On the surprising capacity of linear combinations of embeddings for natural language processing

Lyndon White
BCM in Computation and Pure Mathematics;
BE in Electrical and Electronic Engineering
March 27, 2019

This thesis is presented for the degree of
Doctor of Philosophy
of The University of Western Australia

Thesis Declaration

I, Lyndon White, certify that:

This thesis has been substantially accomplished during enrolment in the degree.

This thesis does not contain material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution.

No part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of The University of Western Australia and where applicable, any partner institution responsible for the joint-award of this degree.

This thesis does not contain any material previously published or written by another person, except where due reference has been made in the text.

The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

The work described in this thesis was partially funded by Australian Research Council grants DP150102405 and LP110100050; and by a Australian Government Research Training Program (RTP) Scholarship.

This thesis contains published work and/or work prepared for publication, some of which has been co-authored.

Lyndon White
March 27, 2019

In memoriam of
Laurie White
1927–2018

Abstract

As Webster's classic 1900 text *"English: Composition and Literature"* states "A sentence is a group of words expressing a complete thought." People use natural language to represent thoughts. Thus the representation of natural language, in turn, is of fundamental importance in the field of artificial intelligence. Natural language understanding is a research area which revolves around how to represent text in a form that an algorithm can manipulate in such a way as to mimic the ability of a human to truly understand the text's meaning. In this dissertation, we aim to extend the practical reach of this area, by exploring a commonly overlooked method for natural language representation: linear combinations (i.e. weighted sums) of embedded representations. This dissertation is organised as a collection of research publications: with our novel contributions published in conference proceedings or journals; and with a comprehensive literature review published as part of a book.

When considering how to represent English input into a natural language processing system, a common response is to view it as a sequential modelling problem: as a discrete time-series of words. A more complex alternative is to base the input model on the grammatical tree structures used by linguists. But there are also simpler models: systems based on just summing the word embeddings. Word embeddings are dense vector representations of words, generally learned as part of a neural network or related system. They are commonly available pretrained, and are used in a variety of systems for different tasks to those they were originally trained for as a form of transfer learning. On a variety of tasks, simply summing these word embeddings work very well – often better than the more complex models. This dissertation examines these linear combinations of embeddings for natural language understanding tasks.

In brief, it is found that a sum of embeddings is a particularly effective dimensionality-reduced representation of a bag of words. The dimensionality reduction is carried out at the word level via the implicit matrix factorization on the collocation probability matrix. It thus captures into the dense word embeddings the key features of lexical semantics: words that occur in similar contexts have similar meanings. We find that summing these representations of words gives us a very useful representation of structures built upon words: such as sentences, phrases, and word senses.

A limitation of the sum of embedding representation is that it is unable to represent word order. This representation does not capture any order related information; unlike for example a recurrent neural network. Recurrent neural networks, and other more complex models, are outperformed by sums of embeddings in tasks where word order is not highly significant. It is found that even in tasks where word order does matter to an extent, the improved training capacity of the simpler model still means that it performs better than more complex models. This limitation thus impacts surprisingly little.

Acknowledgements

I must begin by expressing my gratitude towards my supervisors, Prof. Roberto Togneri, Dr Wei Liu, and Winthrop Prof. Mohammed Bennamoun. I am very fortunate to have such attentive supervisors, who have supported me in my endeavours. This thesis would not be here without their on-going support and advise.

Further, I wish to more generally thank the staff and faculty from the departments of electrical engineering and of computer science who've helped me during this process.

I would like to express how much I enjoyed working with Naeha Sharif on several projects. I look forward to reading her thesis in a few years time.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship. The lack of which no doubt would have resulted in my expiration, and the attendant adverse consequences on the completion of this dissertation. It was also supported by funding from Australian Research Council grants DP150102405 and LP110100050.

During my research computing hardware and power was provided due to the support of the National eResearch Collaboration Tools and Resources project (Nectar), Nvidia, and the UWA University Computer Club (UCC).

I also to acknowledge the support and assistance I've received from my open-source collaborators and compatriots in the JuliaLang community. In particular: Chris Rackauckas, Chrisof Stocker, and Jon Malmaud; though I could easily list a dozen more. One could not ask for a more collegial online community of academics and programmers.

I am grateful for the ongoing support and companionship of my friends and family. In particular, my good friend Roland Kerr, who began his Research Masters at the same time as I was beginning my PhD.

Lastly I must thank my darling wife, Isobel, who has supported me constantly; and has always forgiven me when I am home hours late after just doing "one last thing".

Authorship declaration

Publications

This thesis contains work that has been published and/or prepared for publication.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018a). *Neural Representations of Natural Language*. Studies in Computational Intelligence (Book). Springer Singapore. ISBN: 9789811300615
Location in thesis: Part I
Student contribution to work:
Determined content. Created figures. Wrote book. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2015). "How Well Sentence Embeddings Capture Meaning". In: *Proceedings of the 20th Australasian Document Computing Symposium*. ADCS '15. Parramatta, NSW, Australia: ACM, 9:1–9:8. ISBN: 978-1-4503-4040-3. DOI: 10.1145/2838931.2838932
Location in thesis: ??
Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon. White, Roberto. Togneri, Wei. Liu, and Mohammed Bennamoun (2018). "Learning of Colors from Color Names: Distribution and Point Estimation". In: *Computational Lingustics (Under Review)*
Location in thesis: Chapter 5
Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "Finding Word Sense Embeddings Of Known Meaning". In: *19th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*
Location in thesis: ??
Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "NovelPerspective: Identifying Point of View Characters". In: *Proceedings of ACL 2018, System Demonstrations*. Association for Computational Linguistics
Location in thesis: ??
Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016a). "Generating Bags of Words from the Sums of their Word Embeddings". In: *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*
Location in thesis: ??
Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016b). "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem". In: *IEEE International Conference on Data Mining: High Dimensional Data Mining Workshop (ICDM: HDM)*. DOI: 10.1109/ICDMW.2016.0113
Location in thesis: ??
Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018a). "DataDeps.jl: Repeatable Data Setup for Reproducible Data Science". In: *Journal of Open Research Software (Under Review)*
Location in thesis: Appendix A
Student contribution to work:
Primary author of software. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

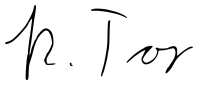
Details of the work:
Lyndon White and Sebastin Santy (2018). "DataDepsGenerators.jl: making reusing data easy by automatically generating DataDeps.jl registration code". In: *Journal of Open Source Software*
Location in thesis: Appendix B
Student contribution to work:
Original author of software. Provided direction, guidance, and code review for its enhancement. Wrote publication.

Details of the work:
Lyndon White and David Ellison (2018). "Embeddings.jl: easy access to pretrained word embeddings from Julia". In: *Journal of Open Source Software (Under Review)*
Location in thesis: Appendix C
Student contribution to work:
Original and primary author of software. Wrote publication.

Details of the work:
Jonathan Malmaud and Lyndon White (2018). "TensorFlow.jl: An Idiomatic Julia Front End for TensorFlow". In: *Journal of Open Source Software*. DOI: 10.21105/joss.01002
Location in thesis: Appendix D
Student contribution to work:
Co-maintainer and second highest contributor to the software. Co-wrote publication.

Permission to use work in thesis

The coauthors signing below give permission to use the aforementioned works in this dissertation, and certify that the student's statements regarding their contribution to the respective co-authored works listed above are correct.

Roberto Togneri:
Primary Supervisor  04/10/18

Wei Liu:
Supervisor  04/10/18

Mohammed Bennamoun:
Supervisor  04/10/18

Sebastin Santy:  24/09/18

David Ellison:  02/10/18

Jonathan Malmaud:  24/09/18

Contents

1 Introduction **1**

I Literature Review **19**

2 Word Representations **21**

3 Word Sense Representations **47**

4 Sentence Representations and Beyond **59**

II Publications **75**

5 Learning of Colors from Color Names: Distribution and Point Estimation **77**

6 Conclusion **111**

Bibliography **115**

III Appendix: Tooling **125**

A DataDeps.jl: Repeatable Data Setup for Replicable Data Science **127**

B DataDepsGenerators.jl: Making Reusing Data Easy by Automatically Generating DataDeps.jl Registration Code **135**

C Embeddings.jl: Easy Access to Pretrained Word Embeddings from Julia **139**

D TensorFlow.jl: An Idiomatic Julia Front End for TensorFlow **141**

Chapter 1

Introduction

It has been a continual surprise, that simple combinations of embeddings perform so well for a variety of tasks in natural language processing. At first glance, such simple methods capturing only unordered word use should have little capacity in representing the rich and highly structured human language. However at a second glance, similar surface information has been used in information retrieval with great success since the inception of the field (Maron 1961). Linear combinations of embeddings can be considered as a dimensionality reduction of a bag of words, with a particular weighting scheme. Dimensionality reduction can be characterised as finding the best low dimensional representation of a high dimensional input according to some quality criterion. In the case of word embeddings, that quality criterion is generally related to the ability to predict the co-occurring words – a salient quality of lexical semantics. As such, linear combinations of embeddings take as input a very sparse high dimensional bag of words (which is itself a strong surface form representation), then reduce it to a dense representation that captures lexical semantics.

When we discuss linear combinations of word embeddings (LCOWE), we are considering various forms of weighted sums of vector word representations. These models are equivalent to representing bags of words (BOW), and are sometimes called *bags of vectors* (Conneau et al. 2018), or *embedding-BOW* (Cífka and Bojar 2018) or similar. The primary focus of this work has been on sums of word embeddings (SOWE), i.e. linear combinations with unit weights. Closely related to this is a mean of word embeddings (MOWE), which is a sum weighted such that it normalizes over the size of the bag of words. More complicated weightings, such as using probabilities, or term significance are also options for constructing LCOWEs.

The mechanism behind the functioning of the addition of word embeddings capturing their combined meaning, was partially explained in one of the pioneering works on word embeddings (Mikolov et al. 2013a). As shown below, for w and u being words, C being an embedding matrix, and $P(\mathbb{V} \mid a)$ being the set of probabilities for each word in the vocabu-

lary $\mathbb{V}$ co-occurring with the word a.

$$C_{:,w} \propto \log P(\mathbb{V} \mid w) \tag{1.1}$$
$$C_{:,u} \propto \log P(\mathbb{V} \mid u) \tag{1.2}$$
$$\therefore C_{:,w} + C_{:,u} \propto \log P(\mathbb{V} \mid w) + \log P(\mathbb{V} \mid u) \tag{1.3}$$
$$= \log P(\mathbb{V} \mid w) \cdot P(\mathbb{V} \mid u) \tag{1.4}$$
$$\propto \log P(\mathbb{V} \mid w \cap u) \tag{1.5}$$

They note that under the skip-gram model, there is a linear relationship between a word embedding and the logarithm of the probability distribution over co-occurring words.[1] Thus there is a linear relationship between the sum of two (or more) embeddings, and the product of the probability distribution over co-occurring words. Which is roughly proportional to the probability distribution over words co-occurring with that two-word bigram (or n-gram).[2] Which is to, say it is proportional to the distribution estimate that would have been found had that bigram (or n-gram) been replaced with a single token. By the distributional hypothesis, the similarity of meaning is characterised by the distribution of words that may co-occur. This is how skip-gram-like word embeddings function, and this relationship explains why its ability to represent meaning similarity generalizes to sums of the word embeddings for short phrases. If one considers this for larger structures than phrases, giving a larger bag-of-of-words, it can be considered that a sum of word embeddings, is proportional to the distribution over other worlds of the likelihood to co-occur with the entire bag of words. Interestingly, this is a distribution over the vocabulary, such that words that could have been present and included in the BOW have a high likelihood.

Throughout the last three years that we have been researching this problem, others have also found, often to their own surprise, the strength of simple linear combinations of embeddings.

Arora, Liang, and Ma (2017)'s work describes a "A simple but tough-to-beat baseline for sentence embeddings", which is a linear combination of word embeddings. Their proposed model is a more complicated combination than considered here. But never-the-less, it is primarily a weighted sum of embeddings, with small adjustments based on linear dimensionality reduction methods. In particular when using the word embeddings of Wieting et al. (2016), they find this to be very competitive when compared with more complex models which take into account word order.

Cífka and Bojar (2018) found that taking a mean of word embeddings outperformed almost all of their more sophisticated machine-translation-based sentence representations, when used on classification and paraphrase detection tasks. This is not to say that linear combinations of embeddings are ideal models for all tasks. They clearly cannot truly handle all the complexities of language. But rather that the occurrence of the complexities they cannot handle is rarer in practice in many tasks than is often expected.

[1] The log in the relationship explains why summing embeddings works well, but taking their product does not. While the sum of two log-likelihoods is a log of the product of likelihoods, the product of two log likelihoods does not correspond to anything with intuitive meaning.

[2] This is only a rough relationship as it depends on the assumption of independence.

Conneau et al. (2018) constructed 10 probing tasks to isolate some of the information captured by sentence representations. They found the strong performance of the mean of word embeddings on sentence level tasks to be striking. They attribute it to the sentence level information being redundantly encoded in the word-forms: the surface level information is surprisingly useful for tasks which at first look very sophisticated. With the exception of their word-content task, they did find that more sophisticated models are able to perform better than the mean of word embeddings. However, when correlating the performance of their probing task against real world tasks, they found that the word-content probing task was by far the most positively correlated with the real word tasks. This makes it clear how valuable this surface information is in practical tasks.

In the work presented in this dissertation, we find that even in tasks where it would seem that non-surface information incorporating word-order is required, in practice other issues cause the more powerful models that are (theoretically) able to handle these situations correctly to be never-the-less outperformed. This is particularly the case where the theoretical improvement from incorporating this information is small, relative to the practical complexity of the techniques that are required to leverage it. Such a case where word order matters but the error from ignoring it is small, is particular illustrated in Chapter 5.

At a high-level, the success of these techniques comes down to that fact that most human language is easy to understand and simple. This expectation of language being easily understood is highlighted by the work of Grice (1975), which claims that the communication is conducted following a cooperative principle. The overall supermaxim for Grice's cooperative principle is that the speakers are expected to "be perspicuous" i.e. to use speech that is clearly expressed and easily understood. The particular relevant maxims within the cooperative principle are: the *maxim of quantity*, that speakers are expected to make contributions that are no more, nor less informative than required; and the *maxim of manner*: that speakers are expected to avoid ambiguity and obscurity of expression, and to make contributions that are brief and orderly. While Grice originally proposed these are exceptions upon conversation, the general principle applies more broadly to natural language communication. This general principle being that language used is normally expected to be understood easily – thus fulfilling the goal of communicating.

Adversarial examples are reasonably easy to construct. An adversarial example to a linear combination of word embeddings is any text where the word order significantly affects that meaning; and where multiple possible word orders exist. For such an adversary to be significant, both word orders must be reasonably likely to occur. However; such cases are rarer than one might expect as is demonstrated in **??**. Particularly when punctuation tokens are included in the embeddings. As such, while these cases certainly exist, we find that for real applications they are sufficiently rare that the simplicity of the linear combinations of embeddings approach can work very well.

The strong performance of LCOWE when applied in sentence or phrase representation contexts, as discussed in **??** and Chapter 5, gives support

to the notion that often word order us not a very significant feature in determining meaning. One would think that word order, and other factors of linguistic structure must contribute significantly to the meaning of the phrase. However, our results suggest that it is often in a minor way, and that for many tasks these linear combinations are superior due to their simplicity and effectiveness. While taking into account the linguistic structure may be the key to bridging the gap between "almost perfect" and "perfect", the current state of the field for many tasks has not reached "almost perfect", and as such simpler methods still form an important part. The successes of the sums of word embeddings for sentence and phrase embeddings, leads us to consider other uses of linear combinations for representation. **??** and **??** consider tasks well outside of phrase representation where the order clearly does not matter: namely word-sense representation, and context of named entity usage across a document.

To further understand the relationship between SOWE and BOW, and the extent to which word order matters, **??** and **??** investigate if it is possible to reverse the conversion from sentence to SOWE. The results in **??** show that it is largely possible to reconstruct bags of words from SOWE, suggesting that when considered as a dimensionality reduction technique SOWE does not lose much information. This is extended in **??** to order those bags of words back to sentences via a simple tri-gram language model. This had some success at outright reconstructing the sentences. This highlights the idea that for many bags of words (which can be reconstructed from a sum of word embeddings) there may truly be only one reasonable sentence from which they might have come. This would explain why SOWE, and BOW, ignorance of word order does not prevent them from being useful representations of sentences.

One of the attractive features of these linear combinations is their simplicity. This is true both in an implementation sense, and in the sense of gradient descent. For example, the vanishing gradient problem in deep networks, especially RNNs (Bengio, Simard, and Frasconi 1994) and RvNNs (Socher 2014), simply does not exist for a sum of word embeddings. A sum of word embeddings is not a deep input structure – it is only one hidden layer. This is in contrast to recurrent neural networks (RNNs) which are deep in time: having effective depth $O(n)$ where n is the number of terms. Similarly, recursive neural networks (RvNNs) are deep in structure: having effective depth $O(\log n)$. Information does not have to propagate as far when a SOWE is used as an input representation. Thus it is easier to attribute changes during gradient descent. This is not to say that SOWE can only be used in a shallow network – it is simply an input representation subnetwork. Just like for RNNs and RvNNs, a deep network can be placed on top of the SOWE.

1.1 Thesis Outline and Contributions

This research tackles a number of natural language understanding problems, and in the solutions draws conclusions on the capacity of linear combinations of embeddings. The dissertation is organised into two parts.

Part I contains a detailed discussion of the established methods for input

representation in natural language understanding tasks. This literature review, however, does not focus on linear combinations of embeddings, which we develop upon through-out the rest of this dissertation. Rather it focuses upon the techniques we build upon, and the alternatives to our methods. Part I was originally published as the main content of our book *Neural Representations of Natural Language* (White et al. 2018a). It excludes the introductory chapters on machine learning and recurrent neural networks which were present in the book. Part II contains investigations on how LCOWE perform in key NLP tasks. These investigations constitute the bulk of this research effort. Further to the literature review section of this dissertation, each chapter in Part II includes a background or related works section with particularly relevant works to that paper discussed.

Part I: Chapter 2 Word Representations

We begin by introducing word embeddings in Chapter 2. Word embeddings form the basis of the work in this dissertation, and more so the basis of many of the advancements in the field more generally. The chapter begins with the consideration from a language modelling perspective, where word embeddings are equivalent to onehot input representations in a neural network being employed for a language modelling task. Then expands towards the considerations of word embeddings as more general purpose representations. This chapter also includes detailed tutorials explaining the details of hierarchical softmax and negative sampling.

Part I: Chapter 3 Word Sense Representations

Word sense representations are discussed in Chapter 3. These are of particular relevance to the work discussed in **??**. More generally the considerations of words having multiple senses informs the discussion of meaning representation more broadly.

Part I: Chapter 4 Sentence Representations and Beyond

Chapter 4 contains an overview of methods used for representing structures large than just words. In particular this section focuses on sentences, but also discusses techniques relevant to shorter phrases. This chapter contains some discussion of the sums of word embeddings that are the focus of this work, but primarily discusses the alternatives which we contrast against.

Overview of Novel Contributions (Part II)

An overview of the tasks investigated in this work is shown in Table 1.1. The representation of *sentences* is investigated in **??**, through a paraphrase grouping tasks. Similarly, the representation of *phrases* is investigated in Chapter 5 through a color understanding (estimation) task. Given the observed properties found by sums of word embeddings, this leads to the investigation into if weighted sums of word sense embeddings

Chapter	Structure	Task	Embeddings
??	Sentences	Paraphrase grouping	Word2Vec (Mikolov et al. 2013a)
Chapter 5	Short Phrases	Color understanding	FastText (Bojanowski et al. 2017)
??	Word Senses	Similarity with context & Word sense disambiguation	AdaGram (Bartunov et al. 2015) & Bespoke greedy sense embeddings
??	Adj. Contexts	POV character detection	FastText (Bojanowski et al. 2017)
??	Sentences	Recovering bags of words	GLoVE (Pennington, Socher, and Manning 2014)
??	Sentences	Recovering sentences	GLoVE (Pennington, Socher, and Manning 2014)

Table 1.1: Summary of the investigations published within this dissertation. The structure column gives the type of linguistic structure being worked with, the embeddings column lists the embedding methods investigated, and the task column describes the goal of the work.

might better resplendent a particular usage of a word in **??**. The capacity also lends to the investigation of using a sum of word embeddings to represent the contexts of all usages of a named entity, for the point-of-view character detection task investigated in **??**. We conclude with a pair of complementary works in **????**, which investigate the ability to recover bags of words and sentences, from SOWE represented sentences. These final works illustrate some of the reasons why linear combinations work so well.

Part II: ??. "How Well Sentence Embeddings Capture Meaning"

Originally published as: Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2015). "How Well Sentence Embeddings Capture Meaning". In: *Proceedings of the 20th Australasian Document Computing Symposium*. ADCS '15. Parramatta, NSW, Australia: ACM, 9:1–9:8. ISBN: 978-1-4503-4040-3. DOI: 10.1145/2838931.2838932.

We begin by examining methods for representing sentences. Sentences are a fundamental unit of communication – a sentence is a single complete idea. The core goal is to determine if different sentence embedding methods clearly separate the different ideas.

Paraphrases are defined by a bidirectional entailment relationship between two sentences. This is an equivalence relationship, it thus gives rise to a partitioning of all sentences in the space of a natural language. If a sentence embedding is of high quality, it will be easy to define a corresponding partitioning of the embedding space. One way to determine how easy it is to define the corresponding partitioning is to attempt to do just that as a supervised classification task using a weak classifier. A weak classifier, namely a linear support vector machine (SVM)), was used as a more powerful classifier could learn arbitrary transforms. The

classification task is to take in a sentence embedding and predict which group of paraphrases it belongs to. The target paraphrase group is defined using other paraphrases with the same meaning as the candidate.

Under this course of evaluation it was found that the sum and mean of word embeddings performed very well as a sentence representation. These LCOWEs were the best performing models under evaluation. They were closely followed by the bag of words, which has the advantage of being of much higher dimensionality than the other models. The LCOWEs outperform the bag of words as they also capture synonyms and other features of lexical relatedness. Slightly worse than the bag of words was the bag of words with PCA dimensionality reduction to 300 dimensions. This confirms our expectation that LCOWEs are a better form of dimensionality reduction for preserving meaning from a bag of words than PCA.

The poor results of the paragraph vector models (Le and Mikolov 2014) is in line with the observation in the footnotes of the less well-known follow up work of Mesnil et al. (2014). Where it was found that the performance reported in Le and Mikolov (2014) cannot be reliably repeated on other tasks, or even on the same tasks with a slightly different implementation.

A limitation of our investigation is that it does not include the examination of any encoder-decoder based methods, such as Skip-Thought (Kiros et al. 2015), or machine translation models. Another limitation of the work is that the unfolding recursive autoencoder (Socher et al. 2011a) evaluation used a pretrained model with only 200 dimensions, rather than 300 dimensions as was used in the other evaluations.

The **key contribution** of this work was to evaluate the properties of sentence representations using an abstract task. This is in-contrast to most prior evaluations, which use less abstract real-world tasks. While real world tasks have their own important value, it is harder to judge the generalisation ability from such cases. For example, a sentence representation that works well for sentiment analysis may not work well for other NLP tasks. The paraphrase space partitioning task is much more abstract and considers the geometric nature of the representation. We thus expect that as an abstract task it would be more informative as a probing evaluation. This idea of using an abstract probing task to evaluate sentence representations has been significantly advanced and generalised to a battery of such tasks in later works such as Adi et al. (2017) and Conneau et al. (2018). The interesting finding in our work, which significantly contributed to the direction of this dissertation, was that the LCOWEs (SOWE/MOWE) were notably the best performing models. They performed very well on the task to separate meaning. Different word content, particularly with lexical similarity features, effectively gives a much stronger separability of the meaning space than any of the more complex methods considered.

Paraphrases provide one source of grounding for evaluation of sentences. Color names are a subset of short phrases which also have a ground truth for meaning – the intended color. They are thus useful for evaluating the performance of LCOWE on short phrases.

Part II: Chapter 5. "Learning of Colors from Color Names: Distribution and Point Estimation"

Originally published as: Lyndon. White, Roberto. Togneri, Wei. Liu, and Mohammed Bennamoun (2018). "Learning of Colors from Color Names: Distribution and Point Estimation". In: *Computational Linguistics (Under Review).*

To evaluate the performance of input representations for short phrases, we considered a color understanding task. Color understanding is considered a grounded microcosm of natural language understanding (Monroe, Goodman, and Potts 2016). It appears as a complicated sub-domain, with many of the same issues that plague natural language understanding in general: it features a lot of ambiguity, substantial morphological and syntax structure, and depends significantly on context that is not made available to the natural language understanding algorithms. Unlike natural language more generally, it has a comparatively small vocabulary, and it has grounded meaning. The meaning of a particular utterance, say `bluish green`, can be grounded to a point in color space, say in HSV (192°, 93%, 72%), based on questioning the speaker. The general meaning of a color phrase can be grounded to a distribution over the color space, based on surveying the population of speakers.

Models were thus created to learn a mapping from the natural language space, to points or distributions in the color space. Three input representations were considered: a sum of word embeddings (SOWE), a convolutional neural network (CNN), and a recurrent neural network (RNN). The SOWE corresponds to a bag of words – no knowledge of order. The CNN corresponds to a bag of ngrams – it includes features of all length, thus can encode order. The RNN is a fully sequential model – all inputs are processed in order and it must remember previous inputs.

It was expected that this task would benefit significantly from a knowledge of word order. For example, `bluish green` and `greenish blue` are visibly different colors. The former being greener than the later. However, it was found that the SOWE was the best performing input representation, followed closely by the CNN , with the RNN performing much worse. This was even the case when the test set was restricted to only contain color names for which multiple different word orders (representing different colors) were found in the training set. This can be attributed to the difficulty in training the more complicated models. In contrast to a simple feed-forward SOWE, in a RNN the gradient must propagate further from the output, and there are more weights to be learned in the gates. This difficulty dominated over the limitation in being able to model the color names correctly. We note that while `bluish green` and `greenish blue` are different colors, they are still very similar colors. As such, the error from treating them as the same, is less than the error caused by training difficulties.

