distributional Structure. *WORD*, 10(3):146–162, 1954. doi: 10.1080/00437956.1954.11659520.

Felix Hill, Roi Reichart, and Anna Korhonen. SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. *Computational Linguistics*, 41(4):665–695, 2015. doi: 10.1162/COLI.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning Distributed Representations of Sentences from Unlabelled Data. In *Proceedings of NAACL-HLT 2016*, pages 1367–1377, San Diego, CA, USA, 2016. ACL.

G E Hinton, J L Mcclelland, and D E Rumelhart. Distributed Representations. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, chapter 3, pages 77–109. MIT Press, Cambridge, MA, USA, 1986.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing, 2017.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989.

Jeremy Howard and Sebastian Ruder. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pages 328–339, Melbourne, Australia, 2018. ACL.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving Word Representations via Global Context and Multiple Word Prototypes.

In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 873–882, Jeju, Republic of Korea, jul 2012. ACL.

Zhiheng Huang, Baidu Research, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv*, 2015.

Zhiwei Jiang, Qing Gu, Yafeng Yin, and Daoxu Chen. Enriching Word Embeddings with Domain Knowledge for Readability Assessment. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 366–378, Santa Fe, NM, USA, 2018. ACL.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, 2018. ICML.

Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 143–151, Pittsburgh, PA, USA, 1997. Carnegie Mellon University.

Rie Johnson and Tong Zhang. Deep Pyramid Convolutional Neural Networks for Text Categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 562–570, Vancouver, Canada, jul 2017. ACL. doi: 10.18653/v1/P17-1052.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 427–431, Valencia, Spain, apr 2017. ACL.

Aishwarya Kamath, Jonas Pfeiffer, Edoardo M Ponti, Goran Glavaš, and Ivan Vulić. Specializing Distributional Vectors of All Words for Lexical Entailment. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 72–83, Florence, Italy, aug 2019. ACL.

Edward L. Keenan and Edward P. Stabler. Language variation and linguistic invariants. *Lingua*, 120(12):2680–2685, dec 2010. ISSN 00243841. doi: 10.1016/J.LINGUA.2010.04.011.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, oct 2014. EMNLP.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-Aware Neural Language Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI, 2016.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured Attention Networks. In *ICLR 2017*. ICLR, 2017.

Diederik P Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. In *ICLR 2015*. ICLR, 2015.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *ACL*, 4:313–327, 2016. ISSN 2307-387X.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-Thought Vectors. In *Advances in Neural Information Processing Systems*, pages 3294–3302. NIPS, 2015.

Sosuke Kobayashi. Contextual Augmentation: Data Augmentation by Words with Paradigmatic Relations. In *Proceedings of NAACL-HLT 2018*, pages 452–457. ACL, 2018.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text Generation from Knowledge Graphs with Graph Transformers. In *Proceedings of NAACL-HLT 2019*, pages 2284–2293, Minneapolis, MN, USA, jun 2019. ACL.

András Kornai. Mathematical Linguistics. Technical report, 2001.

Marcus Kracht. Introduction to Linguistics. Technical report, UCLA, 2007.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105. NeurIPS, 2012.

Matt J Kusner, Yu Sun, Nicholas I Kolkin, and Kilian Q Weinberger. From Word Embeddings To Document Distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

Siwei Lai, Kang Liu, Liheng Xu, and Jun Zhao. How to Generate a Good Word Embedding? *IEEE Intelligent Systems*, 31(6):5–14, 2016.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural Architectures for Named Entity Recognition. In *Proceedings of NAACL-HLT 2016*, pages 260–270, San Diego, CA, USA, 2016. ACL.

Anne Lauscher, Ivan Vulić, Edoardo Maria Ponti, Anna Korhonen, and Goran Glavaš. Informing Unsupervised Pretraining with External Linguistic Knowledge. *arXiv*, 2019.

Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China, 2014a. JMLR.

Qv Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. *International Conference on Machine Learning - ICML 2014*, 32, 2014b. ISSN 10495258. doi: 10.1145/2740908.2742760.

Rémi Lebret and Ronan Collobert. Word Embeddings through Hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 482–490, Gothenburg, Sweden, apr 2014. ACL.

Yan LeCun, B Boser, J S Denker, D Henderson, R E Howard, W Hubbard, and L D Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In

Advances in neural information processing systems 2, pages 396–404. NeurIPS, 1989.

Yann LeCun. Generalization and Network Design Strategies. Technical report, University of Toronto, Toronto, Ontario, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, nov 1998.

Dongkyu Lee, Zhiliang Tian, Lanqing Xue, and Nevin L Zhang. Enhancing Content Preservation in Text Style Transfer Using Reverse Attention and Conditional Layer Normalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 93–102. ACL, aug 2021.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 08936080. doi: 10.1016/S0893-6080(05)80131-5.

Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. *Advances in neural information processing systems*, pages 2177–2185, 2014.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. Seq2seq Dependency Parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, NM, USA, aug 2018. ACL.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, Yoshua Bengio, and Ibm Watson. A Structured Self-Attentive Sentence Embedding. In *ICLR*. ICLR, 2017.

Wang Ling, Tiago Luis, Luís Luis Marujo, Ramon Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio

Amir, Luís Luis Marujo, and Tiago Luís. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (September), 2015. doi: 10.18653/v1/D15-1176.

Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. Learning Semantic Word Embeddings based on Ordinal Knowledge Constraints. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1501–1511. ACL, 2015.

Yang Liu and Mirella Lapata. Learning Structured Text Representations. *arXiv*, 2018.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov, and Paul G Allen. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv*, 2019.

Jie Lu, Vahid Behbood, Peng Hao, Hua Zuo, Shan Xue, and Guangquan Zhang. Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, 80:14–23, 2015. doi: 10.1016/j.knosys.2015.01.010.

Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Dawei Song, and Ming Zhou. A Tensorized Transformer for Language Modeling. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 2019. NeurIPS.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Kelly Marchisio, Youngser Park, Ali Saad-Eldin, Anton Alyakin, Kevin Duh, Carey Priebe, and Philipp Koehn. An Analysis of Euclidean vs. Graph-Based Framing for Bilingual Lexicon Induction from Word Embedding Spaces. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 738–749. EMNLP, nov 2021.

Robert May. The Invariance of Sense. *The Journal of philosophy*, 103(3):111–144, 2006.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in Translation: Contextualized Word Vectors. In *31st Conference on Neural Information Processing Systems*, pages 6297–6308, Long Beach, CA, USA, 2017. NIPS.

Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165, 1989. ISSN 00797421. doi: 10.1016/S0079-7421(08)60536-8.

R Thomas McCoy, Robert Frank, and Tal Linzen. Does Syntax Need to Grow on Trees? Sources of Hierarchical Inductive Bias in Sequence-to-Sequence Networks. *Transactions of the Association for Computational Linguistics*, 8:125–140, 2020. doi: 10.1162/tacl.

Gábor Melis, Tomáš Kočiský, and Phil Blunsom. Mogrifier LSTM. In *ICLR 2020*. ICLR, 2020.

Jean Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, Steven Pinker, Martin A. Nowak, and Erez Lieberman Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014): 176–182, 2011. ISSN 00368075. doi: 10.1126/science.1199644.

Margot Mieskes. A Quantitative Study of Data in the NLP community. In *Pro-*

ceedings of the First Workshop on Ethics in Natural Language Processing, pages 23–29, Valencia, Spain, apr 2017. ACL.

Margot Mieskes, Karën Fort, Aurélie Névéol, Cyril Grouin, and Kevin Cohen. NLP Community Perspectives on Replicability. In *Proceedings of Recent Advances in Natural Language Processing*, pages 768–775, Varna, Bulgaria, sep 2019. RANLP. doi: 10.26615/978-954-452-056-4_089.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *ICLR*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, pages 3111–3119, 2013b.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel., J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-Ending Learning. *Communications of the ACM*, 61(5):103–115, 2018.

Andriy Mnih and Geoffrey Hinton. Three New Graphical Models for Statistical Language Modelling. In *n Proceedings of the 24 th International Conference on Machine Learning*, Corvallis, OR, USA, 2007. ICML.

Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In C. J. C. Burges and L. Bottou and M. Welling and Z. Ghahramani and K. Q. Weinberger, editor, *Advances in Neural Information Processing Systems 26*, pages 2265–2273. NeurIPS, 2013.

Andriy Mnih and Yee W Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29 th International Conference on Machine Learning*, Edimburgh, Scotland, 2012. ICML.

Christopher Moody. Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec. *arXiv*, 2016.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural Language Inference by Tree-Based Convolution and Heuristic Matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 130–136, Berlin, Germany, 2016a. ACL.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural Language Inference by Tree-Based Convolution and Heuristic Matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 130–136. ACL, 2016b.

Khalil Mrini, Franck Dernoncourt, Trung Bui, Walter Chang, and Ndapa Nakashole. Rethinking Self-Attention: Towards Interpretability in Neural Parsing. *arXiv*, may 2020.

Nikola Mrkšic, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašigašic, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-fitting Word Vectors to Linguistic Constraints. In *Proceedings of NAACL-HLT 2016*, pages 142–148, San Diego, CA, USA, 2016. ACL.