Estimating colors from their natural language color names has pragmatic uses. Color estimation from description is useful as a tool for improving human-computer interaction. For example allowing free(-er) text for specifying colors in plotting software, using point estimation. It is also useful in education: people from different cultures, especially non-native

English speakers, may not know exactly what color range is described by `dark salmon`. Our model allows for tools to be created to answer such queries using distribution estimation.

A limitation of this study is in the metrics used. For distribution estimation, the perplexity of the discretized distributions in color space is reported. It would be preferable to use Kullback–Leibler divergence, which would allow comparisons to future works that output truly continuous distributions. Kullback–Leibler divergence is monotonically related the to discretized perplexity, however. For point estimation, it would be preferable to also report an evaluation metric, such as a Delta-E, which is controlled for the varying sensitivity of human perception for different hues. Neither limitation has direct bearing on the assessment of the input representations; which is the assessment of primary interest in the context of this dissertation.

The **key contribution** of this work was to evaluate the properties of short phrase representations using a grounded task of color understanding. Secondary contributions include creating a neural network based method for color distribution estimation, which itself has practical use as a teaching tool and in human-computer interaction; and demonstrating a novel method for point estimation of angular data, such as HSV colors. Again, we found surprisingly that SOWE was the most effective input representation.

Part II: ??. "Finding Word Sense Embeddings Of Known Meaning"

Originally published as: Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "Finding Word Sense Embeddings Of Known Meaning". In: *19th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

With the demonstrated utility of linear combinations of embeddings for representing the meanings of larger structures made from words, it is worth investigating their utility for representing the possible different meanings of words. When it comes to representing word senses, it may be desirable to find a representation for the exact sense of a word being used in a particular example. A very fine grained word sense for just that one use. If one has a collection of induced word senses, it seems reasonable to believe that the ideal word sense for a particular use, would lie somewhere between them in the embedding space. Furthermore, if one knows the probability of each of the coarse induced senses being the correct sense for this use, then it is reasonable to assume that the location of the fine grained sense embedding would be closer to the more likely coarse sense, and further from the less likely coarse sense. As such we propose a method to define these specific case word senses based on a probability weighted sum of coarser word sense embeddings. We say that we *refit* the original sense embeddings, using the single example sentence to induce the fine grained sense embedding.

Using this we define a similarity measure which we call RefittedSim, which we find to work better than AvgSimC (Reisinger and Mooney 2010). AvgSimC is a probability-weighted average of all the pairwise

similarity scores for each sense embedding. In contrast RefittedSim is a single similarity score as measured between the two refitted vectors – which are the probability weighted averages of the coarser sense vectors. On the embeddings used in our evaluations this gave a solid improvement over AvgSimC. It is also asymptotically faster to evaluate.

We also evaluated the use of refitting for word sense disambiguation (WSD). Normally, induced senses cannot be used for word sense disambiguation, as they do not correspond to standard dictionary word senses. By using the WordNet gloss (definition) as an example sentence, we are able to use refitting to create a new set of sense embeddings suitable for WSD. Using these new word sense embeddings we can use the skip-gram formulation for probability of the context given the refitted sense, and so apply Bayes' theorem to find the most-likely sense. However, we found that the results were only marginally better than the most frequent sense baseline. Though it was notably better than the results of the method presented by Agirre et al. (2006); which, to the best of our knowledge, is the only prior method for leveraging induced senses for WSD with only a limited number of examples. Nearly unsupervised WSD is a very difficult problem; with a strong baseline of simply reporting the most-common sense. Our results do suggest that our refitting method does not learn features that are antithetical to its use WSD. However, they do incorporate the most frequent sense as a prior and seem to provide little benefit beyond that.

A limitation of this study is that the evaluations were not performed on refitting state-of-the-art word-sense embeddings; rather it only evaluated on AdaGram (Bartunov et al. 2015), and a bespoke greedy baseline method. As such, while its comparisons between these embeddings using different algorithms are valid, they cannot be readily compared to the current state-of-the-art on the tasks when using better base embedding methods.

The **key contribution** of this work was to define a method for specializing word sense embeddings for a single use case. In doing so, an important property of embeddings from skip-gram like formulations was demonstrated. We showed that a good representation can be found by linearly interpolating between less ideal representations according to how likely they are to be correct. Important secondary contributions include the method for smoothing the probability of correctness; and RefittedSim, a new similarity measure using this refitting to evaluate the similarity of words in context.

Part II: ??. "NovelPerspective: Identifying Point of View Characters"

Originally published as: Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "NovelPerspective: Identifying Point of View Characters". In: *Proceedings of ACL 2018, System Demonstrations*. Association for Computational Linguistics.

Given the success of LCOWEs for representing meaningful linguistic structures (sentences and phrases), a natural follow up question is on their capacity to represent combinations of words that do not feature

this natural kind of structure. These would be more arbitrary bags of words; that never-the-less may be useful features for a particular task. The task investigated in this work was about identifying the point of view characters in a novel.

Given some literary text written in third person limited point of view, such as Robert Jordan's popular *"Wheel of Time"* series of novels, it is useful to a reader (or person analysing the text), to identify which sections are from the perspective of which character. That is to say, we would like to classify the chapters of a book according to which character's point of view it is told from. This at first looks like a multiclass classification problem; however it is in-fact an information extraction problem. The set of possible classes for any given chapter is the set of all named entities in the book. Different books have different characters, thus the set of named entities in the training data will not match that of an arbitrary book selected by a user. As such, the named entity tokens themselves cannot be used in training for this task. Instead, it must be determined whether or not a named entity is the point of view character, based on how the named entity token is used. To do this, a representation of the context of use is needed.

The task can be treated as a binary classification problem. Given some feature vector representing how a particular named entity token was used throughout a chapter, we attempt to find the probability of that named entity being the point of view character. We considered two possible feature sets to use to generate the feature vectors for named entity token use. Both feature sets consider the context primarily in terms of the token (word) immediately prior to, and the token (word) immediately after the named entity. We define a 200 dimensional hand-crafted *classical feature set* in terms of the counts of adjacent part of speech tags, position in the text, and token frequency. We define a *mean of word embedding based feature set* as the concatenation of the mean of the word embedding for the words occurring immediately prior, to the mean of the word occurring immediately after. As this was using 300 dimensional embeddings, this gives a 600 dimensional feature vector.

It was found that the two feature sets performed similarly, with both working very well. It seems like the primary difficulty was with the high dimensionality of the word embedding based feature set. Without sufficient training data, it over-fits easily. Its performance dropped sharply on the test set, compared to its oracle performance if trained on the test set, when the largest book series was removed. This likely could have been ameliorated by using lower dimensional embeddings.

The good performance of the word embedding based feature set is surprising here, as it does not include any frequency information. We used a mean, rather than a sum, of word embeddings to represent the context of named entity token use. In the classical feature set, we found that by far the most important feature was how often that named entity token was used. Indeed just reporting the most frequently mentioned named entity gave a very strong baseline. The lexical information captured by the MOWE is clearly similarly useful to the part of speech tag counts, and almost certainly makes more fine grained information available to the classifier. Thus allowing it to define good decisions boundaries if the

feature vector represents a point of view character or not.

A limitation of this study is that different binary classifiers were used for the two feature sets. Ideally, the performance using a range of classifiers for both would have been reported. Our preliminary results, not including in the study, suggested that the classifier choice was not particularly important. With logistic regression, SVM, and decision trees giving similarly high results for both feature sets.

The **key contribution** of this work was to produce a system to identify the point of view characters from the context of the named entity tokens being used. In doing so it was demonstrated that a MOWE can perform similarly well to a hand-engineered feature set. The system produced was deployed, and is openly available for public use at `https://white.ucc.asn.au/tools/np`.

Part II: ??. "Generating Bags of Words from the Sums of their Word Embeddings"

Originally published as: Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016a). "Generating Bags of Words from the Sums of their Word Embeddings". In: *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

Given the consideration of a sum of word embeddings as a dimensionality reduced form of a bag of words (BOW), an important question to ask is how well is the bag of words recoverable from the sum. A practical way to find a lower-bound on the loss of information is to demonstrate a deterministic method that can recover a portion of the bag of words.

We propose a method to extract the original bag of words from a sum of word embeddings. Thus placing a bound on the information loss during the transformation of BOW to SOWE. This is done via a simple greedy algorithm with a correction step. The core of this method functions by iteratively searching the vocabulary of word embeddings for the nearest embedding to the sum, adding its word to the bag of words and subtracting its embedding from the sum. It is thus only computationally viable with reasonably small vocabularies. This method works as each component word in the sum has a unique directional contribution in the high dimensional space. As one would expect, this works better for higher dimensional embeddings, and for BOW with fewer words. Even with relatively low dimensions it works quite well. This shows that embeddings are not for example constantly cancelling each other in the sum.

We do note that the method would not work as well on a MOWE – unless the number of words in the BOW was known in advance. In a MOWE the magnitude of each word embedding is effectively normalized so that the magnitude of the sum is the invariant to the number of words. This normalisation does not affect the direction, and effects all magnitudes proportionally, thus it would not prevent the greedy search from finding the nearest word vector to the sum. The step to subtract the found embedding from the sum cannot be performed without knowing the number of words in the BOW as this determined the weighting on the embeddings. However, the key properties of the summed embeddings

not interfering (or cancelling out), do still hold for the MOWE, since it is just a scaled SOWE.

An interesting alternative to this deterministic method would be to train a supervised model to project from SOWE to a fuzzy bag of words. This is similar to the word-content task considered by Adi et al. (2017). In that task a binary classifier was trained to take a sentence representation and a word embedding for a single word that may or may not appear in the sentence.

The **key contribution** of this work is a system which (lossily) converts from a SOWE to the BOW which defined it. In doing so it was demonstrated that one can largely recover the bag of words from the sum of word embeddings, thus showing that word content information was effectively maintained.

Part II: ??. "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem"

Originally published as: Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016b). "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem". In: *IEEE International Conference on Data Mining: High Dimensional Data Mining Workshop (ICDM: HDM)*. DOI: 10.1109/ICDMW.2016.0113.

Given that it was demonstrated that the bag of word can be recovered, the obvious follow up question is if we can recover the sentence.

Given a bag of words, a trigram language model is employed to determine the most-likely order for words. This allows bags of words to be turned into the most likely sentences. We define a deterministic algorithm to solve this using linear mixed integer programming. Using this algorithm we can use the partially recovered bags of words from **??** and determine how frequently they can be correctly ordered to find the original sentence.

We find that surprisingly often they can. The majority of sentences of length up to 18 can be successfully recovered from a SOWE. Although, the longer the sentence the more difficult the recovery; we do note that most sentences are short. This suggests that the number of likely possible orderings for the words in an arbitrary sentence is much lower than it may at first seem. Particularly, since this method does so well even though it is based on a simple trigram language model. There is no doubt that a more sophisticated language model would perform even better.

The algorithm used in our method is a minor extension of that of Horvat and Byrne (2014). We take advantage of the slight differences between the word ordering problem and the generalised asymmetric travelling salesman problem. We can eliminate some branches that would not be possible for a travelling salesman solver; by directly defining it as a mixed integer linear programming problem.

The **key contribution** of this work is a system to order bags of words recovered from the sums of word embeddings into the most likely sentences. The capacity to do this and match the original sentence order places a

lower-bound on how well sentences can be represented. If a correctly sentence can be fully recovered from a sum of word embeddings using just a language model; then a SOWE is effectively sending sentences to unique areas of the representation space. The use of the methods of **??** and **??**, together with a system trained to output an approximation to a SOWE, is an interesting, though not necessarily practical, method for natural language generation.

Part III Appendix Tooling

Beyond the main content of this dissertation, included is an appendix detailing software contributions. These tools do not directly contribute towards the main content of this thesis. However, they were created as a result of of this research; and have facilitated several of the experiments involved. They are presented in the form of short software papers. The detail collaborations on improving the Julia (Bezanson et al. 2014) data-science ecosystem, in particular in the area of reproducibility and machine learning.

1.2 Concluding Remarks on Semantic Space Capturing in Natural Language Understanding

We can consider that there is a true semantic space of ideas: a meaning space. When speaking, this space is projected down to a natural languages space, which we represent using an embedding in the representation space, with the hope that this representation can be related to the meaning space. This is shown in the diagram in Figure 1.1.

To quote Webster (1900): "A sentence is a group of words expressing a complete thought.", it is not a complete thought, only the *expression* of one. This projection from idea (the meaning space) to utterance (the natural language space) is imperfect – it is lossy. Many ideas are expressed the same way, and language thus has a lot of ambiguity. When we try to understand the meaning of a natural language utterance we are trying to find the point in the meaning space that the speaker intends. Some times the natural language space alone is enough to recover a good idea of the point in the meaning space the speaker intends, but other times it is not.

The preimage[3] of a point in the natural language space (e.g. a sentence), is a probability distribution over the meaning space that could have lead to that utterance, $P(meaning \mid utterance)$. This distribution could be combined with other factors (in a Bayesian way); either from that natural language context, or the environment more broadly. For example, to use a meaning that centres around word sense: we can identify two (of the many) senses of the word `apples`: one in reference to the fruit, the other in reference to the computers. Thus, on its own the sentence `Apples are good.` suggests a distribution with at least two peaks in the meaning space. Combine that utterance, with the context of being in a computer store, rather than a grocer, and the probability of one of the two

[3] We say preimage in an abuse of mathematical terminology.

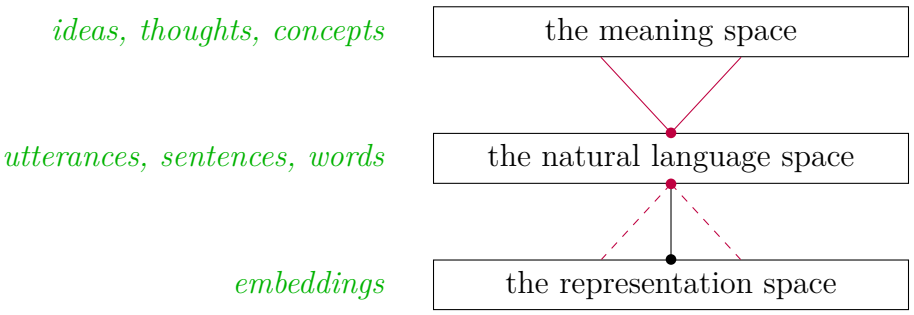


Figure 1.1: The representation space is a computationally manipulate representation of the meaning space. The natural language utterances come from points in the meaning space; though due to ambiguity we can only truly hope to estimate distributions over meanings when interpreting them. A single point embedding is an approximation to a distribution with a single tight peak.

peaks can be increased, though the other not entirely removed. Further around each peak remains adjacent closely related possible meanings. For example the statement could be in relation to only computers, or also to other products. The meaning space is a continuous space, with every thought corresponding to a unique point. It is uncountably large. In contrast, the natural language space is countably large, being composed of finite length combinations of symbols taken from a finite alphabet. An uncountable number of points in the meaning space are projected down to a single point in a natural language space when a thought is put to words.

An embedding space is a particular instance of a representation space, much like the English language is a particular instance of a natural language space. When designing an embedding method (for sentences, words or other structures), we seek to define a representation space that has good properties for reflection relationships in the meaning space in a way that is computationally manipulatable using simple operations (like sums). In particular, it should have a continuous mapping to and from the meaning space. A neighbourhood in the representation space, should correspond to a neighbourhood in the meaning space. **??** investigates this for sentence embeddings. This is done by taking points in the natural language space known to come from very nearby points in the meaning space, that is to say paraphrases, and then checking that they belong to nearby points in the embedding space.

As each point in the natural language space defines a distribution over the meaning space of what may be meant; and the representation space is attempting to be in correspondence to the meaning space; it is such that each point in the natural language space should project to a distribution over the embedding space. Instead, most methods project each natural language point to a single point in the embeddings space. This is viable when the region in the meaning space that the utterance (natural language point) could have come from is small – in particular when the distribution in the meaning space has is of narrow variance and is monomodal. In that case the single point estimate in the embedding space is a useful approximation.

This has a particularly clear utility for word sense embeddings, which are defined by multimodal distributions, with large peaks for each homonym,

and smaller nearby peaks for each polyseme. Furthermore we cannot rule out the speaker using the word incorrectly or metaphorically, which gives rise to nonzero values elsewhere in the meaning space. Word-sense embeddings produce multiple sense embeddings – ideally one corresponding to each peak in the meaning space. We know that these peaks are only rough approximations to the true point in the meaning space for a given usage of a word. **??** attempts to find other points in the embeddings space, that better corresponds to the true point in the meaning space for the particular use. These will be near those peaks given by the point estimates from the senses found using word sense induction. The refitting method (discussed in **??**) efficiently interpolates a point between those peaks based on likelihood.

Unsupervised methods, in particular word embeddings (though it applies also more generally), are ungrounded. They are based only on the natural language space observations. The goal is not to capture meaning in this space, but rather to create a space that is a good input to a supervised system that can learn a good correspondence from the natural language space to the meaning space. While one would not normally think of the SOWE sentence representation space as one for which there would be an easy alignment to the meaning space, **??** shows that it is. A strong point in its favour is that it directly benefits from word embeddings. While themselves ungrounded, word embeddings are excellently suited for creating a representation space, as they have an internal consistency which makes it easy to apply supervision to give grounded meaning representations. Its great strength comes from Firth's distributional hypothesis, that words occurring in similar contexts have similar meaning. While this does not allow the encoding of meaning itself, it does allow the encoding of similarity of meaning. This is ideally suited for creating a space that will make a good source representation for a supervised method applied for natural language understanding task on words. Were that task accomplished with a neural network, the later hidden layers, or the fine-turned embeddings would form a grounded representation of the meaning space. Our results show that that strength is carried forth into linear combinations of such embeddings.

The color understanding task considered in Chapter 5 is interesting. It is a typical natural language understanding system, which takes a point in a natural language space (a color name), moves through a representation space (the output of one of the input modules: SOWE, CNN, or RNN) using supervision to output something from a meaning space. Notably however, the meaning space is *very well grounded* to the HSV color space. We can, for many purposes, say for this natural language understanding task, the color space *is* the meaning space. Using point estimation it outputs a point in the meaning space, reflecting (in some sense) the most reasonable guess of the meaning. Using distribution estimation it outputs a distribution over the meaning space, fully reflecting the knowledge we have to infer the meaning. An important idea is highlighted by the fact that even on the subset of the testing data where word order was ambiguous, SOWE was the best performing model. Word order ambiguity is just one amongst many sources of ambiguity in any representation of natural language. In the color case, it boils down to the additional ambiguity of being unable to encode the word order difference between

`bluish green` and `greenish blue` being negligible compared to the inherent ambiguity in the meaning of either. Both phrases give rise to a large and overlapping distribution across the meaning space.

In cases where there are multiple reasonable word orderings, this means that multiple points in the true meaning space, correspond to a single point in the representational LCOWE space. However, this is not exceptional: many sentences have two or more interpretations, a humorous example being an accidental pun. Thus even in a representation space that fully captures the natural language features, a single point in that representation space corresponds to two points in the meaning space; as the single point in the natural language space could have come from either point in the meaning space. As such, the ambiguity from loss of word order is not a unique and unsalvageable problem. If we thus had a distribution over the meaning space, corresponding to the interpretation of a SOWE, it would have two peaks corresponding to two different word orders. While such a discussion is purely theoretical as we do not have any way to generate such a distribution over the true meaning space, it remains interesting for cases where we have a space that we can treat as being the meaning space (e.g. the HSV space for colors). As we can use other contextual information to define a prior and thus decrease distributions associated with other ambiguities, we can use language models to provide a prior over those peaks; based on the likelihood of word orders. There exists a trivial extension of the work presented in **????**, where the mixed integer programming model is constrained to give the second (and so forth) most likely solution, together with its probability. However, it is not computationally practical, nor useful without a better meaning space representation.

While the research presented in this dissertation has made use of the idea that we are working with a sample from a distribution over a proxy for the meaning space, it is our belief that further advancements would benefit from fully considering word embeddings and other objects from the representation spaces, not as discrete points but as random variables with a linked distribution. This, however, comes with significant challenges as working with the high dimensional distributions that would be required is computationally difficult.

?? and Chapter 5 both consider representing contiguous linguistic structures, in particular sentences and shore phrases. **??** directly explores the ability to find regions of the representation space that match to regions of the meaning space. Further, the output of the input modules discussed in Chapter 5 are (once trained) points in a grounded representation space, though that work did not examine it directly. **??** considers the representation of word senses, and it navigates the representation space to find new representations which better describe a particular use of a word. **??** is more atypical: the need is to represent how a particular named entity token was used throughout a chapter. Which is not a representation of contiguous linguistic structure, but never-the-less is a representation problem for natural language understanding. In all these problems, it seems like linear combinations of embeddings would not suffice for our representational needs. Yet, we find in each case that it is a surprisingly practical representation that should not be discarded out-of-hand. It effectively meets many of our goals for a good representation space.

Part I

Literature Review

Chapter 2

Word Representations

This chapter originally appeared as Chapter 3 of the book "Neural Representations of Natural Language", published by by Springer.The full book can be made available to the examiners on request. This chapter is not to be distributed to the general public and will not appear in the public online archival copy of the thesis.

You shall know a word by the company it keeps.

– J.R. Firth, 1957

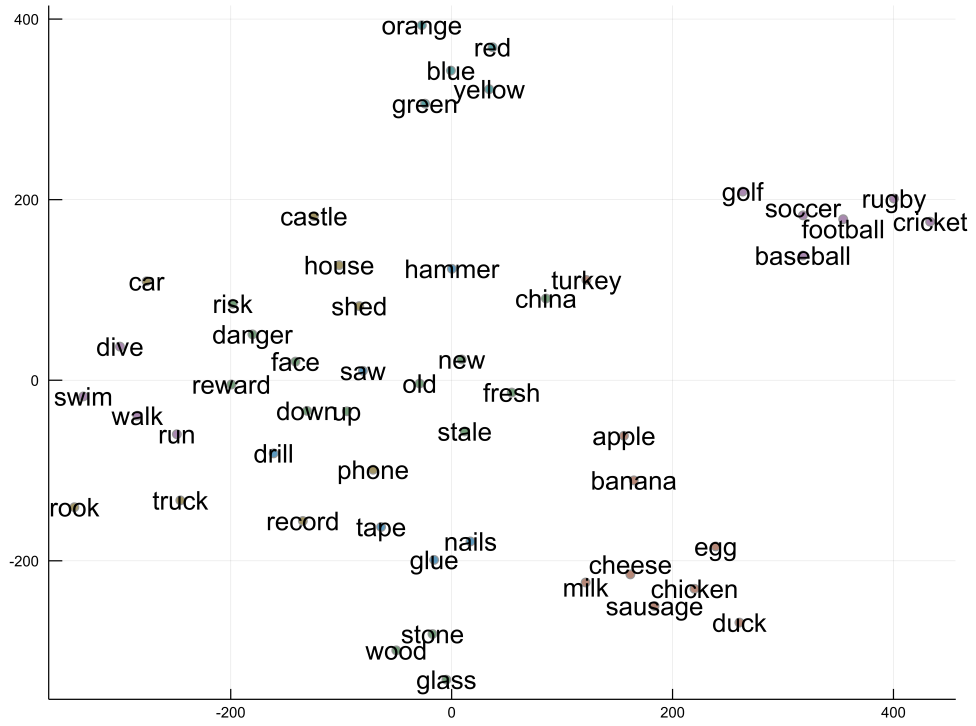
Abstract

Word embeddings are the core innovation that has brought machine learning to the forefront of natural language processing. This chapter discusses how one can create a numerical vector that captures the salient features (e.g. semantic meaning) of a word. Discussion begins with the classic language modelling problem. By solving this, using a neural network-based approach, word-embeddings are created. Techniques such as CBOW and skip-gram models (`word2vec`), and more recent advances in relating this to common linear algebraic reductions on co-locations as discussed. The chapter also includes a detailed discussion of the often confusing hierarchical softmax, and negative sampling techniques. It concludes with a brief look at some other applications and related techniques.

We begin the consideration of the representation of words using neural networks with the work on language modeling. This is not the only place one could begin the consideration: the information retrieval models, such as LSI (Dumais et al. 1988) and LDA (Blei, Ng, and Jordan 2003), based on word co-location with documents would be the other obvious starting point. However, these models are closer to the end point, than they are to the beginning, both chronologically, and in this chapter's layout. From the language modeling work, comes the contextual (or acausal) language model works such as skip-gram, which in turn lead to the post-neural network co-occurrence based works. These co-occurrence works are more similar to the information retrieval co-location based methods than the probabilistic language modeling methods for word embeddings from which we begin this discussion.

Word embeddings are vector representations of words. An dimensionality reduced scatter plot example of some word embeddings is shown in Figure 2.1.

Figure 2.1: Some word embeddings from the FastText project (Bojanowski et al. 2017). They were originally 300 dimensions but have been reduced to 2 using t-SNE (Maaten and Hinton 2008) algorithm. The colors are from 5 manually annotated categories done before this visualisation was produced: `foods`, `sports`, `colors`, `tools`, `other objects`, `other`. Note that many of these words have multiple meanings (see Chapter 3), and could fit into multiple categories. Also notice that the information captioned by the unsupervised word embeddings is far finer grained than the manual categorisation. Notice, for example, the separation of ball-sports, from words like `run` and `walk`. Not also that `china` and `turkey` are together; this no doubt represents that they are both also countries.



2.1 Representations for Language Modeling

The language modeling task is to predict the next word given the prior words (Rosenfeld 2000). For example, if a sentence begins `For lunch I will have a hot`, then there is a high probability that the next word will be `dog` or `meal`, and lower probabilities of words such as `day` or `are`. Mathematically it is formulated as:

$$P(W^i{=}w^i \mid W^{i-1}{=}w^{i-1}, \ldots, W^1{=}w^1) \tag{2.1}$$

or to use the compact notation

$$P(w^i \mid w^{i-1}, \ldots, w^1) \tag{2.2}$$

where W^i is a random variable for the ith word, and w^i is a value (a word) it could, (or does) take. For example:

$$P(\texttt{dog} \mid \texttt{hot}, \texttt{a}, \texttt{want}, \texttt{I}, \texttt{lunch}, \texttt{For})$$

The task is to find the probabilities for the various words that w^i could represent.

The classical approach is trigram statistical language modeling. In this, the number of occurrences of word triples in a corpus is counted. From this joint probability of triples, one can condition upon the first two words, to get a conditional probability of the third. This makes the Markov assumption that the next state depends only on the current state, and that that state can be described by the previous two words. Under this assumption Equation (2.2) becomes:

$$P(w^i \mid w^{i-1}, \ldots, w^1) = P(w^i \mid w^{i-1}, w^{i-2}) \tag{2.3}$$

More generally, one can use an n-gram language model where for any value of n, this is simply a matter of defining the Markov state to contain different numbers of words.

This Markov assumption is, of-course, an approximation. In the previous example, a trigram language model finds $P(w^i \mid \texttt{hot}, \texttt{a})$. It can be seen that the approximation has lost key information. Based only on the previous 2 words the next word w^i could now reasonably be `day`, but the sentence: `For lunch I will have a hot day` makes no sense. However, the Markov assumption in using n-grams is required in order to make the problem tractable – otherwise an unbounded amount of information would need to be stored.

A key issue with n-gram language models is that there exists a data-sparsity problem which causes issues in training them. Particularly for larger values of n. Most combinations of words occur very rarely (Ha et al. 2009). It is thus hard to estimate their occurrence probability. Combinations of words that do not occur in the corpus are naturally given a probability of zero. This is unlikely to be true though – it is simply a matter of rare phrases never occurring in a finite corpus. Several approaches have been taken to handle this. The simplest is add-one smoothing which adds an extra "fake" observation to every combination of terms. In common use are various back-off methods (Katz 1987; Kneser and Ney 1995) which use the bigram probabilities to estimate the probabilities of unseen trigrams (and so forth for other n-grams.). However, these methods

are merely clever statistical tricks – ways to reassign probability mass to leave some left-over for unseen cases. Back-off is smarter than add-one smoothing, as it portions the probability fairly based on the $(n-1)$-gram probability. Better still would be a method which can learn to see the common-role of words (Brown et al. 1992). By looking at the fragment: `For lunch I want a hot`, any reader knows that the next word is most likely going to be a food. We know this for the same reason we know the next word in `For elevenses I had a cold …` is also going to be a food. Even though `elevenses` is a vary rare word, we know from the context that it is a meal (more on this later), and we know it shares other traits with meals, and similarly `have` / `had`, and `hot` / `cold`. These traits influence the words that can occur after them. Hard-clustering words into groups is nontrivial, particularly given words having multiple meanings, and subtle differences in use. Thus the motivation is for a language modeling method which makes use of these shared properties of the words, but considers them in a flexible soft way. This motivates the need for representations which hold such linguistic information. Such representations must be discoverable from the corpus, as it is beyond reasonable to effectively hard-code suitable feature extractors. This is exactly the kind of task which a neural network achieves implicitly in its internal representations.

2.1.1 The Neural Probabilistic Language Model

Bengio et al. (2003) present a method that uses a neural network to create a language model. In doing so it implicitly learns the crucial traits of words, during training. The core mechanism that allowed this was using an embedding or loop-up layer for the input.

Simplified Model considered with Input Embeddings

To understand the neural probabilistic language model, let's first consider a simplified neural trigram language model. This model is a simplification of the model introduced by Bengio et al. (2003). It follows the same principles, and highlights the most important idea in neural language representations. This is that of training a vector representation of a word using a lookup table to map a discrete scalar word to a continuous-space vector which becomes the first layer of the network.

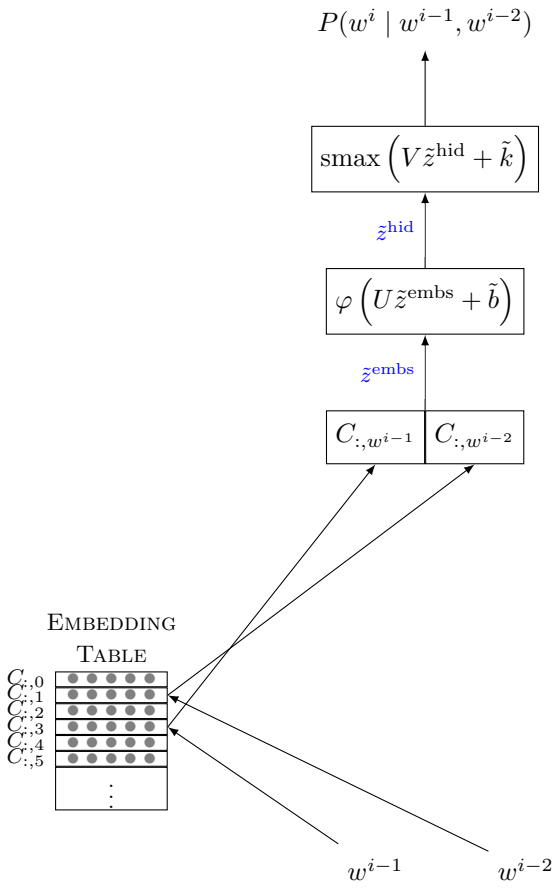
The neural trigram probabilistic network is defined by:

$$P(w^i \mid w^{i-1}, w^{i-2}) = \operatorname{smax}\left(V\varphi\left(U\left[C_{:,w^{i-1}}; C_{:,w^{i-2}}\right] + \tilde{b}\right) + \tilde{k}\right) \quad (2.4)$$

where U, V, $\tilde{b}$, $\tilde{k}$ are the weight matrices and biases of the network. The matrix C defines the embedding table, from which the word embeddings, $C_{:,w^{i-1}}$ and $C_{:,w^{i-2}}$, representing the previous two words (w^{i-1} and w^{i-2}) are retrieved. The network is shown in Figure 2.2

In the neural trigram language model, each of the previous two words is used to look-up a vector from the embedding matrix. These are then concatenated to give a dense, continuous-space input to the above hidden

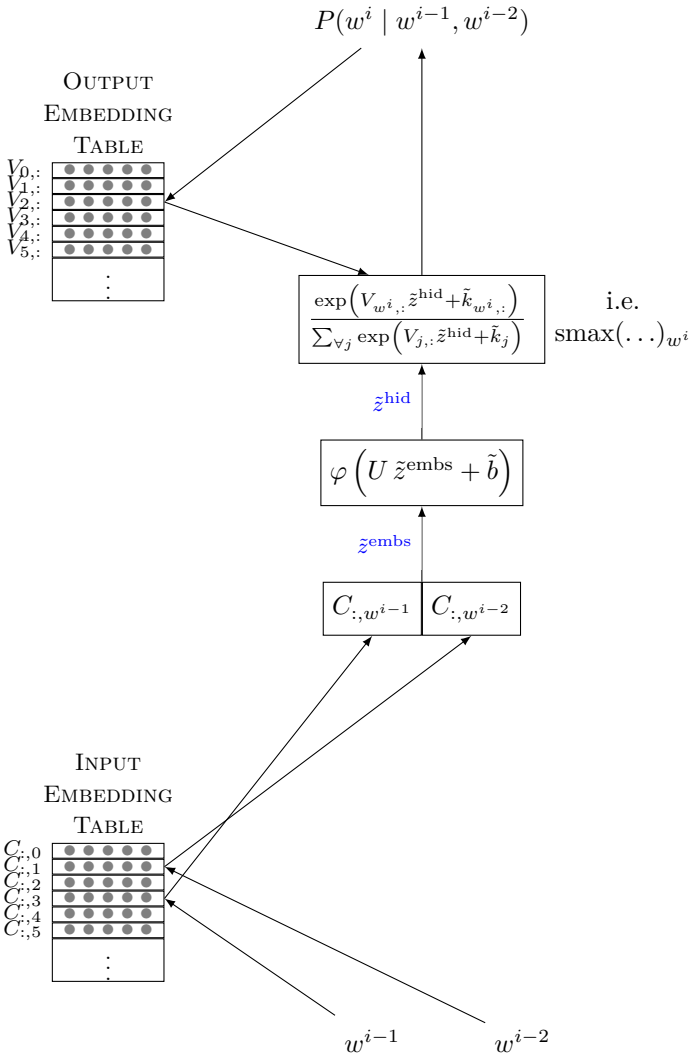
Figure 2.2: The Neural Trigram Language Model



layer. The output layer is a softmax layer, it gives the probabilities for each word in the vocabulary, such that $\hat{y}_{w^i} = P(w^i \mid w^{i-1}, w^{i-2})$. Thus producing a useful language model.

The word embeddings are trained, just like any other parameter of the network (i.e. the other weights and biases) via gradient descent. An effect of this is that the embeddings of words which predict the same future word will be adjusted to be nearer to each other in the vector space. The hidden layer learns to associate information with regions of the embedding space, as the whole network (and every layer) is a continuous function. This effectively allows for information sharing between words. If two word's vectors are close together because they mostly predict the same future words, then that area of the embedding space is associated with predicting those words. If words a and b often occur as the word prior to some similar set of words $(w, x, y, \ldots)$ in the training set and word b also often occurs in the training set before word z, but (by chance) a never does, then this neural language model will predict that z is likely to occur after a. Where-as an n-gram language model would not. This is because a and b have similar embeddings, due to predicting a similar set of words. The model has learnt common features about these words implicitly from how they are used, and can use those to make better predictions. These features are stored in the embeddings which are looked up during the input.

Figure 2.3: Neural Trigram Language Model as considered with output embeddings. This is mathematically identical to Figure 2.2



Simplified Model considered with input and output embeddings

We can actually reinterpret the softmax output layer as also having embeddings. An alternative but equivalent diagram is shown in Figure 2.3.

The final layer of the neural trigram language model can be rewritten per each index corresponding to a possible next word (w^i):

$$\text{smax}(V\tilde{z}^{\text{hid}}+\tilde{k})_{w^i} = \frac{\exp\left(V_{w^i,:}\tilde{z}^{\text{hid}}+\tilde{k}_{w^i}\right)}{\sum_{\forall j}\exp\left(V_{j,:}\tilde{z}^{\text{hid}}+\tilde{k}_j\right)} \tag{2.5}$$

In this formulation, we have $V_{w_i,:}$ as the output embedding for w^i. As we considered $C_{:,w_i}$ as its input embedding.

Bayes-like Reformulation

When we consider the model with output embeddings, it is natural to also consider it under the light of the Bayes-like reformulation from Chapter

1 of White et al. (2018a):

$$P(Y{=}i \quad | \quad Z{=}\tilde{z}) \quad = \quad \frac{R(Z{=}\tilde{z} \mid Y{=}i)\, R(Y{=}i)}{\sum_{\forall j} R(Z{=}\tilde{z} \mid Y{=}j)\, R(Y{=}j)} \tag{2.6}$$

which in this case is:

$$P(w^i \mid w^{i-1}, w^{i-2}) = \frac{R(Z{=}\tilde{z}^{\text{hid}} \mid W^i{=}w^i)\, R(W^i{=}w^i)}{\sum_{\forall v \in \mathbb{V}} R(Z = \tilde{z}^{\text{hid}} \mid W^i{=}v)\, R(W^i{=}v)} \tag{2.7}$$

where $\sum_{\forall v \in \mathbb{V}}$ is summing over every possible word v from the vocabulary $\mathbb{V}$, which does include the case $v = w^i$.

Notice the term:

$$\frac{R(W^i{=}w^i)}{\sum_{\forall v \in \mathbb{V}} R(W^i{=}v)} = \frac{\exp\left(\tilde{k}_{w^i}\right)}{\sum_{\forall v \in \mathbb{V}} \exp\left(\tilde{k}_v\right)} \tag{2.8}$$

$$= \frac{1}{\sum_{\forall v \in \mathbb{V}} \exp\left(\tilde{k}_v - \tilde{k}_{w^i}\right)} \tag{2.9}$$

The term $R(W^i{=}w^i) = \exp\left(\tilde{k}_{w^i}\right)$ is linked to the unigram word probabilities: $P(Y = y)$. If $\mathbb{E}(R(Z \mid W_i) = 1$ then the optimal value for $\tilde{k}$ would be given by the log unigram probabilities: $k_{w^i} = \log P(W^i{=}w^i)$. This condition is equivalent to if $\mathbb{E}(V\tilde{z}^{\text{hid}}) = 0$. Given that V is normally[1] initialized as a zero mean Gaussian, this condition is at least initially true. This suggests, interestingly, that we can predetermine good initial values for the output bias $\tilde{k}$ using the log of the unigram probabilities. In practice this is not required, as it is learnt rapidly by the network during training.

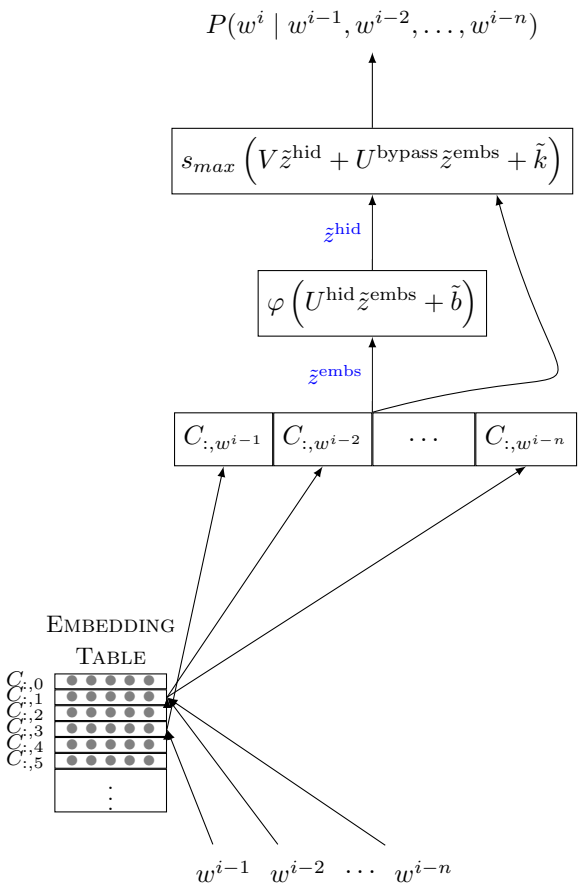
The Neural Probabilistic Language Model

Bengio et al. (2003) derived a more advanced version of the neural language model discussed above. Rather than being a trigram language model, it is an n-gram language model, where n is a hyper-parameter of the model. The knowledge sharing allows the data-sparsity issues to be ameliorated, thus allowing for a larger n than in traditional n-gram language models. Bengio et al. (2003) investigated values for 2, 4 and 5 prior words (i.e. a trigram, 5-gram and 6-gram model). The network used in their work was marginally more complex than the trigram neural language model. As shown in Figure 2.4, it features a layer-bypass connection. For n prior words, the model is described by:

$$\begin{aligned} P(w^i \mid w^{i-1}, \ldots, w^{i-n}) = \text{smax}(& \\ & + V\, \varphi\left(U^{\text{hid}}\left[C_{:,w^{i-1}}; \ldots; C_{:,w^{i-n}}\right] + \tilde{b}\right) \\ & + U^{\text{bypass}}\left[C_{:,w^{i-1}}; \ldots; C_{:,w^{i-n}}\right] \\ & + \tilde{k})_{w^i} \end{aligned} \tag{2.10}$$

[1] no pun intended

Figure 2.4: Neural Probabilistic Language Model



The layer-bypass is a connivance to aid in the learning. It allows the input to directly affect the output without being mediated by the shared hidden layer. This layer-bypass is an unusual feature, not present in future works deriving from this, such as Schwenk (2004). Though in general it is not an unheard of technique in neural network machine learning.

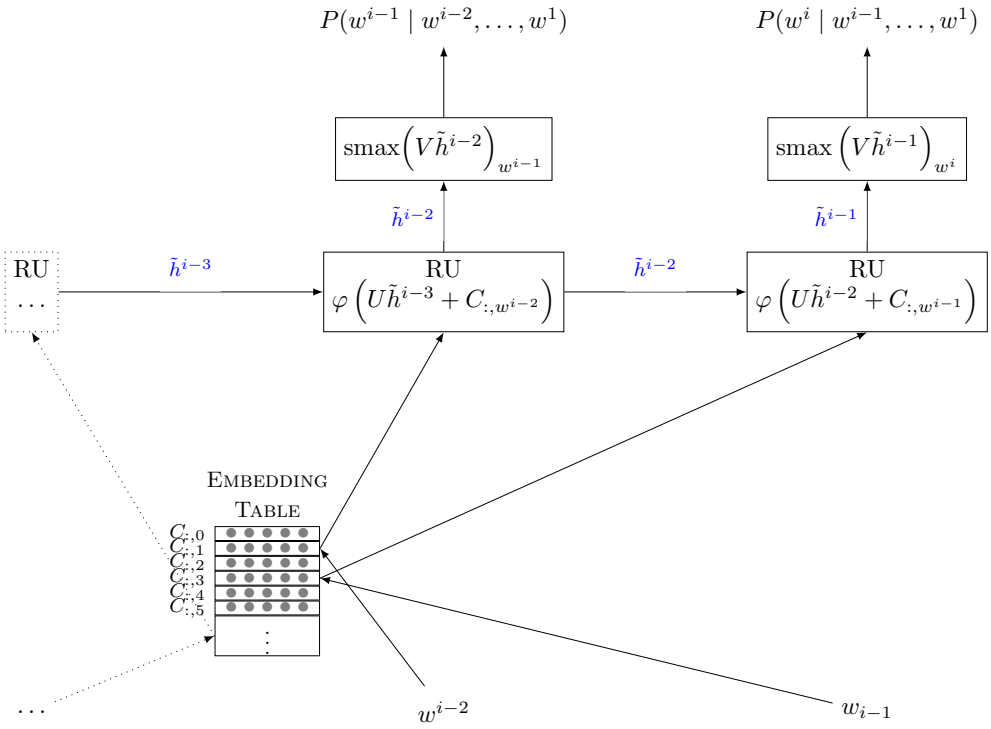
This is the network which begins the notions of using neural networks with vector representations of words. Bengio et al. focused on the use of the of sliding window of previous words – much like the traditional n-grams. At each time-step the window is advanced forward and the next is word predicted based on the shifted context of prior words. This is of-course exactly identical to extracting all n-grams from the corpus and using those as the training data. They very briefly mention that an RNN could be used in its place.

2.1.2 RNN Language Models

In Mikolov et al. (2010) an RNN is used for language modelling, as shown in Figure 2.5. Using the terminology of Chapter 2 of (White et al. 2018a), this is an encoder RNN, made using Basic Recurrent Units. Using an RNN eliminates the Markov assumption of a finite window of prior words forming the state. Instead, the state is learned, and stored in the state component of the RUs.

This state $\tilde{h}_i$ being the hidden state (and output as this is a basic RU)

Figure 2.5: RNN Language Model. The RU equation shown is the basic RU used in Mikolov et al. (2010). It can be substituted for a LSTM RU or an GRU as was done in Sundermeyer, Schlüter, and Ney (2012) and Jozefowicz, Zaremba, and Sutskever (2015), with appropriate changes.



from the i time-step. The ith time-step takes as its input the ith word. As usual this hidden layer was an input to the hidden-layer at the next time-step, as well as to the output softmax.

$$\tilde{h}^i = \varphi\left(U\tilde{h}^{i-1} + C_{:,w_{i-1}}\right) \tag{2.11}$$

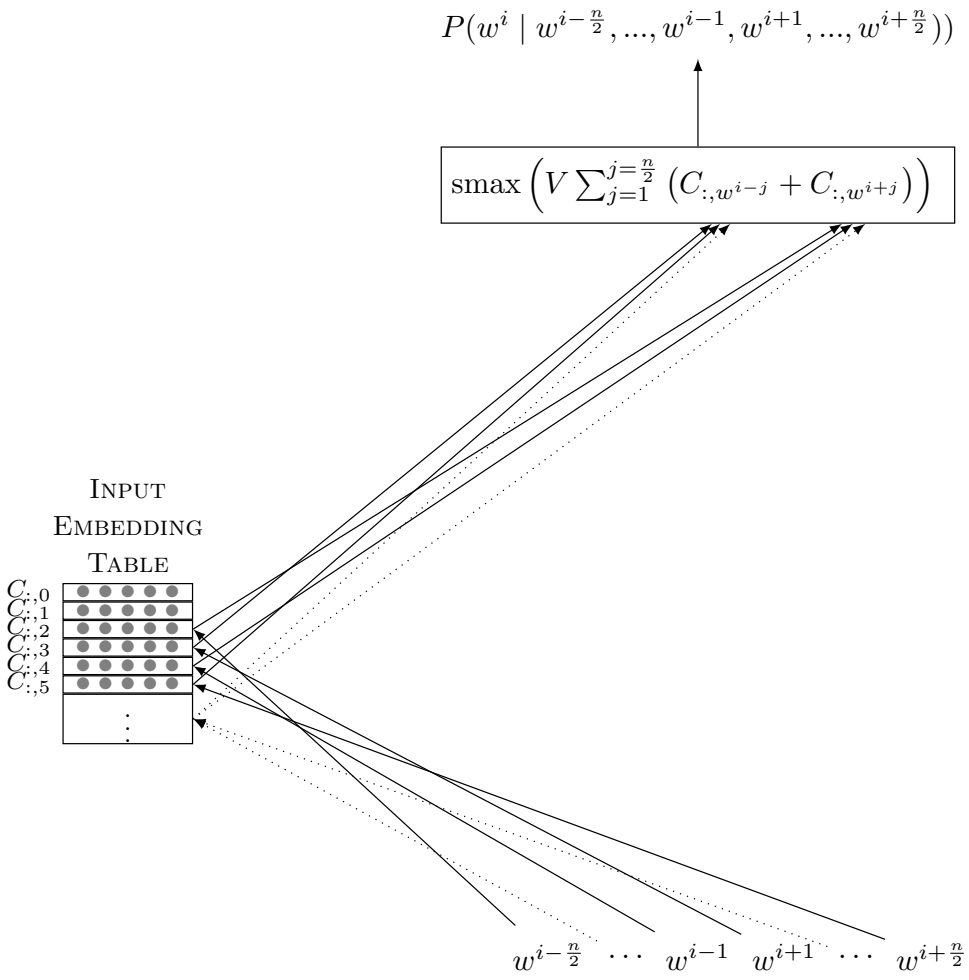
$$P(w^i \mid w^{i-1}, \ldots w^1) = \text{smax}\left(V\tilde{h}^{i-1}\right)_{w^i} \tag{2.12}$$

Rather than using a basic RU, a more advanced RNN such as a LSTM or GRU-based network can be used. This was done by Sundermeyer, Schlüter, and Ney (2012) and Jozefowicz, Zaremba, and Sutskever (2015), both of whom found that the more advanced networks gave significantly better results.

2.2 Acausal Language Modeling

The step beyond a normal language model, which uses the prior words to predict the next word, is what we will term acausal language modelling. Here we use the word acausal in the signal processing sense. It is also sometimes called contextual language modelling, as the whole context is used, not just the prior context. The task here is to predict a missing word, using the words that precede it, as well as the words that come after it.

Figure 2.6: CBOW Language Model



As it is acausal it cannot be implemented in a real-time system, and for many tasks this renders it less, directly, useful than a normal language model. However, it is very useful as a task to learn a good representation for words.

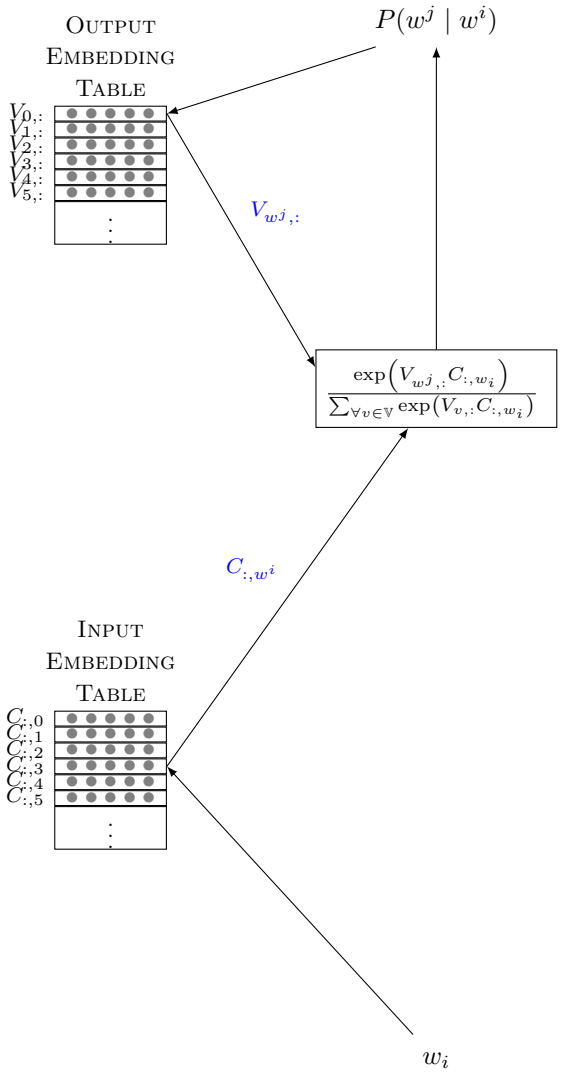
The several of the works discussed in this section also feature hierarchical softmax and negative sampling methods as alternative output methods. As these are complicated and easily misunderstood topics they are discussed in a more tutorial fashion in Section 2.4. This section will focus just on the language model logic; and assume the output is a normal softmax layer.

2.2.1 Continuous Bag of Words

The continuous bag of words (CBOW) method was introduced by Mikolov et al. (2013b). In truth, this is not particularly similar to bag of words at all. No more so than any other word representation that does not have regard for order of the context words (e.g. skip-gram, and GloVe).

The CBOW model takes as its input a context window surrounding a central skipped word, and tries to predict the word that it skipped over. It is very similar to earlier discussed neural language models, except that the window is on both sides. It also does not have any non-linearities;

Figure 2.7: Skip-gram language Language Model. Note that the probability $P(w^j \mid w^i)$ is optimised during training for every w^j in a window around the central word w^i. Note that the final layer in this diagram is just a softmax layer, written in in output embedding form.



and the only hidden layer is the embedding layer.

For a context window of width n words – i.e. $\frac{n}{2}$ words to either side, of the target word w^i, the CBOW model is defined by:

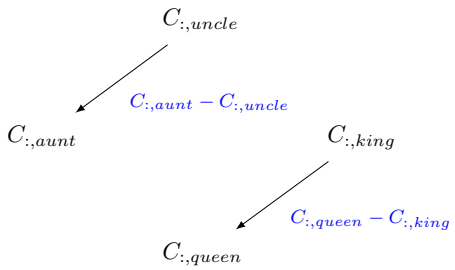
$$P(w^i \mid w^{i-\frac{n}{2}}, \ldots, w^{i-1}, w^{i+1}, \ldots, w^{i+\frac{n}{2}}) = \text{smax}\left(V \sum_{j=i+1}^{j=\frac{n}{2}} \left(C_{:,w^{i-j}} + C_{:,w^{i+j}}\right)\right)_{w^i} \tag{2.13}$$

This is shown in diagrammatic form in Figure 2.6. By optimising across a training dataset, useful word embeddings are found, just like in the normal language model approaches.