Nikola Mrkšić, Ivan Vulić, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gašić, Anna Korhonen, and Steve Young. Semantic Specialization of Distributional Word Vector Spaces using Monolingual and Cross-Lingual Constraints. *Transactions of the Association for Computational Linguistics*, 5:309–324, 2017.

N.d. Project Gutenberg.

Kim Anh Nguyen, Maximilian Köper, Sabine Schulte Im Walde, and Ngoc Thang Vu. Hierarchical Embeddings for Hypernymy Detection and Directionality. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 233–243, 2017.

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, New York, NY, 1999.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 2009. doi: 10.1109/TKDE.2009.191.

Christos H Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. Latent Semantic Indexing: A Probabilistic Analysis. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168, Seattle, WA, USA, may 1998. ACM.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. ISSN 18792782. doi: 10.1016/j.neunet.2019.01.012.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30 th International Conference on Machine Learning*, Atlanta, GA, USA, 2013. ICML.

Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers)*, pages 425–430, Beijing, China, 2015. ACL.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

James W Pennebaker, Ryan L Boyd, Kayla Jordan, and Kate Blackburn. The development and psychometric properties of LIWC2015. Technical report, University of Texas at Austin, Austin, TX, USA, 2015.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. *EMNLP*, pages 1532–1543, 2014.

Matthew E Peters, Mark Neumann, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT 2018*, pages 2227–2237, New Orleans, LA, USA, jun 2018. ACL.

Steven T. Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin and Review*, 21(5):1112–1130, 2014. ISSN 15315320. doi: 10.3758/s13423-014-0585-6.

Judicael Poumay and Ashwin Ittoo. A Comprehensive Comparison of Word Embeddings in Event & Entity Coreference Resolution. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2755–2764. EMNLP, nov 2021.

Ning Qian. On the Momentum Term in Gradient Descent Learning Algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. *Preprint*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI Blog*, 1(8), 2019.

Jonathan Raiman and John Miller. Globally Normalized Reader. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1059–1069, Copenhagen, Denmark, sep 2017. EMNLP.

Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural Network-Based Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (1):23–38, 1998.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A Metric for Distributions with Applications to Image Databases. In *Sixth International Conference on Computer Vision*, pages 59–66, 1998.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

Ryohei Sasano and Anna Korhonen. Investigating Word-Class Distributions in Word Vector Spaces. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3657–3666. ACL, 2020.

Nathaniel Schenker and Jane F Gentleman. On Judging the Significance of Differences by Examining the Overlap Between Confidence Intervals. *The American Statistician*, 55(3):182–186, 2001.

Martin Schmitt, Sahand Sharifzadeh, Volker Tresp, Hinrich Schütze, and Sch¨ Schütze. An Unsupervised Joint System for Text Generation from Knowledge Graphs and Semantic Parsing. *arXiv*, 2020.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015. EMNLP. ISBN 9781941643327. doi: 10.18653/v1/D15-1036.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving Neural Machine Translation Models with Monolingual Data. *arXiv*, 2016a.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, Berlin, Germany, aug 2016b. ACL. ISBN 3000050000.

Patrice Y Simard, Dave Steinkraus, and John C Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 958–963, 2003.

Jonas Sjöberg, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, dec 1995. ISSN 00051098. doi: 10.1016/0005-1098(95) 00120-8.

Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Association for the Advancement of Artificial Intelligence 2017*. AAAI, 2017.

Stanford NLP Group. Stanford Tokenizer.

Mervyn Stone. Cross-Validatory Choice and Assessment of Statistical Predictions on JSTOR. *ournal of the Royal Statistical Society. Series B (Methodological)*, 36 (2):111–147, 1974.

Masashi Sugiyama. Chapter 35 - Linear Dimensionality Reduction. pages 405–428. Morgan Kaufmann, Boston, 2016. ISBN 978-0-12-802121-7. doi: https://doi.org/10.1016/B978-0-12-802121-7.00046-7.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30 th International Conference on Machine Learning*, Atlanta, GA, USA, 2013. ICML.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. *NIPS*, pages 3104–3112, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks.

In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1556–1566, Beijing, China, jul 2015. ACL.

Adly Templeton. Word Equations: Inherently Interpretable Sparse Word Embeddings through Sparse Coding. In *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 177–191. ACL, nov 2021.

Naftali Tishby, Fernando C Pereira, and William Bialek. The Information Bottleneck Method. In *Proc. of the 37th Allerton Conference on Communication, Control and Computing*, 1999.

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. Evaluation of Word Vector Representations by Subspace Alignment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2049–2054, Lisbon, Portugal, sep 2015. EMNLP.

Peter D Turney. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, 37:141–188, 2010.