2.2.2 Skip-gram

The converse of CBOW is the skip-grams model Mikolov et al. (2013b). In this model, the central word is used to predict the words in the context.

Figure 2.8: Example of analogy algebra



The model itself is single word input, and its output is a softmax for the probability of each word in the vocabulary occurring in the context of the input word. This can be indexed to get the individual probability of a given word occurring as usual for a language model. So for input word w^i the probability of w^j occurring in its context is given by:

$$P(w^j \mid w^i) = \text{smax}\left(V\, C_{:,w^i}\right)_{w^j} \tag{2.14}$$

The goal, is to maximise the probabilities of all the observed outputs that actually *do* occur in its context. This is done, as in CBOW by defining a window for the context of a word in the training corpus, $(i - \frac{n}{2}, \ldots, i-1, i+i, \ldots, i+\frac{n}{2})$. It should be understood that while this is presented similarly to a classification task, there is no expectation that the model will actually predict the correct result, given that even during training there are multiple correct results. It is a regression to an accurate estimate of the probabilities of co-occurrence (this is true for probabilistic language models more generally, but is particularly obvious in the skip-gram case).

Note that in skip-gram, like CBOW, the only hidden layer is the embedding layer. Rewriting Equation (2.14) in output embedding form:

$$P(w^j \mid w^i) = \text{smax}\left(V\, C_{:,w^i}\right)_{w^j} \tag{2.15}$$

$$P(w^j \mid w^i) = \frac{\exp\left(V_{w^j,:} C_{:,w^i}\right)}{\sum_{\forall v \in \mathbb{V}} \exp\left(V_{v,:} C_{:,v}\right)} \tag{2.16}$$

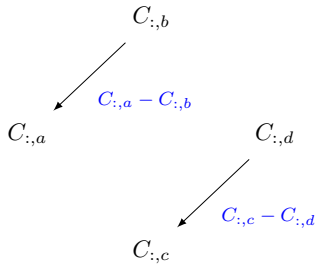
The key term here is the product $V_{w^j,:}\, C_{:,w^i}$. The remainder of Equation (2.16) is to normalise this into a probability. Maximising the probability $P(w^j \mid w^i)$ is equivalent to maximising the dot produce between $V_{w^j,:}$, the output embedding for w^j and $C_{:,w^i}$ the input embedding for w^i. This is to say that the skip-gram probability is maximised when the angular difference between the input embedding for a word, and the output embeddings for its co-occurring words is minimised. The dot-product is a measure of vector similarity – closely related ot the cosine similarity.

Skip-gram is much more commonly used than CBOW.

2.2.3 Analogy Tasks

One of the most notable features of word embeddings is their ability to be used to express analogies using linear algebra. These tasks are keyed

Figure 2.9: Vectors involved in analogy ranking tasks, this may help to understand the math in Equation (2.19)



around answering the question: b is to a, as what is to c? For example, a semantic analogy would be answering that `Aunt` is to `Uncle` as `King` is to `Queen`. A syntactic analogy would be answering that `King` is to `Kings` as `Queen` is to `Queens`. The latest and largest analogy test set is presented by Gladkova, Drozd, and Matsuoka (2016), which evaluates embeddings on 40 subcategories of knowledge. Analogy completion is not a practical task, but rather serves to illustrate the kinds of information being captured, and the way in which it is represented (in this case linearly).

The analogies work by relating similarities of differences between the word vectors. When evaluating word similarity using using word embeddings a number of measures can be employed. By far the cosine similarity is the most common. This is given by

$$\text{sim}(\tilde{u}, \tilde{v}) = \frac{\tilde{u} \cdot \tilde{v}}{\|\tilde{u}\| \, \|\tilde{v}\|} \tag{2.17}$$

This value becomes higher the closer the word embedding $\tilde{u}$ and $\tilde{v}$ are to each other, ignoring vector magnitude. For word embeddings that are working well, then words with closer embeddings should have correspondingly greater similarity. This similarity could be syntactic, semantic or other. The analogy tasks can help identify what kinds of similarities the embeddings are capturing.

Using the similarity scores, a ranking of words to complete the analogy is found. To find the correct word for d in: d is to c as b is to a the following is computed using the table of embeddings C over the vocabulary $\mathbb{V}$:

$$\underset{\forall d \in \mathbb{V}}{\text{argmax}} \, \text{sim}(C_{:,d} - C_{:,c}, C_{:,a} - C_{:,b}) \tag{2.18}$$

$$\text{i.e} \underset{\forall d \in \mathbb{V}}{\text{argmax}} \, \text{sim}(C_{:,d}, \, C_{:,a} - C_{:,b} + C_{:,c}) \tag{2.19}$$

This is shown diagrammaticality in Figures 2.8 and 2.9. Sets of embeddings where the vector displacement between analogy terms are more consistent score better.

Initial results in Mikolov, Yih, and Zweig (2013) were relatively poor, but the surprising finding was that this worked at all. Mikolov et al. (2013b) found that CBOW performed poorly for semantic tasks, but comparatively well for syntactic tasks; skip-gram performed comparatively well for both, though not quite as good in the syntactic tasks as CBOW. Subsequent results found in Pennington, Socher, and Manning (2014) were significantly better again for both.

2.3 Co-location Factorisation

2.3.1 GloVe

Skip-gram, like all probabilistic language models, is a intrinsically prediction-based method. It is effectively optimising a neutral network to predict which words will co-occur in the with in the range of given by the context window width. That optimisation is carried out per-context window, that is to say the network is updated based on the local co-occurrences. In Pennington, Socher, and Manning (2014) the authors show that if one were to change that optimisation to be global over all co-occurrences, then the optimisation criteria becomes minimising the cross-entropy between the true co-occurrence probabilities, and the value of the embedding product, with the cross entropy measure being weighted by the frequency of the occurrence of the word. That is to say if skip-gram were optimised globally it would be equivalent to minimising:

$$Loss = -\sum_{\forall w^i \in \mathbb{V}} \sum_{\forall w^j \in \mathbb{V}} X_{w^i,w^j} P(w^j \mid w^i) \log(V_{w^j,:} C_{:,w^i}) \tag{2.20}$$

for $\mathbb{V}$ being the vocabulary and for X being the a matrix of the true co-occurrence counts, (such that X_{w^i,w^j} is the number of times words w^i and w^j co-occur), and for P being the predicted probability output by the skip-gram.

Minimising this cross-entropy efficiently means factorising the true co-occurrence matrix X, into the input and output embedding matrices C and V, under a particular set of weightings given by the cross entropy measure.

Pennington, Socher, and Manning (2014) propose an approach based on this idea. For each word co-occurrence of w^i and w^j in the vocabulary: they attempt to find optimal values for the embedding tables C, V and the per word biases $\tilde{b}$, $\tilde{k}$ such that the function $s(w^i, w^j)$ (below) expresses an approximate log-likelihood of w^i and w^j.

$$\text{optimise} \quad s(w^i, w^j) = V_{w^j,:} C_{:,w^i} + \tilde{b}_{w^i} + \tilde{k}_{w^j} \tag{2.21}$$

$$\text{such that} \quad s(w^i, w^j) \approx \log(X_{w^i,w^j}) \tag{2.22}$$

This is done via the minimisation of

$$Loss = -\sum_{\forall w^i} \sum_{\forall w^j} f(X_{w^i,w^j}) \left(s(w^i, w^j) - \log(X_{w^i,w^j}) \right) \tag{2.23}$$

Where $f(x)$ is a weighing between 0 and 1 given by:

$$f(x) = \begin{cases} \left(\frac{x}{100}\right)^{0.75} & x < 100 \\ 1 & \text{otherwise} \end{cases} \tag{2.24}$$

This can be considered as a saturating variant of the effective weighing of skip-gram being X_{w^i,w^j}.

While GloVe out-performed skip-gram in initial tests subsequent more extensive testing in Levy, Goldberg, and Dagan (2015) with more tuned parameters, found that skip-gram marginally out-performed GloVe on all tasks.

2.3.2 Further equivalence of Co-location Prediction to Factorisation

GloVe highlights the relationship between the co-located word prediction neural network models, and the more traditional non-negative matrix factorization of co-location counts used in topic modeling. Very similar properties were also explored for skip-grams with negative sampling in Levy and Goldberg (2014) and in Li et al. (2015) with more direct mathematical equivalence to weighed co-occurrence matrix factorisation; Later, Cotterell et al. (2017) showed the equivalence to exponential principal component analysis (PCA). Landgraf and Bellay (2017) goes on to extend this to show that it is a weighted logistic PCA, which is a special case of the exponential PCA. Many works exist in this area now.

2.3.3 Conclusion

We have now concluded that neural predictive co-location models are functionally very similar to matrix factorisation of co-location counts with suitable weightings, and suitable similarity metrics. One might now suggest a variety of word embeddings to be created from a variety of different matrix factorisations with different weightings and constraints. Traditionally large matrix factorisations have significant problems in terms of computational time and memory usage. A common solution to this, in applied mathematics, is to handle the factorisation using an iterative optimisation procedure. Training a neural network, such as skip-gram, is indeed just such an iterative optimisation procedure.

2.4 Hierarchical Softmax and Negative Sampling

Hierarchical softmax, and negative sampling are effectively alternative output layers which are computationally cheaper to evaluate than regular softmax. They are powerful methods which pragmatically allow for large speed-up in any task which involves outputting very large classification probabilities – such as language modelling.

2.4.1 Hierarchical Softmax

Hierarchical softmax was first presented in Morin and Bengio (2005). Its recent use was popularised by Mikolov et al. (2013b), where words are placed as leaves in a Huffman tree, with their depth determined by their frequency.

One of the most expensive parts of training and using a neural language model is to calculate the final softmax layer output. This is because the softmax denominator includes terms for each word in the vocabulary. Even if only one word's probability is to be calculated, one denominator term per word in the vocabulary must be evaluated. In hierarchical softmax, each word (output choice), is considered as a leaf on a binary tree. Each level of the tree roughly halves the space of the output words

to be considered. The final level to be evaluated for a given word contains the word's leaf-node and another branch, which may be a leaf-node for another word, or a deeper sub-tree

The tree is normally a Huffman tree (Huffman 1952), as was found to be effective by Mikolov et al. (2013b). This means that for each word w^i, the word's depth (i.e its code's length) $l(w^i)$ is such that over all words: $\sum_{\forall w^j \in \mathbb{V}} P(w^j) \times l(w^j)$ is minimised. Where $P(w^i)$ is word w^i's unigram probability, and $\mathbb{V}$ is the vocabulary. The approximate solution to this is that $l(w^i) \approx -\log_2(P(w^i))$. From the tree, each word can be assign a code in the usual way, with 0 for example representing taking one branch, and 1 representing the other. Each point in the code corresponds to a node in the binary tree, which has decision tied to it. This code is used to transform the large multinomial softmax classification into a series of binary logistic classifications. It is important to understand that the layers in the tree are not layers of the neural network in the normal sense – the layers of the tree do not have an output that is used as the input to another. The layers of the tree are rather subsets of the neurons on the output layer, with a relationship imparted on them.

It was noted by Mikolov et al. (2013b), that for vocabulary $\mathbb{V}$:

- Using normal softmax would require each evaluation to perform $|\mathbb{V}|$ operations.
- Using hierarchical softmax with a balanced tree, would mean the expected number of operations across all words would be $\log_2(|\mathbb{V}|)$.
- Using a Huffman tree gives the expected number of operations: $\sum_{\forall w^j \in \mathbb{V}} -P(w^j)\log_2(P(w^i)) = H(\mathbb{V})$, where $H(\mathbb{V})$ is the unigram entropy of words in the training corpus.

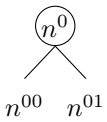
The worse case value for the entropy is $\log_2(|\mathbb{V}|)$. In-fact Huffman encoding is provably optimal in this way. As such this is the minimal number of operations required in the average case.

An incredibly gentle introduction to hierarchical softmax

In this section, for brevity, we will ignore the bias component of each decision at each node. It can either be handled nearly identically to the weight; or the matrix can be written in *design matrix form* with an implicitly appended column of ones; or it can even be ignored in the implementation (as was done in Mikolov et al. (2013b)). The reasoning for being able to ignore it is that the bias in normal softmax encodes unigram probability information; in hierarchical softmax, when used with the common Huffman encoding, its the tree's depth in tree encodes its unigram probability. In this case, not using a bias would at most cause an error proportionate to 2^{-k}, where k is the smallest integer such that $2^{-k} > P(w^i)$.

First consider a binary tree with just 1 layer and 2 leaves The leaves are n^{00} and n^{01}, each of these leaf nodes corresponds to a word from the vocabulary, which has size two, for this toy example.

Figure 2.10: Tree for 2 words



From the initial root which we call n^0, we can go to either node n^{00} or node n^{01}, based on the input from the layer below which we will call $\tilde{z}$.

Here we write n^{01} to represent the event of the first non-root node being the branch given by following left branch, while n^{01} being to follow the right branch. (The order within the same level is arbitrary in any-case, but for our visualisation purposes we'll used this convention.)

We are naming the root node as a notation convenience so we can talk about the decision made at n^0. Note that $P(n^0) = 1$, as all words include the root-node on their path.

We wish to know the probability of the next node being the left node (i.e. $P(n^{00} \mid \tilde{z})$) or the right-node (i.e. $P(n^{01} \mid \tilde{z})$). As these are leaf nodes, the prediction either equivalent to the prediction of one or the other of the two words in our vocabulary.

We could represent the decision with a softmax with two outputs. However, since it is a binary decision, we do not need a softmax, we can just use a sigmoid.

$$P(n^{01} \mid \tilde{z}) = 1 - P(n^{00} \mid \tilde{z}) \tag{2.25}$$

The weight matrix for a sigmoid layer has a number of columns governed by the number of outputs. As there is only one output, it is just a row vector. We are going to index it out of a matrix V. For the notation, we will use index 0 as it is associated with the decision at node n^0. Thus we call it $V_{0,:}$.

$$P(n^{00} \mid \tilde{z}) = \sigma(V_{0,:}\tilde{z}) \tag{2.26}$$
$$P(n^{01} \mid \tilde{z}) = 1 - \sigma(V_{0,:}\tilde{z}) \tag{2.27}$$

Note that for the sigmoid function: $1 - \sigma(x) = \sigma(-x)$. Allowing the formulation to be written:

$$P(n^{01} \mid \tilde{z}) = \sigma(-V_{0,:}\tilde{z}) \tag{2.28}$$

thus

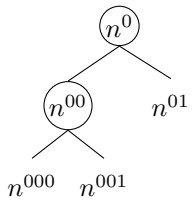
$$P(n^{0i} \mid \tilde{z}) = \sigma((-1)^i V_{0,:}\tilde{z}) \tag{2.29}$$

Noting that in Equation (2.29), i is either 0 (with $-1^0 = 1$) or 1 (with $-1^1 = -1$)).

Now consider 2 layers with 3 leaves Consider a tree with nodes: n^0, n^{00},n^{000}, n^{001}, n^{01}. The leaves are n^{000}, n^{001}, and n^{01}, each of which represents one of the 3 words from the vocabulary.

From earlier we still have:

Figure 2.11: Tree for 3 words



$$P(n^{00} \mid \tilde{z}) = \sigma(V_{0,:}\tilde{z}) \tag{2.30}$$
$$P(n^{01} \mid \tilde{z}) = \sigma(-V_{0,:}\tilde{z}) \tag{2.31}$$

We must now to calculate $P(n^{000} \mid \tilde{z})$. Another binary decision must be made at node n^{00}. The decision at n^{00} is to find out if the predicted next node is n^{000} or n^{001}. This decision is made, with the assumption that we have reached n^{00} already.

So the decision is defined by $P(n^{000} \mid z,\, n^{00})$ is given by:

$$P(n^{000} \mid \tilde{z}) = P(n^{000} \mid \tilde{z},\, n^{00})\, P(n^{00} \mid \tilde{z}) \tag{2.32}$$
$$P(n^{000} \mid \tilde{z}, n^{00}) = \sigma(V_{00,:}\tilde{z}) \tag{2.33}$$
$$P(n^{001} \mid \tilde{z}, n^{00}) = \sigma(-V_{00,:}\tilde{z}) \tag{2.34}$$

We can use the conditional probability chain rule to recombine to compute the three leaf nodes final probabilities.

$$P(n^{01} \mid \tilde{z}) = \sigma(-V_{0,:}\tilde{z}) \tag{2.35}$$
$$P(n^{000} \mid \tilde{z}) = \sigma(V_{00,:}\tilde{z})\sigma(V_{0,:}\tilde{z}) \tag{2.36}$$
$$P(n^{001} \mid \tilde{z}) = \sigma(-V_{00,:}\tilde{z})\sigma(V_{0,:}\tilde{z}) \tag{2.37}$$

Continuing this logic Using this system, we know that for a node encoded at position $[0t^1t^2t^3 \dots t^L]$, e.g. $[010 \dots 1]$, its probability can be found recursively as:

$$P(n^{0t^1 \dots t^L} \mid \tilde{z}) = P(n^{0t^1 \dots t^L} \mid \tilde{z},\, n^{0t^1 \dots t^{L-1}})\, P(n^{0t^1 \dots t^{L-1}} \mid \tilde{z}) \tag{2.38}$$

Thus:

$$P(n^{0t^1} \mid \tilde{z}) = \sigma\left((-1)^{t^1} V_{0,:}\tilde{z}\right) \tag{2.39}$$
$$P(n^{0t^1,t^2} \mid \tilde{z},\, n^{0t^1}) = \sigma\left((-1)^{t^2} V_{0t^1,:}\tilde{z}\right) \tag{2.40}$$
$$P(n^{0t^1 \dots t^i} \mid \tilde{z},\, n^{0t^1 \dots t^{i-1}}) = \sigma\left((-1)^{t^i} V_{0t^1 \dots t^{i-1},:}\tilde{z}\right) \tag{2.41}$$

The conditional probability chain rule, is applied to get:

$$P(n^{0t^1 \dots t^L} \mid \tilde{z}) = \prod_{i=1}^{i=L} \sigma\left((-1)^{t^i} V_{0t^1 \dots t^{i-1},:}\tilde{z}\right) \tag{2.42}$$

Formulation

The formulation above is not the same as in other works. This subsection shows the final steps to reach the conventional form used in Mikolov et al. (2013a).

Here we have determined that the 0th/left branch represents the positive choice, and the other probability is defined in terms of this. It is equivalent to have the 1th/right branch representing the positive choice:

$$P(n^{0t^1\ldots t^L} \mid \tilde{z}) = \prod_{i=1}^{i=L} \sigma\left((-1)^{t^i+1} V_{0t^1\ldots t^{i-1},:}\tilde{z}\right) \tag{2.43}$$

or to allow it to vary per node: as in the formulation of Mikolov et al. (2013a). In that work they use $ch(n)$ to represent an arbitrary child node of the node n and use an indicator function $[\![a=b]\!] = \begin{cases} 1 & a=b \\ -1 & a\neq b \end{cases}$ such that they can write $\left[\!\left[n^b = ch(n^a)\right]\!\right]$ which will be 1 if n^a is an arbitrary (but consistent) child of n^b, and 0 otherwise.

$$P(n^{0t^1\ldots t^L} \mid \tilde{z}) = \prod_{i=1}^{i=L} \sigma\left(\left[\!\left[n^{0t^1\ldots t^i} = ch(n^{0t^1\ldots t^{i-1}})\right]\!\right] V_{0t^1\ldots t^{i-1},:}\tilde{z}\right) \tag{2.44}$$

There is no functional difference between the three formulations. Though the final one is perhaps a key reason for the difficulties in understanding the hierarchical softmax algorithm.

Loss Function

Using normal softmax, during the training, the cross-entropy between the model's predictions and the ground truth as given in the training set is minimised. Cross entropy is given by

$$CE(P^\star, P) = \sum_{\forall w^i \in \mathbb{V}} \sum_{\forall z^j \in \mathbb{Z}} -P^\star(w^i \mid z^j) \log P(w^i \mid z^j) \tag{2.45}$$

Where $P^\star$ is the true distribution, and P is the approximate distribution given by our model (in other sections we have abused notation to use P for both). $\mathbb{Z}$ is the set of values that are input into the model, (or equivalently the values derived from them from lower layers) – Ithe context words in language modelling. $\mathbb{V}$ is the set of outputs, the vocabulary in language modeling. The training dataset $\mathcal{X}$ consists of pairs from $\mathbb{V}\times\mathbb{Z}$.

The true probabilities (from $P^\star$) are implicitly given by the frequency of the training pairs in the training dataset $\mathcal{X}$.

$$Loss = CE(P^\star, P) = \frac{1}{|\mathcal{X}|} \sum_{\forall (w^i, z^i)\in\mathcal{X}} -\log P(w^i \mid z^i) \tag{2.46}$$

The intuitive understanding of this, is that we are maximising the probability estimate of all pairings which actually occur in the training set, proportionate to how often the occur. Note that the $\mathbb{Z}$ can be non-discrete values, as was the whole benefit of using embeddings, as discussed in Section 2.1.1.

This works identically for hierarchical softmax as for normal softmax. It is simply a matter of substituting in the (different) equations for P. Then applying back-propagation as usual.

2.4.2 Negative Sampling

Negative sampling was introduced in Mikolov et al. (2013a) as another method to speed up this problem. Much like hierarchical softmax in its purpose. However, negative sampling does not modify the network's output, but rather the loss function.

Negative Sampling is a simplification of Noise Contrast Estimation (Gutmann and Hyvärinen 2012). Unlike Noise Contrast Estimation (and unlike softmax), it does not in fact result in the model converging to the same output as if it were trained with softmax and cross-entropy loss. However the goal with these word embeddings is not to actually perform the language modelling task, but only to capture a high-quality vector representation of the words involved.

A Motivation of Negative Sampling

Recall from Section 2.2.2 that the (supposed) goal, is to estimate $P(w^j \mid w^i)$. In truth, the goal is just to get a good representation, but that is achieved via optimising the model to predict the words. In Section 2.2.2 we considered the representation of $P(w^j \mid w^i)$ as the w^jth element of the softmax output.

$$P(w^j \mid w^i) = \text{smax}(V\, C_{:,w^i})_{w^j} \tag{2.47}$$

$$P(w^j \mid w^i) = \frac{\exp\left(V_{w^j,:} C_{:,w^i}\right)}{\sum_{k=1}^{k=N} \exp\left(V_{k,:} C_{:,k}\right)} \tag{2.48}$$

This is not the only valid representation. One could use a sigmoid neuron for a direct answer to the co-location probability of w^j occurring near w^i. Though this would throw away the promise of the probability distribution to sum to one across all possible words that could be co-located with w^i. That promise could be enforced by other constraints during training, but in this case it will not be. It is a valid probability if one does not consider it as a single categorical prediction, but rather as independent predictions.

$$P(w^j \mid w^i) = \sigma(V\, C_{:,w^i})_{w^j} \tag{2.49}$$

$$\text{i.e.} \quad P(w^j \mid w^i) = \sigma(V_{w^j,:} C_{:,w^i}) \tag{2.50}$$

Lets start from the cross-entropy loss. In training word w^j does occur near w^i, we know this because they are a training pair presented from the training dataset $\mathcal{X}$. Therefore, since it occurs, we could make a loss function based on minimising the negative log-likelihood of all observations.

$$Loss = \sum_{\forall(w^i,w^j)\in\mathcal{X}} -\log P(w^j \mid w^i) \tag{2.51}$$

This is the cross-entropy loss, excluding the scaling factor for how often it occurs.

However, we are not using softmax in the model output, which means that there is no trade off for increasing (for example) $P(w^1 \mid w^i)$ vs $P(w^2 \mid w^i)$. This thus admits the trivially optimal solution $\forall w^j \in \mathbb{V}$ $P(w^j \mid w^i) = 1$. This is obviously wrong – even beyond not being a proper distribution – some words are more commonly co-occurring than others.

So from this we can improve the statement. What is desired from the loss function is to reward models that predict the probability of words that *do* co-occur as being higher, than the probability of words that *do not*. We know that w^j does occur near w^i as it is in the training set. Now, let us select via some arbitrary means a w^k that does not – a negative sample. We want the loss function to be such that $P(w^k \mid w^i) < P(w^j \mid w^i)$. So for this single term in the loss we would have:

$$loss(w^j, w^i) = \log P(w^k \mid w^i) - \log P(w^j \mid w^i) \tag{2.52}$$

The question is then: how is the negative sample w^k to be found? One option would be to deterministically search the corpus for these negative samples, making sure to never select words that actually do co-occur. However that would require enumerating the entire corpus. We can instead just pick them randomly, we can sample from the unigram distribution. As statistically, in any given corpus most words do not co-occur, a randomly selected word in all likelihood will not be one that truly does co-occur – and if it is, then that small mistake will vanish as noise in the training, overcome by all the correct truly negative samples.

At this point, we can question, why limit ourselves to one negative sample? We could take many, and do several at a time, and get more confidence that $P(w^j \mid w^i)$ is indeed greater than other (non-existent) co-occurrence probabilities. This gives the improved loss function of

$$loss(w^j, w^i) = \left(\sum_{\forall w^k \in \text{samples}(D^{1g})} \log P(w^k \mid w^i)\right) - \log P(w^j \mid w^i) \tag{2.53}$$

where D^{1g} stands for the unigram distribution of the vocabulary and samples(D^{1g}) is a function that returns some number of samples from it.

Consider, though is this fair to the samples? We are taking them as representatives of all words that do not co-occur. Should a word that is unlikely to occur at all, *but was unlucky enough to be sampled*, contribute the same to the loss as a word that was very likely to occur? More reasonable is that the loss contribution should be in proportion to how likely the samples were to occur. Otherwise it will add unexpected

changes and result in noisy training. Adding a weighting based on the unigram probability ($P^{1g}(w^k)$) gives:

$$loss(w^j, w^i) = \left(\sum_{\forall w^k \in \text{samples}(D^{1g})} P^{1g}(w^k) \log P(w^k \mid w^i) \right) - \log P(w^j \mid w^i) \quad (2.54)$$

The expected value is defined by

$$\mathop{\mathbb{E}}_{X \sim D}[f(x)] = \sum_{\forall x \text{ values for } X} P^{d} f(x) \quad (2.55)$$

In an abuse of notation, we apply this to the samples, as a sample expected value and write:

$$\sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k \sim D^{1g}}[\log P(w^k \mid w^i)] \quad (2.56)$$

to be the sum of the n samples expected values. This notation (abuse) is as used in Mikolov et al. (2013a). It gives the form:

$$loss(w^j, w^i) = \left(\sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k \sim D^{1g}}[\log P(w^k \mid w^i)]) \right) - \log P(w^j \mid w^i) \quad (2.57)$$

Consider that the choice of unigram distribution for the negative samples is not the only choice. For example, we might wish to increase the relative occurrence of rare words in the negative samples, to help them fit better from limited training data. This is commonly done via subsampling in the positive samples (i.e. the training cases)). So we replace D^{1g} with D^{ns} being the distribution of negative samples from the vocabulary, to be specified as a hyper-parameter of training.

Mikolov et al. (2013a) uses a distribution such that

$$P^{D^{ns}}(w^k) = \frac{P^{D^{1g}}(w^k)^{\frac{2}{3}}}{\sum_{\forall w^o \in \mathbb{V}} P^{D^{1g}}(w^o)^{\frac{2}{3}}} \quad (2.58)$$

which they find to give better performance than the unigram or uniform distributions.

Using this, and substituting in the sigmoid for the probabilities, this becomes:

$$loss(w^j, w^i) = \left(\sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k \sim D^{ns}}[\log \sigma(V_{w^k,:} C_{:,w^i}) \right) - \log \sigma(V_{w^j,:} C_{:,w^i}) \quad (2.59)$$

By adding a constant we do not change the optimal value. If we add the constant $-K$, we can subtract 1 in each sample term.

$$loss(w^j, w^i) = \left(\sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k \sim D^{\text{ns}}}[-1 + \log \sigma(V_{w^k,:} C_{:,w^i})\right) - \log \sigma(V_{w^j,:} C_{:,w^i}) \quad (2.60)$$

Finally we make use of the identity $1 - \sigma(\tilde{z}) = \sigma(-\tilde{z})$ giving:

$$loss(w^j, w^i) = - \log \sigma(V_{w^j,:} C_{:,w^i}) - \sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k \sim D^{\text{ns}}}[\log \sigma(-V_{w^k,:} C_{:,w^i})] \quad (2.61)$$

Calculating the total loss over the training set $\mathcal{X}$:

$$Loss = -\sum_{\forall (w^i, w^j) \in \mathcal{X}} \left(\log \sigma(V_{w^j,:} C_{:,w^i}) + \sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k \sim D^{\text{ns}}}[\log \sigma(-V_{w^k,:} C_{:,w^i})]\right) \quad (2.62)$$

This is the negative sampling loss function used in Mikolov et al. (2013a). Perhaps the most confusing part of this is the notation. Without the abuses around expected value, this is written:

$$Loss = -\sum_{\forall (w^i, w^j) \in \mathcal{X}} \left(\log \sigma(V_{w^j,:} C_{:,w^i}) + \sum_{\forall w^k \in \text{samples}(D^{\text{ns}})} P^{\text{D}^{\text{ns}}}(w^k) \log \sigma(-V_{w^k,:} C_{:,w^i})\right) \quad (2.63)$$

2.5 Natural Language Applications – beyond language modeling

While statistical language models are useful, they are of-course in no way the be-all and end-all of natural language processing. Simultaneously with the developments around representations for the language modelling tasks, work was being done on solving other NLP problems using similar techniques (Collobert and Weston 2008).

2.5.1 Using Word Embeddings as Features

Turian, Ratinov, and Bengio (2010) discuss what is now perhaps the most important use of word embeddings. The use of the embeddings as features, in unrelated feature driven models. One can find word embeddings using any of the methods discussed above. These embeddings can be then used as features instead of, for example bag of words or hand-crafted feature sets. Turian, Ratinov, and Bengio (2010) found improvements on the state of the art for chunking and Named Entity Recognition (NER), using the word embedding methods of that time. Since then, these results have been superseded again using newer methods.

2.6 Aligning Vector Spaces Across Languages

Given two vocabulary vector spaces, for example one for German and one for English, a natural and common question is if they can be aligned such that one has a single vector space for both. Using canonical correlation analysis (CCA) one can do exactly that. There also exists generalised CCA for any number of vector spaces (Fu et al. 2016), as well as kernel CCA for a non-linear alignment.

The inputs to CCA, are two sets of vectors, normally expressed as matrices. We will call these: $C \in \mathbb{R}^{n^{\mathrm{C}} \times m^{\mathrm{C}}}$ and $V \in \mathbb{R}^{n^{\mathrm{V}} \times m^{\mathrm{V}}}$. They are both sets of vector representations, not necessarily of the same dimensionality. They could be the output of any of the embedding models discussed earlier, or even a sparse (non-embedding) representations such as the point-wise mutual information of the co-occurrence counts. The other input is series pairs of elements from within those those sets that are to be aligned. We will call the elements from that series of pairs from the original sets $C^\star$ and $V^\star$ respectively. $C^\star$ and $V^\star$ are subsets of the original sets, with the same number of representations. In the example of applying this to translation, if each vector was a word embedding: $C^\star$ and $V^\star$ would contains only words with a single known best translation, and this does not have to be the whole vocabulary of either language.

By performing CCA one solves to find a series of vectors (also expressed as a matrix), $S = \left[\tilde{s}^1 \dots \tilde{s}^{\mathrm{d}}\right]$ and $T = \left[\tilde{t}^1 \dots \tilde{t}^{\mathrm{d}}\right]$, such that the correlation between $C^\star \tilde{s}^{\mathrm{i}}$ and $V^\star \tilde{t}^{\mathrm{i}}$ is maximised, with the constraint that for all $j < i$ that $C^\star \tilde{s}^{\mathrm{i}}$ is uncorrelated with $C^\star \tilde{s}^{\mathrm{j}}$ and that $V^\star \tilde{t}^{\mathrm{i}}$ is uncorrelated with $V^\star \tilde{t}^{\mathrm{j}}$. This is very similar to principal component analysis (PCA), and like PCA the number of components to use (d) is a variable which can be decreased to achieve dimensionality reduction. When complete, taking S and T as matrices gives projection matrices which project C and V to a space where aligned elements are as correlated as possible. The new common vector space embeddings are given by: CS and VT. Even for sparse inputs the outputs will be dense embeddings.

Faruqui and Dyer (2014) investigated this primarily as a means to use additional data to improve performance on monolingual tasks. In this, they found a small and inconsistent improvement. However, we suggest it is much more interesting as a multi-lingual tool. It allows similarity measures to be made between words of different languages. Gujral, Khayrallah, and Koehn (2016) use this as part of a hybrid system to translate out of vocabulary words. Klein et al. (2015) use it to link word-embeddings with image embeddings.

Dhillon, Foster, and Ungar (2011) investigated using this to create word-embeddings. We noted in Equation (2.16), that skip-gram maximise the similarity of the output and input embeddings according to the dot-product. CCA also maximises similarity (according the correlation), between the vectors from one set, and the vectors for another. As such given representations for two words from the same context, initialised randomly, CCA could be used repeatedly to optimise towards good word embedding capturing shared meaning from contexts. This principle was used by Dhillon, Foster, and Ungar (2011), though their final process more complex than described here. It is perhaps one of the more un-

usual ways to create word embeddings as compared to any of the methods discussed earlier.

Aligning embeddings using linear algebra after they are fully trained is not the only means to end up with a common vector space. One can also directly train embeddings on multiple languages concurrently as was done in Shi et al. (2015), amongst others. Similarly, on the sentence embedding side Zou et al. (2013), and Socher et al. (2014) train embeddings from different languages and modalities (respectively) directly to be near to their partners (these are discussed in Chapter 4). A survey paper on such methods was recently published by Ruder (2017).

Chapter 3

Word Sense Representations

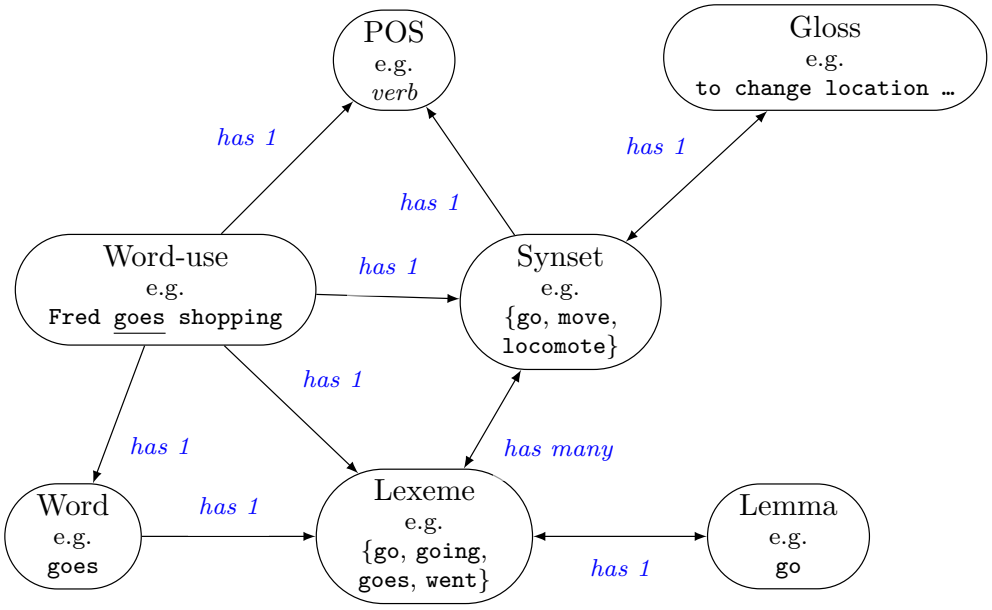
This chapter originally appeared as Chapter 4 of the book "Neural Representations of Natural Language", published by by Springer. The full book can be made available to the examiners on request. This chapter is not to be distributed to the general public and will not appear in the public online archival copy of the thesis.

> **1a.** *In a literal, exact, or actual sense; not figuratively, allegorically, etc.*
> **1b.** *Used to indicate that the following word or phrase must be taken in its literal sense, usually to add emphasis.*
> **1c.** *colloq. Used to indicate that some (frequently conventional) metaphorical or hyperbolical expression is to be taken in the strongest admissible sense: 'virtually, as good as'; (also) 'completely, utterly, absolutely'* ...
> **2a** *With reference to a version of something, as a transcription, translation, etc.: in the very words, word for word.*
> **2b.** *In extended use. With exact fidelity of representation; faithfully.*
> **3a.** *With or by the letters (of a word). Obs. rare.*
> **3b.** *In or with regard to letters or literature. Obs. rare.*

– the seven senses of `literally`, *Oxford English Dictionary*, 3rd ed., 2011

Abstract

Figure 3.1: The relationship between terms used to discuss various word sense problems. The lemma is used as the representation for the lexeme, for WordNet's purposes when indexing. For many tasks each the word-use is pre-tagged with its lemma and POS tag, as these can be found with high reliability using standard tools. Note that the arrows in this diagram are *directional*. That is to say, for example, each Synset *has 1* POS, but each POS *has many* Synsets.



In this chapter, techniques for representing the multiple meanings of a single word are discussed. This is a growing area, and is particularly important in languages where polysemous and homonymous words are common. This includes English, but it is even more prevalent in Mandarin for example. The techniques discussed can broadly be classified as lexical word sense representation, and as word sense induction. The inductive techniques can be sub-classified as clustering-based or as prediction-based.

3.1 Word Senses

Words have multiple meanings. A single representation for a word cannot truly describe the correct meaning in all contexts. It may have some features that are applicable to some uses but not to others, it may be an average of all features for all uses, or it may only represent the most common sense. For most word-embeddings it will be an unclear combination of all of the above. Word sense embeddings attempt to find representations not of words, but of particular senses of words.

The standard way to assign word senses is via some lexicographical resource, such as a dictionary, or a thesaurus. There is not a canonical list of word senses that are consistently defined in English. Every dictionary is unique, with different definitions and numbers of word senses. The most commonly used lexicographical resource is WordNet (Miller 1995), and the multi-lingual BabelNet (Navigli and Ponzetto 2010). The relationship between the terminology used in word sense problems is shown in Figure 3.1

3.1.1 Word Sense Disambiguation

Word sense disambiguation is one of the hardest problems in NLP. Very few systems significantly out perform the baseline, i.e. the most frequent sense (MFS) technique.

Progress on the problem is made difficult by several factors.

The sense is hard to identify from the context. Determining the sense may require very long range information: for example the information on context may not even be in the same sentence. It may require knowing the domain of the text, because word sense uses vary between domains. Such information is external to the text itself. It may in-fact be intentionally unclear, with multiple correct interpretations, as in a pun. It maybe unintentionally unknowable, due to a poor writing style, such that it would confuse any human reader. These difficulties are compounded by the limited amount of data available.

There is only a relatively small amount of labelled data for word sense problems. It is the general virtue of machine learning that given enough data, almost any input-output mapping problem (i.e. function approximation) can be solved. Such an amount of word sense annotated data is not available. This is in contrast to finding unsupervised word embeddings, which can be trained on any text that has ever been written. The lack of very large scale training corpora renders fully supervised methods difficult. It also results in small sized testing corpora; which leads to systems that may appear to perform well (on those small test corpora), but do not generalise to real world uses. In addition, the lack of human agreement on the correct sense, resulting in weak ground truth, further makes creating new resources harder. This limited amount of data compounds the problem's inherent difficulties.

It can also be said that word senses are highly artificial and do not adequately represent meaning. However, WSD is required to interface with lexicographical resources, such as translation dictionaries (e.g. BabelNet), ontologies (e.g. OpenCyc), and other datasets (e.g. ImageNet (Deng et al. 2009)).

It may be interesting to note, that the number of meanings that a word has is approximately inversely proportional related to its frequency of use rank (Zipf 1945). That is to say the most common words have far more meanings than rarer words. It is related to (and compounds with) the more well-known Zipf's Law on word use (Zipf 1949), and can similarly be explained-based on Zipf's core premise of the principle of least effort. This aligns well with our notion that precise (e.g. technical) words exist but are used only infrequently – since they are only explaining a single situation. This also means that by most word-uses are potentially very ambiguous.

The most commonly used word sense (for a given word) is also overwhelmingly more frequent than its less common brethren – word sense usage also being roughly Zipfian distributed (Kilgarriff 2004). For this reason the Most Frequent Sense (MFS) is a surprisingly hard baseline to beat in any WSD task.

Most Frequent Sense

Given a sense annotated corpus, it is easy to count how often each sense of a word occurs. Due to the over-whelming frequency of the most frequent sense, it is unlikely for even a small training corpus to have the most frequent sense differing from the use in the language as a whole.

The Most Frequent Sense (MFS) method of word sense disambiguation is defined by counting the frequency of a particular word sense for a particular POS tagged word. For the ith word use being the word w^i, having some sense s^j then without any further context the probability of that sense being the correct sense is $P(s^j \mid w^i)$. One can use the part of speech tag p_i (for the ith word use) as an additional condition, and thus find $P(s^j \mid w^i, p_i)$. WordNet encodes this information for each lemma-synset pair (i.e. each word sense) using the SemCor corpus counts. This is also used for sense ordering, which is why most frequent sense is sometimes called first sense. This is a readily available and practical method for getting a baseline probability of each sense. Most frequent sense can be applied for word sense disambiguation using this frequency-based probability estimate: $\text{argmax}_{\forall s^j} P(s^j \mid w^i, p_i)$.

In the most recent SemEval WSD task (Moro and Navigli 2015), MFS beat all submitted entries for English, both overall, and on almost all cuts of the data. The results for other languages were not as good, however in other languages the true corpus-derived sense counts were not used.

3.2 Word Sense Representation

It is desirable to create a vector representation of a word sense much like in Chapter 2 representations were created for words. We desire to an embedding to represent each word sense, as normally represented by a word-synset pair. This section considers the representations for the lexical word senses as given from a dictionary. We consider a direct method of using a labelled corpus, and an indirect method makes use of simpler sense-embeddings to partially label a corpus before retraining. These methods create representations corresponding to senses from WordNet. Section 3.3 considers the case when the senses are to also be discovered, as well as represented.

3.2.1 Directly supervised method

The simple and direct method is to take a dataset that is annotated with word senses, and then treat each sense-word pair as if it were a single word, then apply any of the methods for word representation discussed in Chapter 2. Iacobacci, Pilehvar, and Navigli (2015) use a CBOW language model (Mikolov et al. 2013b) to do this. This does, however, run into the aforementioned problem, that there is relatively little training data that has been manually sense annotated. Iacobacci, Pilehvar, and Navigli (2015) use a third-party WSD tool, namely BabelFly (Moro, Raganato, and Navigli 2014), to annotate the corpus with senses. This allows for existing word representation techniques to be applied.

Chen, Liu, and Sun (2014) applies a similar technique, but using a word-embedding-based partial WSD system of their own devising, rather than an external WSD tool.

3.2.2 Word embedding-based disambiguation method

Chen, Liu, and Sun (2014) uses an almost semi-supervised approach to train sense vectors. They partially disambiguate their training corpus, using initial word sense vectors and WordNet. They then completely replace these original (phase one) sense-vectors, by using the partially disambiguated corpus to train new (phase two) sense-vectors via a skip-gram variant. This process is shown in Figure 3.2.

The **first phase** of this method is in essence a word-embedding-based WSD system. When assessed as such, they report that it only marginally exceeds the MFS baseline, though that is not at all unusual for WSD algorithms as discussed above.

They assign a sense vector to every word sense in WordNet. This sense vector is the average of word-embeddings of a subset of words in the gloss, as determined using pretrained skip-grams (Mikolov et al. 2013b). For the word w with word sense w^{s^i}, a set of candidate words, $cands(w^{s^i})$, is selected from the gloss based on the following set of requirements. First, the word must be a content word: that is a verb, noun, adverb or adjective; secondly, its cosine distance to w must be below some threshold δ; finally, it must not be the word itself. When these requirements are followed $cands(w^{s^i})$ is a set of significant closely related words from the gloss.

The phase one sense vector for w^{s^i} is the mean of the word vectors for all the words in $cands(w^{s^i})$. The effect of this is that we expect that the phase one sense vectors for most words in the same synset will be similar but not necessarily identical. This expectation is not guaranteed however. As an example, consider the use of the word `china` as a synonym for `porcelain`: the single sense vector for `china` will likely be dominated by its more significant use referring to the country, which would cause very few words in the gloss for the `porcelain` synset to be included in *cands*. Resulting in the phase one sense vectors for the synonymous senses of `porcelain` and `china` actually being very different.

The phase one sense vectors are used to disambiguate the words in their unlabelled training corpus. For each sentence in the corpus, an initial *content vector* is defined by taking the mean of the skip-gram word embedding (not word sense) for all content words in the sentence. For each word in the sentence, each possible sense-embedding is compared to the context vector. If one or more senses vectors are found to be closer than a given threshold, then that word is tagged with the closest of those senses, and the context vector is updated to use the sense-vector instead of the word vector. Words that do not come within the threshold are not tagged, and the context vector is not updated. This is an important part of their algorithm, as it ensures that words without clear senses do not get a sense ascribed to them. This thus avoids any dubious sense tags for the next training step.

In **phase two** of training Chen, Liu, and Sun (2014) employ the skip-gram word-embedding method, with a variation, to predict the word senses. They train it on the partially disambiguated corpus produced in phase one. The original sense vectors are discarded. Rather than the model being tasked only to predict the surrounding words, it is tasked to predict surrounding words and their sense-tags (where present). In the loss function the prediction of tags and words is weighted equally.

Note that the input of the skip-gram is the just central word, not the pair of central word with sense-tag. In this method, the word sense embeddings are output embeddings; though it would not be unreasonable to reverse it to use input embeddings with sense tags, or even to do both. The option to have input embeddings and output embeddings be from different sets, is reminiscent of Schwenk (2004) for word embeddings.

The phase one sense vectors have not been assessed on their representational quality. It could be assumed that because the results for these were not reported, they were worse than those found in phase two. The phase two sense vectors were not assessed for their capacity to be used for word sense disambiguation. It would be desirable to extend the method of Chen, Liu, and Sun (2014), to use the phase two vectors for WSD. This would allow this method to be used to disambiguate its own training data, thus enabling the method to become self-supervised.

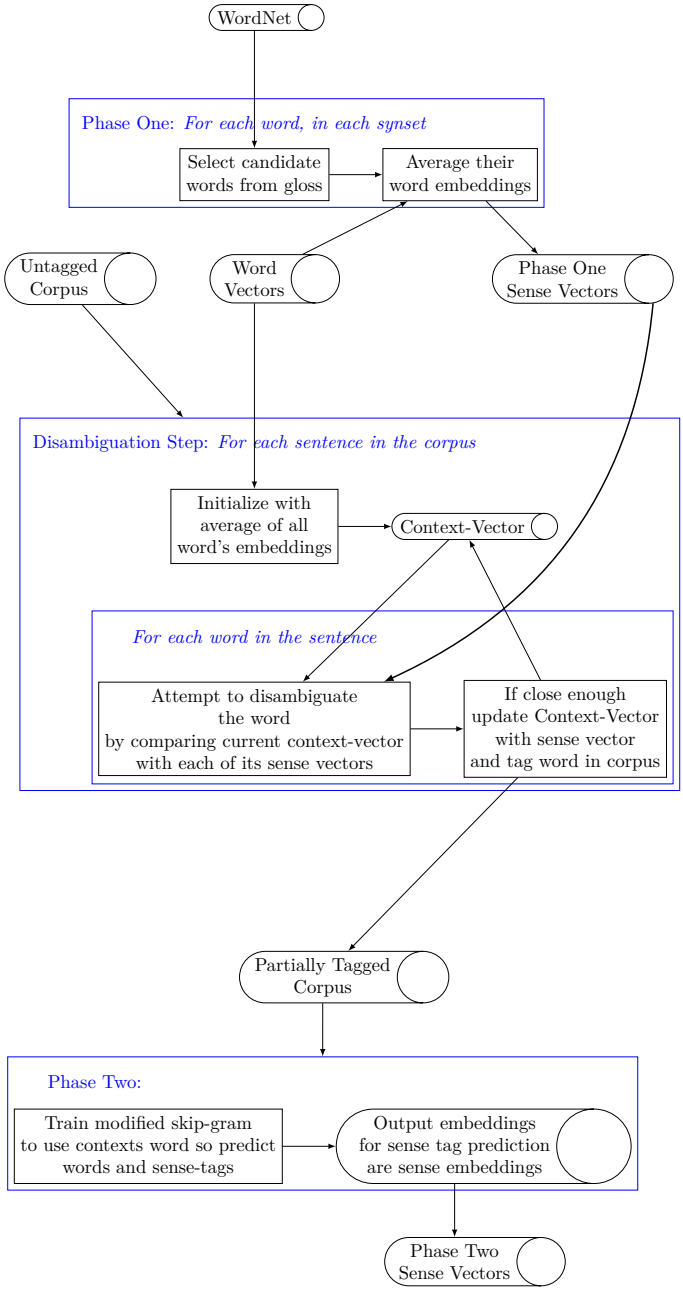
3.3 Word Sense Induction (WSI)

In this section we will discuss methods for finding a word sense without reference to a standard set of senses. Such systems must discover the word senses at the same time as they find their representations. One strong advantage of these methods is that they do not require a labelled dataset. As discussed there are relatively few high-quality word sense labelled datasets. The other key advantage of these systems is that they do not rely on fixed senses determined by a lexicographer. This is particularly useful if the word senses are highly domain specific; or in a language without strong lexicographical resources. This allows the data to inform on what word senses exist.

Most vector word sense induction and representation approaches are evaluated on similarity tests. Such tests include WordSim-353 (Finkelstein et al. 2001) for context-less, or Stanford's Contextual Word Similarities (SCWS) for similarity with context information (Huang et al. 2012). This evaluation is also suitable for evaluating single sense word-embeddings, e.g. skip-grams.

We can divide the WSI systems into context clustering-based approaches, and co-location prediction-based approaches. This division is similar to the separation of co-location matrix factorisation, and co-location prediction-based approaches discussed in Chapter 2. It can be assumed thus that at the core, like for word embeddings, they are fundamentally very similar. One could think of prediction of collocated words as a soft indirect clustering of contexts that can have those words.

Figure 3.2: The process used by Chen, Liu, and Sun (2014) to create word sense embeddings.



3.3.1 Context Clustering-based Approaches

As the meaning of a word, according to word embedding principles, is determined by the contexts in which it occurs, we expect that different meanings (senses) of the same words should occur in different contexts. If we cluster the contexts that a word occurs in, one would expect to find distinct clusters for each sense of the word. It is on this principle that the context clustering-based approaches function.

Offline clustering

The fundamental method for most clustering-based approaches is as per Schütze (1998). That original work is not a neural word sense embedding, however the approach remains the same. Pantel and Lin (2002) and Reisinger and Mooney (2010) are also not strictly neural word embedding approaches (being more classical vector representations), however the overall method is also very similar.

The clustering process is done by considering all word uses, with their contexts. The contexts can be a fixed-sized window of words (as is done with many word-embedding models), the sentence, or defined using some other rule. Given a pair of contexts, some method of measuring their similarity must be defined. In vector representational works, this is ubiquitously done by assigning each context a vector, and then using the cosine similarity between those vectors.

The **first step** in all the offline clustering methods is thus to define the representations of the contexts. Different methods define the context vectors differently:

- Schütze (1998) uses variations of inverse-document-frequency (idf) weighted bags of words, including applying dimensionality reduction to find a dense representation.
- Pantel and Lin (2002) use the mutual information vectors between words and their contexts.
- Reisinger and Mooney (2010), use td-idf or χ^2 weighted bag of words.
- Huang et al. (2012) uses td-idf weighted averages of (their own) single sense word embeddings for all words in the context.
- Kågebäck et al. (2015) also uses a weighted average of single sense word skip-gram embeddings, with the weighting based on two factors. One based on how close the words were, and the other on how likely the co-occurrence was according to the skip-gram model.

It is interesting to note that idf, td-idf, mutual information, skip-gram co-occurrence probabilities (being a proxy for point-wise mutual information (Levy and Goldberg 2014)), are all closely related measures.