Shikhar Vashishth, Manik Bhandari, Prateek Yadav, Piyush Rai, Chiranjib Bhattacharyya, and Partha Talukdar. Incorporating Syntactic and Semantic Information in Word Embeddings using Graph Convolutional Networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3308–3318, Florence, Italy, jul 2019. ACL.

Ashish Vaswani, Google Brain, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017. NIPS.

Ivan Vulić and Nikola Mrkšic. Specialising Word Vectors for Lexical Entailment. In *Proceedings of NAACL-HLT 2018*, pages 1135–1145, New Orleans, LA, USA, 2018. ACL.

Ivan Vulić, Goran Glavaš, Nikola Mrkšić, and Anna Korhonen. Post-Specialisation: Retrofitting Vectors of Words Unseen in Lexical Resources. In *Proceedings of NAACL-HLT 2018*, pages 516–527. ACL, 2018.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv*, page 13, 2018a.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 3266–3280, Vancouver, Canada, 2019a. NeurIPS.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *ICLR 2019*. ICLR, 2019b.

Bin Wang, Angela Wang, Chen Fenxiao, Yuncheng Wang, and C.-C Jay Kuo. Evaluating Word Embedding Models: Methods and Experimental Results. *arXiv:1901.09785*, 2019c.

Chenguang Wang, Mu Li, and Alexander J Smola. Language Models with Transformers. *arXiv*, 2019d.

Guotai Wang, Wenqi Li, Michael Aertsen, K U Leuven, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. Test-time augmentation with uncertainty estimation for deep learning-based medical image segmentation. In *1st Conference on Medical Imaging with Deep Learning (MIDL 2018)*, Amsterdam, The Netherlands, 2018b. MIDL.

Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint Embedding of Words and Labels for Text Classification. In *Proceedings of the 56th Annual Meeting of* *the Association for Computational Linguistics (Long Papers)*, pages 2321–2331, Melbourne, Australia, jul 2018c. ACL.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. In *ICLR 2018*. ICLR, 2018.

Laura Wendlandt, Jonathan K Kummerfeld, and Rada Mihalcea. Factors Influencing the Surprising Instability of Word Embeddings. In ACL, editor, *Proceedings of NAACL-HLT 2018*, pages 2092–2102, New Orleans, LA, USA, 2018.

Paul J Werbos. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78(10):1550–1560, oct 1990.

John Wieting and Kevin Gimpel. Revisiting Recurrent Networks for Paraphrastic Sentence Embeddings. *arXiv:1705.00364*, 2017.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. ACL, 2017.

Ronald J. Williams and David Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280, jun 1989. ISSN 0899-7667. doi: 10.1162/NECO.1989.1.2.270.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017. NeurIPS.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick Von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 EMNLP (Systems Demonstrations)*, pages 38–45, 2020.

Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6. IEEE, 2016.

Lingfei Wu, Ian En-Hsu Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J Witbrock. Word Mover's Embedding: From Word2Vec to Document Embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4524–4534, Brussels, Belgium, 2018. ACL.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv*, 2016a.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv*, 2016b.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised Data Augmentation for Consistency Training. *arXiv*, 2019.

Yadollah Yaghoobzadeh and Hinrich Schütze. Intrinsic Subspace Evaluation of Word Embedding Representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 236–246, 2016.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 2019. NeurIPS.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, aug 2018.

Mo Yu and Mark Dredze. Improving Lexical Embeddings with Semantic Knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers)*, pages 545–550, Baltimore, MD, USA, 2014. Association for Computational Linguistics.

Matthew D Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv*, 2012.

Meishan Zhang, Nan Yu, and Guohong Fu. A Simple and Effective Neural Model for Joint Word Segmentation and POS Tagging. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1528–1538, sep 2018.

Xiang Zhang, Junbo Zhao, and Yann Lecun. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems*, pages 649–657. NeurIPS, 2016. ISBN 0123456789.

Yian Zhang, Alex Warstadt, Haau-Sing Li, and Samuel R Bowman. When Do You Need Billions of Words of Pretraining Data? *arXiv*, 2020.

Jiangbin Zheng, Yile Wang, Ge Wang, Jun Xia, Yufei Huang, Guojiang Zhao, Yue Zhang, and Stan Z Li. Using Context-to-Vector with Graph Retrofitting to Improve Word Embeddings. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics Volume 1: Long Papers*, volume 1, pages 8154–8163. Long Papers, may 2022.

Yi-Tong Zhou and Rama Chellappa. Computation of optical flow using a neural network. In *International Conference on Neural Networks*, San Diego, CA, USA, jul 1988. IEEE.

George Kingsley Zipf. *The Psychobiology of Language*. Routledge, London, UK, 1936.