The **second step** in off-line clustering is to apply a clustering method to cluster the word-uses. This clustering is done based on the calculated similarity of the context representation where the words are used. Again, different WSI methods use different clustering algorithms.

- Schütze (1998) uses a group average agglomerative clustering method.
- Pantel and Lin (2002) use a custom hierarchical clustering method.
- Reisinger and Mooney (2010) use mixtures of von-Mises-Fisher distributions.
- Huang et al. (2012) use spherical k-means.
- Kågebäck et al. (2015) use k-means.

The **final step** is to find a vector representation of each cluster. For non-neural embedding methods this step is not always done, as defining a representation is not the goal, though in general it can be derived from most clustering techniques. Schütze (1998) and Kågebäck et al. (2015) use the centroids of their clusters. Huang et al. (2012) use a method of relabelling the word uses with a cluster identifier, then train a (single-sense) word embedding method on cluster identifiers rather than words. This relabelling technique is similar to the method later used by Chen, Liu, and Sun (2014) for learning lexical sense representations, as discussed in Section 3.2.2. As each cluster of contexts represents a sense, those cluster embeddings are thus also considered as suitable word sense embeddings.

To summarize, all the methods for inducing word sense embeddings via off-line clustering follow the same process. **First**: represent the contexts of word use, so as to be able to measure their similarity. **Second**: use the context's similarity to cluster them. **Finally**: find a vector representation of each cluster. This cluster representation is the induced sense embedding.

Online clustering

The methods discussed above all use off-line clustering. That is to say the clustering is performed after the embedding is trained. Neelakantan et al. (2015) perform the clustering during training. To do this they use a modified skip-gram-based method. They start with a fixed number of randomly initialised sense vectors for each context. These sense vectors are used as input embeddings for the skip-gram context prediction task, over single sense output embeddings. Each sense also has, linked to it, a context cluster centroid, which is the average of all output embeddings for the contexts that the sense is assigned to. Each time a training instance is presented, the average of the context output embeddings is compared to each sense's context cluster centroid. The context is assigned to the cluster with the closest centroid, updating the centroid value. This can be seen as similar to performing a single k-means update step for each training instance. Optionally, if the closest centroid is further from the context vector than some threshold, a new sense can be created using that context vector as the initial centroid. After the assignment of the context to a cluster, the corresponding sense vector is selected for use as the input vector in the skip-gram context prediction task.

Kågebäck et al. (2015) investigated using their weighting function (as discussed in Section 3.3.1) with the online clustering used by Neelakantan et al. (2015). They found that this improved the quality of the representations. More generally any such weighting function could be used. This

online clustering approach is loosely similar to the co-location prediction-based approaches.

3.3.2 Co-location Prediction-based Approaches

Rather than clustering the contexts, and using those clusters to determine embeddings for different senses, one could consider the sense as a latent variable in the task used to find word embeddings – normally a language modelling task. The principle is that it is not the word that determines its collocated context words, but rather the word sense. So the word sense can be modelled as a hidden variable, where the word, and the context words are being observed.

Tian et al. (2014) used this to define a skip-gram-based method for word sense embeddings. For input word w^i with senses $\mathcal{S}(w^i)$, the probability of output word w^o occurring near w^i can be given as:

$$P(w^o \mid w^i) = \sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k, w^i) P(s^k \mid w^i) \tag{3.1}$$

Given that a sense s^k only belongs to one word w^i, we know that kth sense of the ith word only occurs when the ith word occurs. We have that the join probability $P(w^i, s^k) = P(s^k)$.

We can thus rewrite Equation (3.1) as:

$$P(w^o \mid w^i) = \sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k)\, P(s^k \mid w^i) \tag{3.2}$$

A softmax classifier can be used to define $P(w^o \mid s^k)$, just like in normal language modelling. With output embeddings for the words w^o, and input embeddings for the word senses s^k. This softmax can be sped-up using negative sampling or hierarchical softmax. The later was done by Tian et al. (2014).

Equation (3.2) is in the form of a mixture model with a latent variable. Such a class of problems are often solved using the Expectation Maximisation (EM) method. In short, the EM procedure functions by performing two alternating steps. The **E-step** calculates the expected chance of assigning word sense for each training case ($\hat{P}(s^l \mid w^o)$) in the training set $\mathcal{X}$. Where a training case is a pairing of a word use w^i, and context word w^o, with $s^l \in \mathcal{S}(w^i)$, formally we have:

$$\hat{P}(s^l \mid w^o) = \frac{\hat{P}(s^l \mid w^i) P(w^o \mid s^l)}{\sum_{\forall s^k \in \mathcal{S}(w^i)} \hat{P}(w^o \mid s^k) P(s^k \mid w^i)} \tag{3.3}$$

The **M-step** updates the prior likelihood of each sense (that is without context) using the expected assignments from the E-step.

$$\hat{P}(s^l \mid w^i) = \frac{1}{|\mathcal{X}|} \sum_{\forall (w^o, w^i) \in \mathcal{X}} \hat{P}(s^l \mid w^o) \tag{3.4}$$

During this step the likelihood of the $P(w^o \mid w^i)$ can be optimised to maximise the likelihood of the observations. This is done via gradient descent on the neural network parameters of the softmax component: $P(w^o \mid s^k)$. By using this EM optimisation the network can fit values for the embeddings in that softmax component.

A limitation of the method used by Tian et al. (2014), is that the number of each sense must be known in advance. One could attempt to solve this by using, for example, the number of senses assigned by a lexicographical resource (e.g. WordNet). However, situations where such resources are not available or not suitable are one of the main circumstances in which WSI is desirable (for example in work using domain specific terminology, or under-resourced languages). In these cases one could apply a heuristic-based on the distribution of senses-based on the distribution of words (Zipf 1945). An attractive alternative would be to allow senses to be determined-based on how the words are used. If they are used in two different ways, then they should have two different senses. How a word is being used can be determined by the contexts in which it appears.

Bartunov et al. (2015) extend on this work by making the number of senses for each word itself a fit-able parameter of the model. This is a rather Bayesian modelling approach, where one considers the distribution of the prior.

Considering again the form of Equation (3.2)

$$P(w^o \mid w^i) = \sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k) P(s^k \mid w^i) \tag{3.5}$$

The prior probability of a sense given a word, but no context, is $P(s^k \mid w^i)$. This is Dirichlet distributed. This comes from the definition of the Dirichlet distribution as the the prior probability of any categorical classification task. When considering that the sense my be one from an unlimited collection of possible senses, then that prior becomes a Dirichlet process.

In essence, this prior over a potentially unlimited number of possible senses becomes another parameter of the model (along with the input sense embeddings and output word embeddings). The fitting of the parameters of such a model is beyond the scope of this book; it is not entirely dissimilar to the fitting via expectation maximisation incorporating gradient descent used by Tian et al. (2014). The final output of Bartunov et al. (2015) is as desired: a set of induced sense embeddings, and a language model that is able to predict how likely a word is to occur near that word sense ($P(w^o \mid s^k)$).

By application of Bayes' theorem, the sense language model can be inverted to take a word's context, and predict the probability of each word sense.

$$P(s^l \mid w^o) = \frac{P(w^o \mid s^l) P(s^l \mid w^i)}{\sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k) P(s^k \mid w^i)} \tag{3.6}$$

with the common (but technically incorrect) assumption that all words in the context are independent.

Given a context window:
$\mathcal{W}^i = \left(w^{i-\frac{n}{2}}, \ldots, w^{i-1}, w^{i+1}, \ldots, w^{i+\frac{n}{2}}\right)$, we have:

$$P(s^l \mid \mathcal{W}^i) = \frac{\prod_{\forall w^j \in \mathcal{W}^i} P(w^j \mid s^l) P(s^l | w^i)}{\sum_{s^k \in \mathcal{S}(w^i)} \prod_{\forall w^j \in \mathcal{W}^i} P(w^j \mid s^k) P(s^k | w^i)} \tag{3.7}$$

3.4 Conclusion

Word sense representations allow the representations of the senses of words when one word has multiple meanings. This increases the expressiveness of the representation. These representations can in general be applied anywhere word embeddings can. They are particularly useful for translation, and in languages with large numbers of homonyms.

The word representation discussions in this chapter naturally lead to the next section on phrase representation. Rather than a single word having many meanings, the next chapter will discuss how a single meaning may take multiple words to express. In such longer structure's representations, the sense embeddings discussed here are often unnecessary, as the ambiguity may be resolved by the longer structure. Indeed, the methods discussed in this chapter have relied on that fact to distinguish the senses using the contexts.

Chapter 4

Sentence Representations and Beyond

A sentence is a group of words expressing a complete thought.
– *English Composition and Literature*, Webster, 1923

Abstract

This chapter discusses representations for larger structures in natural language. The primary focus is on the sentence level. However, many of the techniques also apply to sub-sentence structures (phrases), and super-sentence structures (documents). The three main types of representations discussed here are: unordered models, such as sum of word embeddings; sequential models, such as recurrent neural networks; and structured models, such as recursive autoencoders.

It can be argued that the core of true AI, is in capturing and manipulating the representation of an idea. In natural language a sentence (as defined by Webster in the quote above), is such a representation of an idea, but it is not machine manipulatable. As such the conversion of sentences to a machine manipulatable representation is an important task in AI research.

All techniques which can represent documents (or paragraphs) by necessity represent sentences as well. A document (or a paragraph), can consist only of a single sentence. Many of these models always work for sub-sentence structures also, like key-phrases. When considering representing larger documents, neural network embedding models directly compete with vector information retrieval models, such as LSI (Dumais et al. 1988), probabilistic LSI (Hofmann 2000) and LDA (Blei, Ng, and Jordan 2003).

4.1 Unordered and Weakly Ordered Representations

A model that does not take into account word order cannot perfectly capture the meaning of a sentence. Mitchell and Lapata (2008) give the poignant examples of:

- `It was not the sales manager who hit the bottle that day, but the office worker with the serious drinking problem.`
- `That day the office manager, who was drinking, hit the problem sales worker with a bottle, but it was not serious.`

These two sentences have the same words, but in a different structure, resulting in their very different meanings. In practice, however, representations which discard word order can be quite effective.

4.1.1 Sums of Word Embeddings

Classically, in information retrieval, documents have been represented as bags of words (BOW). That is to say a vector with length equal to the size of the vocabulary, with each position representing the count of the number of occurrences of a single word. This is much the same as a *one-hot vector* representing a word, but with every word in the sentence/document counted. The word embedding equivalent is sums of word embeddings (SOWE), and mean of word embeddings (MOWE). These methods, like BOW, lose all order information in the representation. In many cases it is possible to recover a BOW from a much lower dimensional SOWE (White et al. 2016a).

Surprisingly, these unordered methods have been found on many tasks to be extremely well performing, better than several of the more advanced techniques discussed later in this chapter. This has been noted in several works including: White et al. (2015), Ritter et al. (2015) and Wang, Liu, and McDonald (2017). It has been suggested that this is because in English there are only a few likely ways to order any given bag of words. It has been noted that given a simple n-gram language model the original sentences can often be recovered from BOW (Horvat and Byrne 2014) and thus also from a SOWE (White et al. 2016b). Thus word-order may not in-fact be as important as one would expect in many natural language tasks, as it is in practice more proscribed than one would expect. That is to say very few sentences with the same word content, will in-practice be able to have it rearranged for a very different meaning. However, this is unsatisfying, and certainly cannot capture fine grained meaning.

The step beyond this is to encode the n-grams into a bag of words like structure. This is a bag of n-grams (BON), e.g. bag of trigrams. Each index in the vector thus represents the occurrence of an n-gram in the text. So `It is a good day today`, has the trigrams: `(It is a)`,`(is a good)`,`(a good day)`, `(good day today)`. As is obvious for all but the most pathological sentences, recovering the full sentence order from a bag of n-grams is possible even without a language model.

The natural analogy to this with word embeddings might seem to be to find n-gram embeddings by the concatenation of n word embeddings; and then to sum these. However, such a sum is less informative than

it might seem. As the sum in each concatenated section is equal to the others, minus the edge words.

Instead one should train an n-gram embedding model directly. The method discussed in Chapter 2, can be adapted to use n-grams rather than words as the basic token. This was explored in detail by (Li et al. 2017). Their model is based on the skip-gram word embedding method. They take as input an n-gram embedding, and attempt to predict the surrounding n-grams. This reduces to the original skip-gram method for the case of unigrams. Note that the surrounding n-grams will overlap in words (for $n > 1$) with the input n-gram. As the overlap is not complete, this task remains difficult enough to encourage useful information to be represented in the embeddings. Li et al. (2017) also consider training n-gram embeddings as a bi-product of text classification tasks.

4.1.2 Paragraph Vector Models (Defunct)

Le and Mikolov (2014) introduced two models for representing documents of any length by using augmented word-embedding models. The models are called Paragraph Vector Distributed Memory (PV-DM) model, and the Paragraph Vector Distributed Bag of Words model (PV-DBOW). The name Paragraph Vector is a misnomer, it function on texts of any length and has most often (to our knowledge) been applied to documents and sentences rather than any in-between structures. The CBOW and skip-gram models are are extended with an additional context vector that represents the current document (or other super-word structure, such as sentence or paragraph). This, like the word embeddings, is initialised randomly, then trained during the task. Le and Mikolov (2014) considered that the context vector itself must contain useful information about the context. The effect in both cases of adding a context vector is to allow the network to learn a mildly different accusal language model depending on the context. To do this, the context vector would have to learn a representation for the context.

PV-DBOW is an extension of CBOW. The inputs to the model are not only the word-embedding $C_{:,w_j}$ for the words w^j from the window, but also a context-embedding $D_{:,d^k}$ for its current context (sentence, paragraph or document) d^k. The task remains to predict which word was the missing word from the center of the context w^i.

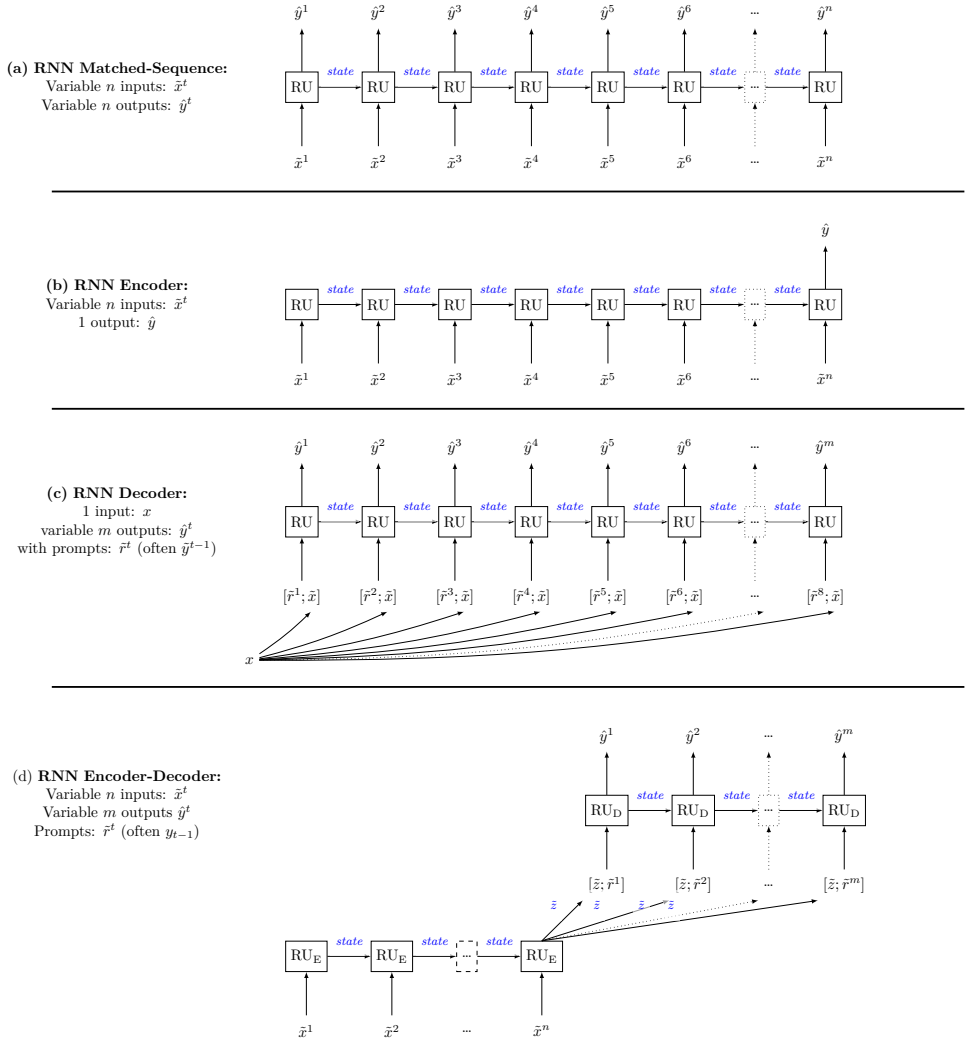
$$\begin{aligned} &P(w^i \mid d^k,\, w^{i-\frac{n}{2}}, ..., w^{i-1}, w^{i+1}, ..., w^{i+\frac{n}{2}}) \\ &\quad = \text{smax}(W D_{:,d^k} + U \sum_{j=i+1}^{j=\frac{n}{2}} \left(C_{:,w^{i-j}} + C_{:,w^{i+j}}\right)) \end{aligned} \tag{4.1}$$

PV-DM is the equivalent extension for skip-grams. Here the input to the model is not only the central word, but also the context vector. Again, the task remains to predict the other words from the window.

$$P(w^j \mid d^k,\, w^i) = \left[\text{smax}(W D_{:,d^k} + V\, C_{:,w^i})\right]_{w_j} \tag{4.2}$$

The results of this work are now considered of limited validity. There were failures to reproduce the reported results in the original evaluations which

Figure 4.1: The unrolled structure of an RNN for use in (a) Matched-sequence (b) Encoding, (c) Decoding and (d) Encoding-Decoding (sequence-to-sequence) problems. RU is the recurrent unit – the neural network which reoccurs at each time step.



were on sentiment analysis tasks. These were documented online by several users, including by the second author.[1] A follow up paper, Mesnil et al. (2014) found that reweighed bags of n-grams (Wang and Manning 2012) out performed the paragraph vector models. Conversely, Lau and Baldwin (2016) found that on short text-similarity problems, with the right tuning, the paragraph vector models could perform well; however they did not consider the reweighed n-grams of (Wang and Manning 2012). On a different short text task, White et al. (2015) found the paragraph vector models to significantly be out-performed by SOWE, MOWE, BOW, and BOW with dimensionality reduction. This highlights the importance of rigorous testing against a suitable baseline, on the task in question.

4.2 Sequential Models

The majority of this section draws on the recurrent neural networks (RNN) as discussed in Chapter 2 of White et al. (2018a). Every RNN

[1] `https://groups.google.com/forum/#!msg/word2vec-toolkit/Q49FIrNOQRo/DoRuBoVNFb0J`

learns a representation of all its input and output in its state. We can use RNN encoders and decoders (as shown in Figure 4.1) to generate representations of sequences by extracting a coding layer. One can take any RNN encoder, and select one of the hidden state layers after the final recurrent unit (RU) that has processed the last word in the sentence. Similarly for any RNN decoder, one can select any hidden state layer before the first recurrent unit that begins to produce words. For an RNN encoder-decoder, this means selecting the hidden layer from between. This was originally considered in Cho et al. (2014), when using a machine translation RNN, to create embeddings for the translated phrases. Several other RNNs have been used in this way since.

4.2.1 VAE and encoder-decoder

Bowman et al. (2016b) presents an extension on this notion, where in-between the encode and the decode stages there is a variational autoencoder (VAE). This is shown in Figure 4.2. The variational autoencoder (Kingma and Welling 2014) has been demonstrated to have very good properties in a number of machine learning applications: they are able to work to find continuous latent variable distributions over arbitrary likelihood functions (such as in the neural network); and are very fast to train. Using the VAE, it is hoped that a better representation can be found for the sequence of words in the input and output.

Bowman et al. (2016b) trained the network as encoder-decoder reproducing its exact input. They found that short syntactically similar sentences were located near to each other according to this space, further to that, because it has a decoder, it can generate these nearby sentences, which is not possible for most sentence embedding methods.

Interestingly, they use the VAE output, i.e. the *code*, only as the state input to the decoder. This is in-contrast to the encoder-decoders of Cho et al. (2014), where the *code* was concatenated to the input at every timestep of the decoder. Bowman et al. (2016b) investigated such a configuration, and found that it did not yield an improvement in performance.

4.2.2 Skip-thought

Kiros et al. (2015) draws inspiration from the works on acausal language modelling, to attempt to predict the previous and next sentence. Like in the acausal language modelling methods, this task is not the true goal. Their true goal is to capture a good representation of the current sentence. As shown in Figure 4.3 they use an encoder-decoder RNN, with two decoder parts. One decoder is to produce the previous sentence. The encoder part takes as it's input is the current sentence, and produces as its output the code, which is input to the decoders. The other decoder is to produce the next sentence. As described in Chapter 2 of White et al. (2018a), the prompt used for the decoders includes the previous word, concatenated to the code (from the encoder output).

That output code is the representation of the sentence.

Figure 4.2: The VAE plus encoder-decoder of Bowman et al. (2016b). Note that during training, $\hat{y}^i = w^i$, as it is an autoencoder model. As is normal for encoder-decoders the prompts are the previous output (target during training, predicted during testing): $r^i = y^{i-1}$, with $r^1 = y^0 =$ `<EOS>` being a pseudo-token marker for the end of the string. The Emb. step represents the embedding table lookup. In the diagrams for Chapter 2 we showed this as as a table but just as a block here for conciseness.

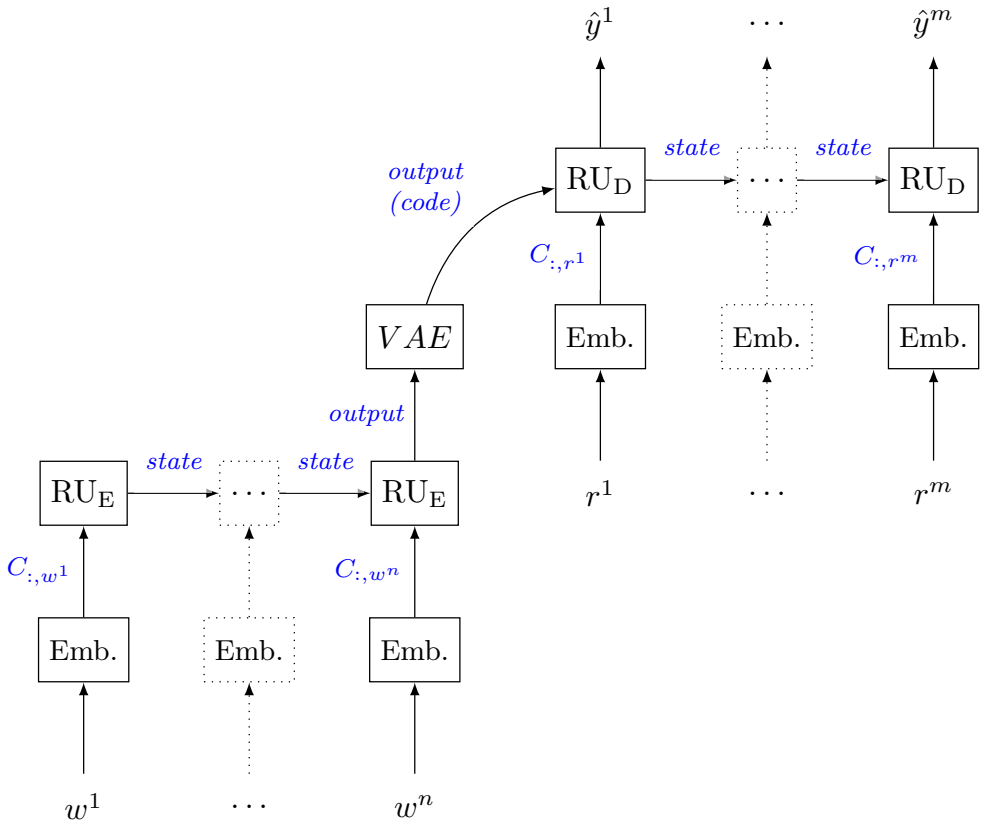
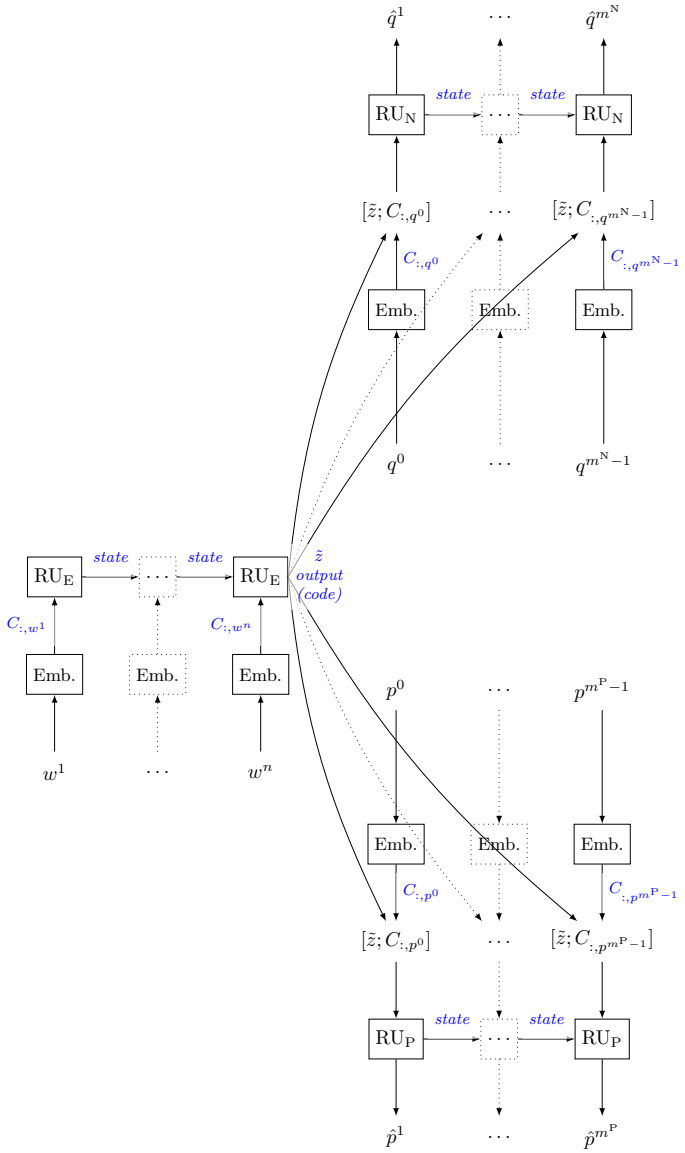


Figure 4.3: The skip-thought model (Kiros et al. 2015). Note that for the next and previous sentences respectively the outputs are $\hat{q}^i$ and $\hat{p}^i$, and the prompts are q^{i-1} and p^{i-1}. As there is no intent to use the decoders after training, there is no need to worry about providing an evaluation-time prompt, so the prompt is always the previous word. $p^0 = p^{m^{\mathrm{P}}} = q^0 = q^{m^{\mathrm{Q}}} =$ `<EOS>` being a pseudo-token marker for the end of the string. The input words are w^i, which come from the current sentence. the Emb. steps represents the look-up of the embedding for the word.



4.3 Structured Models

The sequential models are limited to processed the information as a series of time-steps one after the other. They processes sentences as ordered lists of words. However, the actual structure of a natural language is not so simple. Linguists tend to break sentences down into a tree structure. This is referred to as parsing. The two most common forms are constituency parse trees, and dependency parse trees. Examples of each are shown in Figures 4.4 and 4.5. It is beyond the scope of this book to explain the precise meaning of these trees, and how to find them. The essence is that these trees represent the structure of the sentence, according to how linguists believe sentences are processed by humans.

The constituency parse breaks the sentence down into parts such as noun phrase (NP) and verb phrase (VP), which are in turn broken down into phrases, or (POS tagged) words. The constituency parse is well thought-of as a hierarchical breakdown of a sentence into its parts. Conversely, a dependency parse is better thought of as a set of binary relations between head-terms and their dependent terms. These structures are well linguistically motivated, so it makes sense to use them in the processing of natural language.

We refer here to models incorporating tree (or graph) structures as structural models. Particular variations have their own names, such as recursive neural networks (RvNN), and recursive autoencoders (RAE). We use the term structural model as an all encompassing term, and minimise the use of the easily misread terms: recursive vs recurrent neural networks. A sequential model (an RNN) is a particular case of a structural model, just as a linked list is a particular type of tree. However, we will exclude sequential models them this discussion except where marked.

The initial work on structural models was done in the thesis of Socher (2014). It builds on the work of Goller and Kuchler (1996) and Pollack (1990), which present back-propagation through structure. Back-propagation can be applied to networks of any structure, as the chain-rule can be applied to any differentiable equation to find its derivative. Structured networks, like all other networks, are formed by the composition of differentiable functions, so are differentiable. In a normal network the same composition of functions is used for all input cases, whereas in a structured network it is allowed to vary based on the inputs. This means that structuring a network according to its parse tree is possible.

4.3.1 Constituency Parse Tree (Binary)

Tree structured networks work by applying a recursive unit (which we will call RV) function across pairs (or other groups) of the representations of the lower levels, to produce a combined representation. The network structure for an input of binary tree structured text is itself a binary tree of RVs. Each RV (i.e. node in the graph) can be defined by the

Figure 4.4: A constituency parse tree for the sentence: `This is a simple example of a parse tree`. In this diagram the leaf nodes are the input words, their intimidate parents are their POS tags, and the other nodes with multiple children represent sub-phrases of the sentence, for example NP is a Noun Phrase.

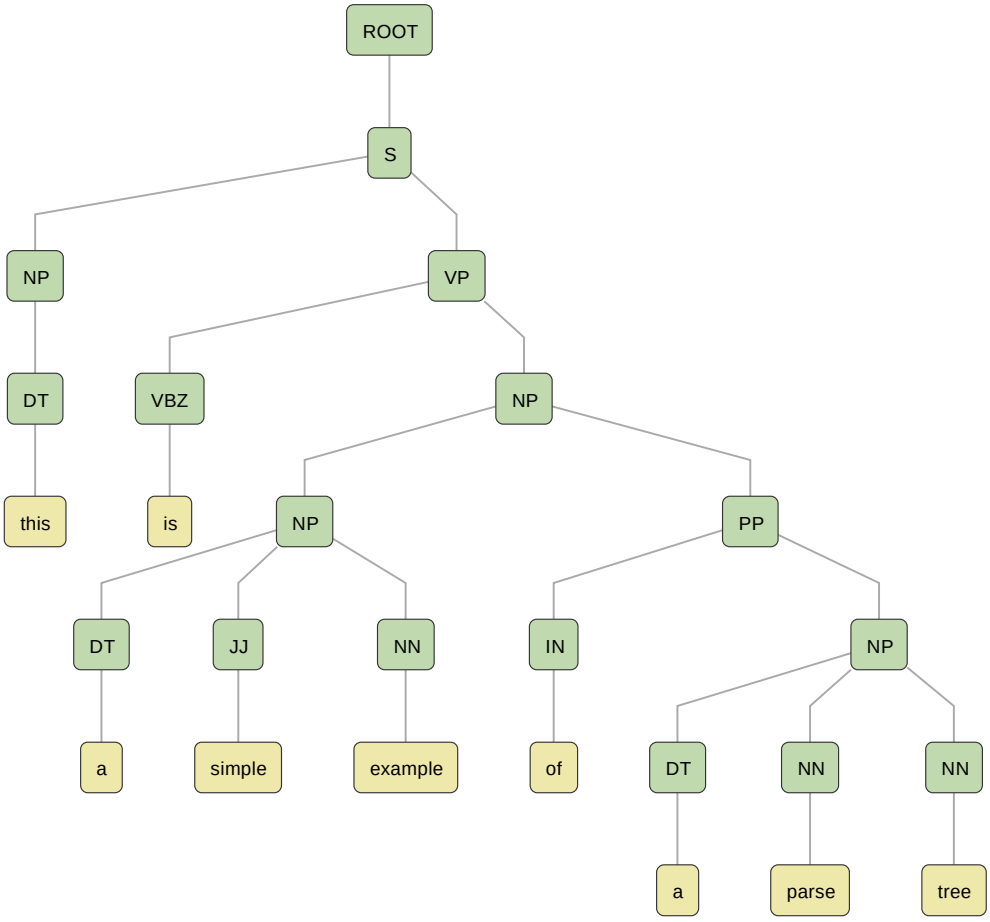
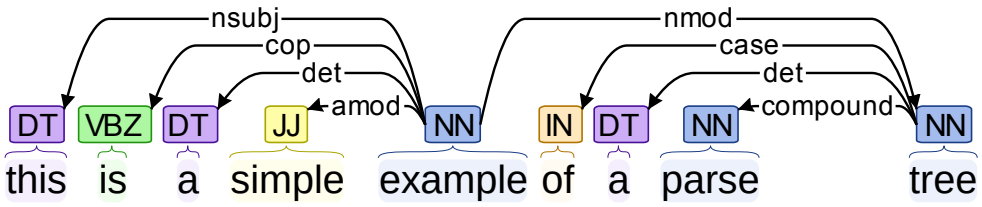


Figure 4.5: A dependency parse tree for the sentence `This is a simple example of a parse tree`, This flattened view may be misleading. `example` is at the peak of the tree, with direct children being: `this`,`is`,`a`,`simple`, and `tree`. `tree` has direct children being: `of`,`a`, and `parse`.



composition function:

$$f^{\text{RV}}(\tilde{u}, \tilde{v}) = \varphi\left(\begin{bmatrix} S & R \end{bmatrix} \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} + \tilde{b}\right) \tag{4.3}$$

$$= \varphi\left(S\tilde{u} + R\tilde{v} + \tilde{b}\right) \tag{4.4}$$

where $\tilde{u}$ and $\tilde{v}$ are the left and right substructures embeddings (word embeddings at the leaf node level), and S and R are the matrices defining how the left and right children's representations are to be combined.

This is a useful form as all constituency parse trees can be converted into binary parse trees, via left-factoring or right factoring (adding new nodes to the left or right to take some of the children). This is sometimes called binarization, or putting them into Chomsky normal form. This form of structured network has been used in many words, including Socher, Manning, and Ng (2010), Socher et al. (2011c), Socher et al. (2011a), Socher et al. (2011b) and Zhang et al. (2014). Notice that S and R matrices are shared for all RVs, so all substructures are composed in the same way, based only on whether they are on the left, or the right.

4.3.2 Dependency Tree

The dependency tree is the other commonly considered parse-tree. Structured networks based upon the dependency tree have been used by Socher et al. (2014), Iyyer et al. (2014), and Iyyer, Boyd-Graber, and Daumé III (2014). In these works rather than a using composition matrix for left-child and right-child, the composition matrix varies depending on the type of relationship of between the head word and its child. Each dependency relationship type has its own composition matrix. That is to say there are distinct composition matrices for each of `nsub`, `det`, `nmod`, `case` etc. This allows for multiple inputs to a single head node to be distinguished by their relationship, rather than their order. This is important for networks using a dependency parse tree structure as the relationship is significant, and the structure allows a node to have any number of inputs.

Consider a function $\pi(i, j)$ which returns the relationship between the head word at position i and the child word at position j. For example, using the tree shown in Figure 4.5, which has $w^8 =$ `parse` and $w^9 =$ `tree` then $\pi(8, 9) =$ `compound`. This is used to define the composed representation for each RV:

$$f^{\text{RV}}(i) = \varphi\left(W^{\text{head}} C_{:,w^i} + \sum_{j \in \text{children}(i)} W^{\pi(\text{i,j})} f_{RV}(j) + \tilde{b}\right) \tag{4.5}$$

Here $C_{:,w^i}$ is the word embedding for w^i, and W^{head} encodes the contribution of the headword to the composed representation. Similarly, $W^{\pi(\text{i,j})}$ encodes the contribution of the child words. Note that the terminal case is just $f_{RV}(i) = \varphi\left(W^{\text{head}} C_{:,w^i} + \tilde{b}\right)$ when a node i has no children. This use of the relationship to determine the composition matrix, increases both the networks expressiveness, and also handles the non-binary nature of dependency trees.

A similar technique could be applied to constituency parse trees. This would be using the part of speech (e.g. VBZ, NN) and phrase tags (e.g. NP, VP) for the sub-structures to choose the weight matrix. This would, however, lose the word-order information when multiple inputs have the same tag. This would be the case, for example, in the right-most branch shown in Figure 4.4, where both `parse` and `tree` have the NN POS tag, and thus using only the tags, rather than the order would leave `parse tree` indistinguishable from `tree parse`. This is not a problem for the dependency parse, as word relationships unambiguously correspond to the role in the phrase's meaning. As such, allowing the dependency relationship to define the mathematical relationship, as encoded in the composition matrix, only enhances expressibility.

For even greater capacity for the inputs to control the composition, would be to allow every word be composed in a different way. This can be done by giving the child nodes there own composition matrices, to go with there embedding vectors. The composition matrices encode the relationship, and the operation done in the composition. So not only is the representation of the (sub)phrase determined by a relationship between its constituents (as represented by their embeddings), but the nature of that relationship (as represented by the matrix) is also determined by those same constituents. In this approach at the leaf-nodes, every word not only has a word vector, but also a word matrix. This is discussed in Section 4.4.

4.3.3 Parsing

The initial work for both contingency tree structured networks (Socher, Manning, and Ng 2010) and for dependency tree structured networks (Stenetorp 2013) was on the creation of parsers. This is actually rather different to the works that followed. In other works the structure is provided as part of the input (and is found during preprocessing). Whereas a parser must induce the structure of the network, from the unstructured input text. This is simpler for contingency parsing, than for dependency parsing.

When creating a binary contingency parse tree, any pair of nodes can only be merged if they are adjacent. The process described by Socher, Manning, and Ng (2010), is to consider which nodes are to be composed into a higher level structure each in turn. For each pair of adjacent nodes, an RV can be applied to get a merged representation. A linear scoring function is also learned, that takes a merged representation and determines how good it was. This is trained such that correct merges score highly. Hinge loss is employed for this purpose. The Hinge loss function works on similar principles to negative sampling (see the motivation given in Section 2.4.2). Hinge loss is used to cause the merges that occur in the training set to score higher than those that do not. To perform the parse, nodes are merged; replacing them with their composed representation; and the new adjacent pairing score is then recomputed. Socher, Manning, and Ng (2010) considered both greedy, and dynamic programming search to determine the order of composition, as well as a number of variants to add additional information to the process. The dependency tree parser extends beyond this method.

Dependency trees can have child-nodes that do not correspond to adjacent words (non-projective language). This means that the parser must consider that any (unlinked) node be linked to any other node. Traditional transition-based dependency parsers function by iteratively predicting links (transitions) to add to the structure based on its current state. Stenetorp (2013) observed that a composed representation similar to Equation (4.4), was an ideal input to a softmax classifier that would predict the next link to make. Conversely, the representation that is suitable for predicting the next link to make, is itself a composed representation. Note, that Stenetorp (2013) uses the same matrices for all relationships (unlike the works discussed in Section 4.3.2). This is required, as the relationships must be determined from the links made, and thus are not available before the parse. Bowman et al. (2016a), presents a work an an extension of the same principles, which combines the parsing step with the processing of the data to accomplish some task, in their case detecting entailment.

4.3.4 Recursive Autoencoders

Recursive autoencoders are autoencoders, just as the autoencoder discussed in Chapter 1 of White et al. (2018a), they reproduce their input. It should be noted that unlike the encoder-decoder RNN discussed in Section 4.2.1, they cannot be trivially used to generate natural language from an arbitrary embeddings, as they require the knowledge of the tree structure to unfold into. Solving this would be the inverse problem of parsing (discussed in Section 4.3.3).

The models presented in Socher et al. (2011a) and Iyyer, Boyd-Graber, and Daumé III (2014) are unfolding recursive autoencoders. In these models an identical inverse tree is defined above the highest node. The loss function is the sum of the errors at the leaves, i.e. the distance in vector space between the reconstructed words embeddings and the input word-embeddings. This was based on a simpler earlier model: the normal (that is to say, not unfolding) recursive autoencoder.

The normal recursive autoencoder, as used in Socher et al. (2011c) and Zhang et al. (2014) only performs the unfolding for a single node at a time during training. That means that it assesses how well each merge can individually be reconstructed, not the success of the overall reconstruction. This per merge reconstruction has a loss function based on the difference between the reconstructed embeddings and the inputs embeddings. Note that those inputs/reconstructions are not word embeddings: they are the learned merged representations, except when the inputs happen to be leaf node. This single unfold loss covers the auto-encoding nature of each merge; but does not give any direct assurances of the auto-encoding nature of the whole structure. However, it should be noted that while it is not trained for, the reconstruction components (that during training are applied only at nodes) can nevertheless be applied recursively from the top layer, to allow for full reconstruction.

Semi-supervised

In the case of all these autoencoders, except Iyyer, Boyd-Graber, and Daumé III (2014), a second source of information is also used to calculate the loss during training. The networks are being simultaneously trained to perform a task, and to regenerate their input. This is often considered as semi-supervised learning, as unlabelled data can be used to train the auto-encoding part (unsupervised) gaining a good representation, and the labelled data can be used to train the task output part (supervised) making that representation useful for the task. This is done by imposing an additional loss function onto the output of the central/top node.

- In Socher et al. (2011c) this was for sentiment analysis.
- In Socher et al. (2011a) this was for paraphrase detection.
- In Zhang et al. (2014) this was the distance between embeddings of equivalent translated phrases of two RAEs for different languages.

The reconstruction loss and the supervised loss can be summed, optimised in alternating sequences, or the reconstructed loss can be optimised first, then the labelled data used for fine-tuning.

4.4 Matrix Vector Models

4.4.1 Structured Matrix Vector Model

Socher et al. (2012) proposed that each node in the graph should define not only a vector embedding, but a matrix defining how it was to be combined with other nodes. That is to say, each word and each phrase has both an embedding, and a composition matrix.

Consider this for binary constituency parse trees. The composition function is as follows:

$$f^{\mathrm{RV}}(\tilde{u}, \tilde{v}, U, V) = \varphi\left([S\ R][U\tilde{v}; V\tilde{u}] + \tilde{b}\right) \tag{4.6}$$

$$= \varphi\left(S\,U\tilde{v} + R\,V\tilde{u} + \tilde{b}\right) \tag{4.7}$$

$$F^{\mathrm{RV}}(U, V) = W\,[U; V] = W^{\mathrm{l}}U + W^{\mathrm{r}}V \tag{4.8}$$

f^{RV} gives the composed embedding, and F^{RV} gives the composing matrix. The S and R represent the left and right composition matrix components that are the same for all nodes (regardless of content). The U and V represent the per word/node child composition matrix components. We note that S and R could, in theory, be rolled in to U and R as part of the learning. The $\tilde{u}$ and $\tilde{v}$ represent the per word/node children embeddings, and W represents the method for merging two composition matrices.

We note that one can define increasingly complex and powerful structured networks along these lines; though one does run the risk of very long training times and of over-fitting.

4.4.2 Sequential Matrix Vector Model

A similar approach, of capturing a per word matrix, was used on a sequential model by Wang, Liu, and McDonald (2017). While sequential models are a special case of structured models, it should be noted that unlike the structured models discussed prior, this matrix vector RNN features a gated memory. This matrix-vector RNN is an extension of the GRU discussed in Chapter 2 of White et al. (2018a), but without a reset gate.

In this sequential model, advancing a time step, is to perform a composition. This composition is for between the input word and the (previous) state. Rather than directly between two nodes in the network as in the structural case. It should be understood that composing with the state is not the same as composing the current input with the previous input. But rather as composing the current input with all previous inputs (though not equally).

As depicted in Figure 4.6 each word, w^t is represented by a word embedding $\tilde{x}^t$ and matrix: $\tilde{X}^{w^t}$, these are the inputs at each time step. The network outputs and states are the composed embedding $\hat{y}^t$ and matrix Y^t.

$$h^t = \tanh\left(W^{\mathrm{h}}[x^t; \hat{y}^{t-1}] + \tilde{b}^{\mathrm{h}}\right) \tag{4.9}$$

$$z^t = \sigma\left(Y^{t-1}x^t + X^t\hat{y}^{t-1} + \tilde{b}^{\mathrm{z}}\right) \tag{4.10}$$

$$\hat{y}^t = z^t \odot h^t + (1 - z^t) \odot \hat{y}^{t-1} \tag{4.11}$$

$$Y^t = \tanh\left(W^{\mathrm{Y}}[Y^{t-1}; X^t] + \tilde{b}^{\mathrm{Y}}\right) \tag{4.12}$$

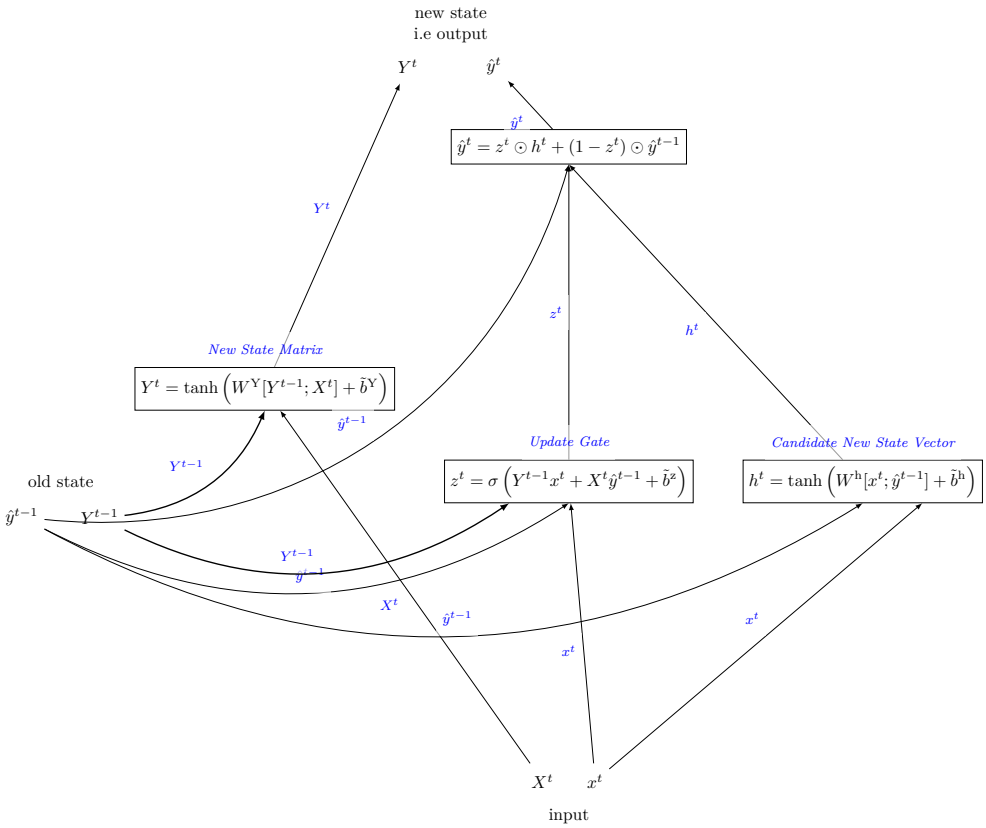
The matrices W^{h}, W^{Y} and the biases $\tilde{b}^{\mathrm{h}}$, $\tilde{b}^{\mathrm{z}}$, $\tilde{b}^{\mathrm{Y}}$ are shared across all time steps/compositions. W^{Y} controls how the next state-composition Y^t matrix is generated from its previous value and the input composition matrix, X^t; W^{h} similarly controls the value of the candidate state-embedding h^t.

h^t is the candidate composed embedding (to be output/used as state). z_t is the update gate, it controls how much of the actual composed embedding ($\hat{y}^t$) comes from the candidate h^t and how much comes from the previous value ($\hat{y}^{t-1}$). The composition matrix Y^t (which is also part of the state/output) is not gated.

Notice, that the state composition matrix Y^{t-1} is only used to control the gate z^t, not to directly affect the candidate composited embedding h^t. Indeed, in fact one can note that all similarity to the structural method of Socher et al. (2012) is applied in the gate z^t. The method for calculating h^t is similar to that of a normal RU.

The work of Wang, Liu, and McDonald (2017), was targeting short phrases. This likely explains the reason for not needing a forget gates. The extension is obvious, and may be beneficial when applying this method to sentences

Figure 4.6: A Matrix Vector recurrent unit



4.5 Conclusion, on compositionality

It is tempting to think of the structured models as compositional, and the sequential models as non-compositional. However, this is incorrect.

The compositional nature of the structured models is obvious: the vector for a phrase is composed from the vectors of the words that the phrase is formed from.

Sequential models are able to learn the structures. For example, learning that a word from n time steps ago is to be remembered in the RNN state, to then be optimally combined with the current word, in the determination of the next state. This indirectly allows the same compositionality as the structured models. It has been shown that sequential models are indeed in-practice able to learn such relationships between words (White et al. 2017). More generally as almost all aspects of language have some degree of compositionality, and sequential models work very well on most language tasks, this implicitly shows that they have sufficient representational capacity to learn sufficient degrees of compositional processing to accomplish these tasks.

In fact, it has been suggested that even some unordered models such as sum of word embeddings are able to capture some of what would be thought of as compositional information. Ritter et al. (2015) devised a small corpus of short sentences describing containing relationships between the locations of objects. The task and dataset was constructed such that a model must understand some compositionality, to be able to classify which relationships were described. Ritter et al. (2015) tested

several sentence representations as the input to a naïve Bayes classifier being trained to predict the relationship. They found that when using sums of high-quality word embeddings as the input, the accuracy not only exceeded the baseline, but even exceeded that from using representation from a structural model. This suggests that a surprising amount of compositional information is being captured into the embeddings; which allows simple addition to be used as a composition rule. Though it being ignorant of word order does mean it certainly couldn't be doing so perfectly, however the presence of other words my be surprisingly effective hinting at the word order (White et al. 2016b), thus allow for more apparently compositional knowledge to be encoded than is expected.

To conclude, the compositionality capacity of many models is not as clear cut as it may initially seem. Further to that the requirement for a particular task to actually handle compositional reasoning is also not always present, or at least not always a significant factor in practical applications. We have discussed many models in this section, and their complexity varies significantly. They range from the very simple sum of word embeddings all the way to the the structured matrix models, which are some of the more complicated neural networks ever proposed.

Part II

Publications

Chapter 5

Learning of Colors from Color Names: Distribution and Point Estimation

This paper is currently under review for Computational Linguistics.

Abstract

Color names are often made up of multiple words. As a task in natural language understanding we investigate in depth the capacity of neural networks based on sums of word embeddings (SOWE), recurrence (LSTM and GRU based RNNs) and convolution (CNN), to estimate colors from sequences of terms. We consider both point and distribution estimates of color. We argue that the latter has a particular value as there is no clear agreement between people as to what a particular color describes – different people have a different idea of what it means to be "very dark orange", for example. Surprisingly, despite it's simplicity, the sum of word embeddings generally performs the best on almost all evaluations.

5.1 Introduction

Consider that the word `tan` may mean one of many colors for different people in different circumstances: ranging from the bronze of a tanned sunbather, to the brown of tanned leather; `green` may mean anything from `aquamarine` to `forest green`; and even `forest green` may mean the rich shades of a rain-forest, or the near grey of Australian bush land. Thus the *color intended* cannot be uniquely inferred from a color name. Without further context, it does nevertheless remain possible to estimate likelihoods of which colors are intended based on the population's use of the words.

Color understanding, that is, generating color from text, is an important subtask in natural language understanding. For example, in a natural language enabled human-machine interface, when asked to select the `dark bluish green` object, it would be much useful if we could rank each object based on how likely its color matches against a learned distribution of the color name `dark bluish green`. This way if the most-likely object is eliminated (via another factor), the second most likely one can be considered. A threshold can be set to terminate the search. This kind

of likelihood-based approach is not possible when we have only exact semantics based on point estimates.

Color understanding is a challenging domain, due to high levels of ambiguity, the multiple roles taken by the same words, the many modifiers, and the many shades of meaning. In many ways it is a grounded microcosm of natural language understanding. Due to its difficulty, texts containing color descriptions such as `the flower has petals that are bright pinkish purple with white stigma` are used to demonstrate the capability of the-state-of-the-art image generation systems (**reed2016generative**; **2015arXiv151102793M**). The core focus of the work we present is to map from the short-phrase descriptions of a color, to representation in a color-space such as HSV (**smith1978color**). The HSV color space is a grounded meaning space for the short phrase. Due to this grounding, and the aforementioned linguistic phenomena, this is a particularly interesting short phrase understanding task. Issues of illumination and perceived color based on context are considered out of the scope of this article.

5.1.1 Distribution vs. Point Estimation

As illustrated, proper understanding of color names requires considering *the color intended* as a random variable. In other words, a color name should map to a distribution, not just a single point or region. For a given color name, any number of points in the color-space could be intended, with some being more or less likely than others. Or equivalently, up to interpretation, it may intend a region but the likelihood of what points are covered is variable and uncertain. This distribution is often multimodal and has a high and asymmetrical variance, which further renders regression to a single point unsuitable.

A single point estimate does not capture the diverse nature of the color names adequately. Moreover, it is impossible to find the single best point estimation method. For example: for a bimodal distribution, using the distribution's mean as a point estimate will minimize the total squared error, but it will select a point in the valley between the peaks, which is less likely and less meaningful as a characterisation of that color. Similarly for an asymmetrical distribution, where the mean will be off to one side of the peak. Conversely, using the modes of the distribution the highest (most likely) peaks will be selected, but will on average be more incorrect as measured by the mean squared error. The correct trade-off, if a point estimate is required, depends on the final use of the system. Another problem is that point estimates do not capture the sensitivity. In an asymmetrical distribution, having a point slightly off-centre in one direction may result in a very different probability, this more generally holds for a narrow variance distribution. Conversely for a very wide variance distribution (for example one approaching the uniform distribution) the point estimate value may matter very little with all points providing similar probabilities. Color distributions are almost always multimodal or asymmetrical, and exhibit widely differing variances for different names. This can be seen in the histograms of the training data shown in Figure 5.7 in Section 5.6.1. Note that while only a small (but particularly interesting) set of colors demonstrate multimodality in

the hue channel, as noted by **mcmahan2015bayesian**, when considering all channels the other problematic features abound. Asymmetry in-particular is ubiquitous in the value and saturation channels. Given these issues, producing a point estimate has only limited value: estimating a distribution is a more general task. However we do consider the point estimation task, as it allows contrast in assessing the input module (SOWE/CNN/GRU/LSTM) of our proposed methods across the two different output modules (distribution/point estimation).

Generation of color from text has not received much attention in prior work. To the best of our knowledge, the only similar work is **DBLP:journals/corr/** which only considers point estimation, and uses a dataset containing far too few observations to allow for learning probability distributions from population usages of the color names. **DBLP:journals/corr/KawakamiDRS16** uses a character sequence based model, rather than a word sequence model, which is inline with the very small amount of training data for each color name they have. To our knowledge the generation of probability distributions in a color-space, from color names as considered as sequences of words, has not been investigated at all by any prior work. This paper is the first investigation of such kind. There have been several works on the reverse problem (**mcmahan2015bayesian**; **meomcmahanstone:color**; Monroe, Goodman, and Potts 2016): the generation of a textual name for a color from a point in a color-space. From these works on the reverse problem, there is a clear trend towards data-driven approaches in recent years where more color names and observations are used. This motivates our own data-driven approach presented in this paper.

5.1.2 Contributions

Problem statement: given a set of $\langle$color-name, $(h, s, v)\rangle$ pairs, we need to learn a mapping from any color-name, seen or unseen, to a color-value or a distribution in HSV color space.

We propose a neural network based architecture that can be broken down into an **input module**, which learns a vector representation of color-names, and a linked **output module**, which produces either a probability distribution or a point estimate. The **output module** uses a softmax output layer for probability distribution estimation, or a novel HSV output layer for point estimation. To carry out the representational learning of color-names, four different color-name embedding learning models are investigated for use in the **input module**: Sum Of Word Embeddings (SOWE), Convolutional Neural Network (CNN) and two types of Recurrent Neural Network (LSTM and GRU RNNs). All four input modules use pretrained FastText embeddings (Bojanowski et al. 2017) to represent the individual tokens making up the color names, but combine them using difference mechanisms. The capacity of these input models is of primary interest to this work.

To evaluate and compare the three learning models, we designed a series of experiments to assess their capability in capturing compositionality of language used in color names. These include: **(1)** evaluation on all color names (full task); **(2)** evaluation on color names when the order of the

words matters (order task); **(3)** evaluation on color names which never occur in the training data in that exact form, but for which all terms occur in the training data (unseen combination task); **(4)** qualitative demonstration of outputs for color names with terms which do not occur in the training data at all, but for which we know their word embeddings (embedding only task).

To express the estimated distribution for the output module, we discretize the HSV color-space to produce a histogram. This allows us to take advantage of the well-known softmax based methods for the estimation of a probability mass distribution using a neural network. An interesting challenge when considering this discretization is the smoothness of the estimate. The true space is continuous, even if we are discretizing it at a resolution as high as the original color monitors used to collect the data. Being continuous means that a small change in the point location in the color-space should correspond to a small change in how likely that point is according to the probability distribution. Informally, this means the histograms should look smooth, and not spiky. We investigated using a Kernel Density Estimation (KDE) based method for smoothing the training data, and further we conclude that the neural networks learn this smoothness.

We conclude that the simplest SOWE model is generally the best model for all tasks both for distribution and point estimation. It is followed closely by the CNN; with the RNNs both performing significantly worse (see Section 5.6). We believe that due to the nature of color understanding as a microcosm of natural language understanding, the results of our investigations have some implications for the capacity of how these models can be used for representing language compositionality in short phrase understanding.

5.2 Related Work

The understanding of color names has long been a concern of psycholinguistics and anthropologists (**berlin1969basic**; **heider1972universals**; **HEIDER1972337**; **mylonas2015use**). It is thus no surprise that there should be a corresponding field of research in natural language processing.

The earliest works revolve around explicit color dictionaries. This includes the ISCC-NBS color system (**kelly1955iscc**) of 26 words, that are composed according to a context free grammar, such that phrases are mapped to single points in the color-space; and the simpler, non-compositional, 11 basic colors of **berlin1969basic**. Works including **Berk:1982:HFS:358589.358606**; **conway1992experimental**; **ele1994computational**; **mojsilovic2005computational**; **menegaz2007discrete**; **van2009learning** propose methods for the automatic mapping of colors to and from these small manually defined sets of color names. We note that **menegaz2007discrete**; **van2009learning** both propose systems that discretize the color-space, though to a much coarser level than we consider in this work.

The large Munroe dataset (**Munroe2010XKCDdataset**), has allowed a data driven approach to natural language color problems. In-contrast

to earlier manually defined color dictionaries, it has a large number of colors, a non-trivial vocabulary, and is sourced from a survey of hundreds of thousands of respondents. Full details on this dataset can be found in Section 5.4.4. The availability of a large color corpus has allowed machine learning based methods to be used in recent works including **mcmahan2015bayesian**; **meomcmahanstone:color**; **acl2018WinnLighter**; Monroe, Goodman, and Potts (2016) and this article.

mcmahan2015bayesian and **meomcmahanstone:color** present a Bayesian method for color estimation and color naming. Their work primarily focuses on mapping from colors to to their exact names, the reverse of our task. While their method is reversible: to go from exact color names to probabilities, they do not present any evaluations of this. These works are based on defining fuzzy rectangular distributions in the color-space to cover the distribution estimated from the data, which are used in a Bayesian system to non-compositionally determine the color name. This work focuses only on exact color names, where as later works consider the sequential nature of multi-word color names.

Monroe, Goodman, and Potts (2016) map a point in the color-space, to a sequence of probability estimates over color words. They extend beyond all prior color naming systems to produce a compositional color namer based on the Munroe dataset. Their method uses a recurrent neural network (RNN), which takes as input a color-space point, and the previous output word, and gives a probability of the next word to output – this is a conditional language model. We tackle the inverse problem, natural language understanding rather than generation. Our distribution estimation models map from a sequence of terms, to a distribution in color-space. Similarly, our point estimation models map from a sequence of terms to a single point in color-space.

DBLP:journals/corr/KawakamiDRS16 proposes another compositional color naming model. They use a per-character RNN and a variational autoencoder approach. It is in principle very similar to Monroe, Goodman, and Potts (2016), but functioning on a character, rather than word level. The work by Kawakami et al. also includes a method for generating colors. However they only consider the generation of point estimates, rather than distributions. The primary focus of our work is on generating distributions. The datasets used by Kawakami et al. contain only very small numbers of observations for each color name (often just one). These datasets are thus not suitable for modeling the distribution in color-space as interpreted by a population. Further, given the very small number of examples they are not well suited for use with word-based modeling: the character based modeling employed by Kawakami et al. is much more suitable. As such, we do not attempt to compare with their work.

DBLP:journals/corr/MonroeHGP17 present a neural network solution to a communication game, where a speaker is presented with three colors and asked to describe one of them, and the listener is to work out which color is being described. Speaker and listener models are trained, using LSTM-based decoders and encoders, respectively. The final time-step of their model produces a 100 dimensional representation of the description provided. From this, a Gaussian distributed score function

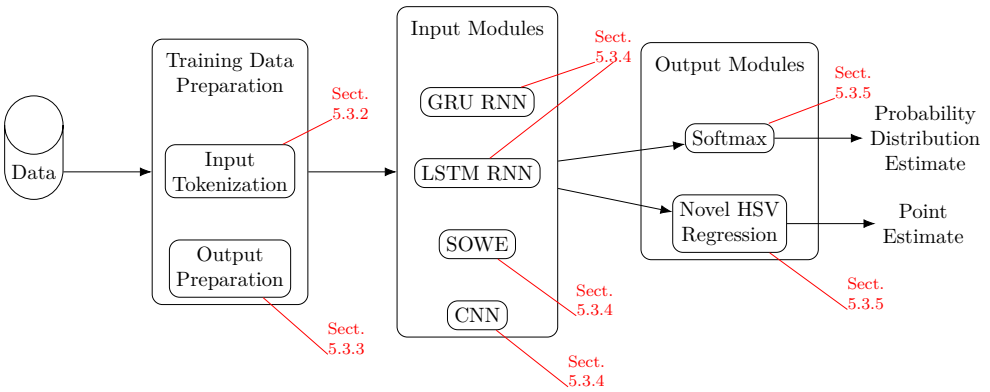


Figure 5.1: The overall architecture of our system.

is calculated, over a high dimensional color-space defined by Monroe, Goodman, and Potts (2016), which is then used to score each of the three options. While this method does work with a probability distribution, as a step in its goal, this distribution is always both symmetric and unimodal – albeit in a high-dimensional color-space.

acl2018WinnLighter demonstrates a neural network for producing directional vectors in a color space indicating how comparatives such as `lighter` and `darker` change a color. This effectively creates a ray (half-line) in color space along which possible colors described lie. Their networks takes as its inputs a word embedding for a comparative adjective, and a point in RGB color-space. It outputs a directional vector in the RGB space along which occurs the possible modified version of the input color point according to the given adjective. The magnitude of this directional vector is trained such that adding it to the source color point, will give a good point estimate of the modified color. For example mapping from `green` to a `darker green` is: $((164, 227, 77), \text{darker}) \mapsto (141, 190, 61)$ for a point estimate. When using it for a ray estimate it is the half line from the first, through the second point, where every point further along the ray is `darker` than the earlier point. The color adjectives may have up to two words, to allow for expressions such as `more neon`. This is allowed by taking as a fixed sized input of two embeddings – when only one input is required, the other is replaced by a zero vector. Their training and evaluation is based on data sourced from the Munroe dataset.

5.3 Method

5.3.1 System Architecture

Our overall system architecture for all models is shown in Figure 5.1. This shows how color names are transformed into distribution or point estimates over the HSV color-space.

5.3.2 Input Data Preparation

We desire a color prediction model which takes as input a sequence of words that make up the color's name rather than simply mapping from

the whole phrase (whole phrase mapping does not scale to new user input, given the combinatorial nature of language). Towards this end, color names are first tokenized into individual words. For the input into our neural network based models, these words are represented with pretrained word embeddings.

Tokenization

During tokenization a color name is split into terms with consistent spelling. For example, `bluish kahki` would become the sequence of 3 tokens: [`blue`, `ish`, `khaki`]. Other than spelling, the tokenization results in the splitting of affixes and combining tokens (such as hyphens). Combining tokens and related affixes affect how multiple colors can be combined. The full list of tokenization rules can be found in the accompanying source code. Some further examples showing how combining tokens and affixes are used and tokenized:

- `blue purple` $\mapsto$ [`blue`, `purple`].
- `blue-purple` $\mapsto$ [`blue`, `-`, `purple`].
- `bluish purple` $\mapsto$ [`blue`, `ish`, `purple`]
- `bluy purple` $\mapsto$ [`blue`, `y`, `purple`]
- `blurple` $\mapsto$ [`blue-purple`]

The final example of `blurple` $\mapsto$ [`blue-purple`] is a special-case. It is the only portmanteau in the dataset, and we do not have a clear way to tokenize it into a series of terms which occur in our pretrained embedding's vocabulary (see Section 5.3.2). The portmanteau `blurple` is not in common use in any training set used for creating word embeddings, so no pretrained embedding is available.[1] As such we handle it by treating it as the single token `blue-purple` for purposes of finding an embedding. There are many similar hyphenated tokens in the pretrained embeddings vocabulary, however (with that exception) we do not use them as it reduces the sequential modeling task to the point of being uninteresting.

Embeddings

All our neural network based solutions incorporate an embedding layer. This embedding layer maps from tokenized words to vectors. We make use of 300 dimensional pretrained FastText embeddings (Bojanowski et al. 2017)[2].

The embeddings are not trained during the task, but are kept fixed. As per the universal approximation theorem (**leshno1993uat**; **SONODA2017uat**) the layers above the embedding layer allow for arbitrary continuous transformations. By fixing the embeddings, and learning this transformation, we can produce estimates of colors from embeddings alone, without any training data at all, as shown in Section 5.6.4.

[1] Methods do exist to generate embeddings for out of vocabulary words (like `blurple`), particularly with FastText embeddings (Bojanowski et al. 2017). But we do not investigate those here.

[2] Available from `https://fasttext.cc/docs/en/english-vectors.html`

5.3.3 Output Data Preparation for Training Distribution Estimation Models

To train the distribution estimation models we need to preprocess the training data into a distribution. The raw training data itself, is just a collection of $\langle$color-name, $(h, s, v)\rangle$ pairs – samples from the distributions for each named-color. This is suitable for training the point estimation models, but not for the distribution estimation models . We thus convert it into a tractable form, of one histogram per output channel – by assuming the output channels are conditionally independent of each other.

Conditional Independence Assumption

We make the assumption that given the name of the color, the distribution of the hue, saturation and value channels are independent. That is to say, it is assumed if the color name is known, then knowing the value of one channel would not provide any additional information as to the value of the other two channels. The same assumption is made, though not remarked upon, in **meomcmahanstone:color** and **mcmahan2015bayesian**. This assumption of conditional independence allows considerable saving in computational resources. Approximating the 3D joint distribution as the product of three 1D distributions decreases the space complexity from $O(n^3)$ to $O(n)$ in the discretized step that follows.

Superficial checks were carried out on the accuracy of this assumption. Spearman's correlation on the training data suggests that for over three quarters of all color names, there is only weak correlation between the channels for most colors (Q3 = 0.187). However, this measure underestimates correlation for values which have a circular relative value, such as hue. Of the 16 color-spaces evaluated, HSV had the lowest correlation by a large margin. Full details, including the table of correlations, are available in supplementary materials (Section 5.8.1). These results are suggestive, rather than solidly indicative, on the degree of correctness of the conditional independence assumption. We consider the assumption sufficient for this investigation; as it does not impact on the correctness of results. A method that does not make this assumption may perform better when evaluated using the same metrics we use here.

Discretization

For distribution estimation, our models are trained to output histograms. This is a discretized representation of the continuous distribution. Following standard practice, interpolation-based methods can be used to handle it as a continuous distribution. By making use of the conditional independence assumption (see Section 5.3.3), we output one 256-bin histogram per channel. We note that 24-bit color (as was used in the survey that collected the dataset) can have all the information captured by a 256 bin discretization per channel. 24 bit color allows for a total of 2^{24} colors to be represented, and even one-hot encoding for each of the 256 bin discretization channels allows for the same. As such there is no meaningful loss of information during the discretization step when using histograms

over a truly continuous estimation method, such as a Gaussian mixture model. Although such models may have other advantages (such as the apriori information added by specifying the distribution), we do not investigate them here, instead considering the simplest non-parametric estimation model (the histogram), which has the simple implementation in a neural network using a softmax output layer.

Discretizing the data in this way is a useful solution used in several other machine learning systems. **oord2016pixel**; **DBLP:journals/corr/OordDZSVG** apply a similar discretization step and found their method to outperform the more complex truly continuous distribution outputting systems.

For training purposes we thus convert all the observations into histograms. One set of training histograms is produced per color description present in the dataset – that is to say a training histogram is created for `brownish green`, `greenish brown`, `green` etc. We perform uniform weight attribution of points to bins as described by **jones1984remark**. In-short, this method of tabulation is to define the bins by their midpoints, and to allocate probability mass to each bin based on how close an observe point is to the adjacent midpoints. A point precisely on a midpoint would have all its probability mass allocated to that bin; whereas a point halfway between midpoints would have 50% of its mass allocated to each. For example were we to have bins with midpoints at 1 and 3: then observation at 2, then 50% of probability mass for this observation would be allocated to the bin with midpoint 1, and 50Whereas if there observation was at 2.5 then 25% of its mass would be allocated to the bin at 1, and 25% of its mass to the bin at 3. Our bins are at much finer resolution than this example, dividing the space between 0 and 1 into 256 bins. This form of tabulation commonly used during the first step of performing kernel density estimation, prior to the application of the kernel.

5.3.4 Color Name Representation Learning (Input Modules)

For each of the models investigated we define an input module. This module handles the input of the color name, which is provided as a sequence of tokens. It produces a fixed sized dense representation of the color name, which is the input to the output module Section 5.3.5). In all models the input and output modules are trained concurrently as a single system.

Recurrent Neural Networks (GRU and LSTM RNNs)

A Recurrent Neural Network (RNN) is a common choice for this kind of task, due to the variable length of the input. We consider two ”flavours” of RNN: Gated Recurrent Unit (GRU) networks (**cho2014properties**), and Long Short Term Memory (LSTM) networks (**gers1999learning**; Hochreiter and Schmidhuber 1997). They differ only in their recurrent unit's internal structure. The general structure of both input modules is shown in Figure 5.2. It is similar to Monroe, Goodman, and Potts (2016), or indeed to most other word sequence learning models. Each word is first transformed to an embedding representation. This representation is trained with the rest of the network allowing per word information to be

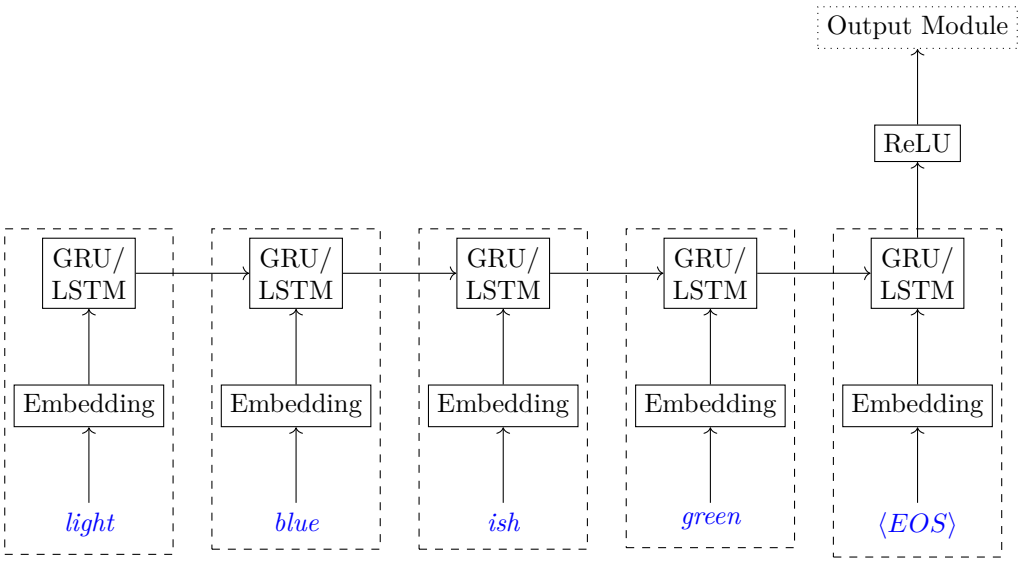


Figure 5.2: The LSTM/GRU Input module for the example input `light greenish blue`. Each dashed box represents 1 time-step. .

efficiently learned. The embedding is used as the input for the recurrent unit, either a GRU or an LSTM depending on the model considered. The stream is terminated with an End of Stream (`<EOS>`) pseudo-token, represented using a zero-vector. The output of the last time-step is fed to a Rectified Linear Unit (ReLU), and then to the output model.

During preliminary investigations we also considered a vanilla RNN (that is to say one without any gating). Early results on the development set suggested that it performed only marginally worse than the GRU or LSTM networks. That it does not perform much worse than the models with features to improve memory is unsurprising, as the color names have at most 5 terms. We constrained our full investigation to the more popular GRU and LSTM networks.

Sum of Word Embeddings (SOWE)

Using a simple sum of word embeddings as a layer in a neural network is less typical than an RNN structure, though it is well established as a useful representation, and has been used as an input to other classifiers such as support vector machines (e.g. as in White et al. (2015) and White et al. (2018b)). Any number of word embeddings can be added to the sum, thus allowing it to handle sequences of any length. However, it has no representation of the order. The structure we used is shown in Figure 5.3.

Convolutional Neural Network(CNN)

A convolutional neural network (shown in Figure 5.4) can be applied to the task by applying 2D convolution over the stacked word embeddings. We use 64 filters of size between one and five. Five is number of tokens in the longest color-name, so this allows it to learn full length relations.

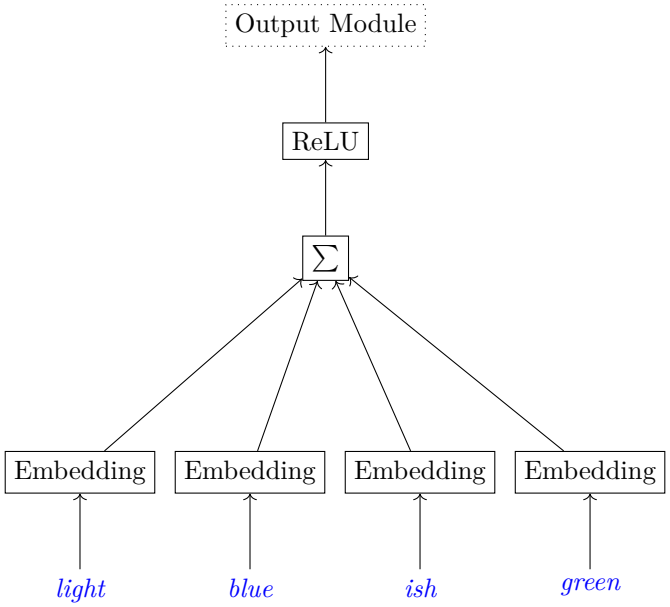


Figure 5.3: The SOWE input module for the example input `light bluish green`

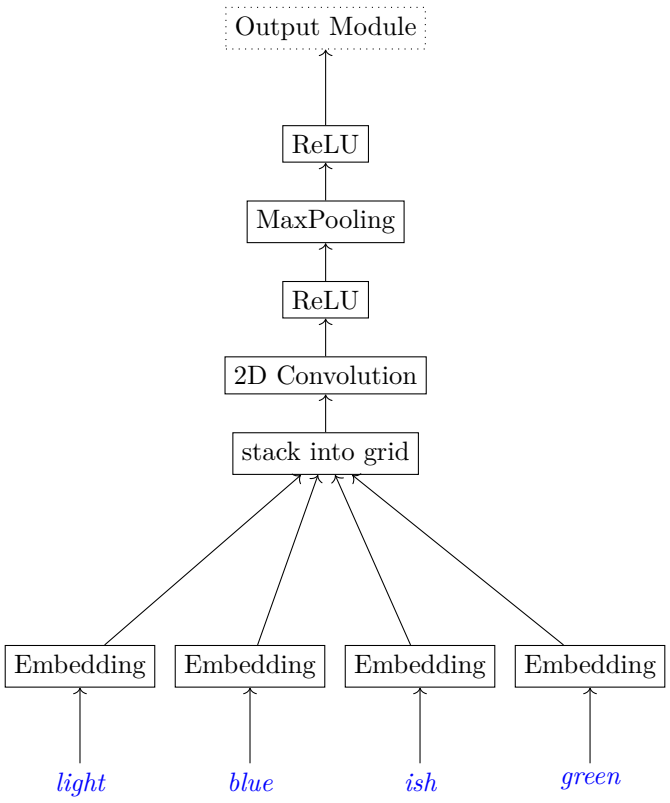


Figure 5.4: The CNN input module for the example input `light bluish green`.

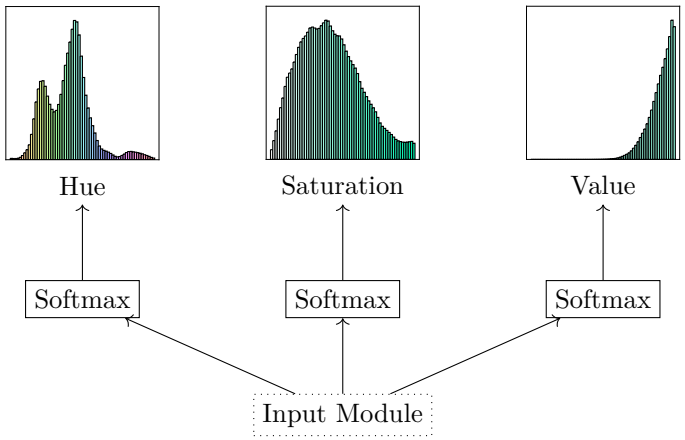


Figure 5.5: The Distribution Output Module

5.3.5 Distribution and Point Estimation (Output Modules)

On top of the input module, we define an output module to suit the neural network for the task of either distribution estimation or point estimation. The input module defines how the terms are composed into the network. The output module defines how the network takes its hidden representation and produces an output.

Distribution Estimation

The distributions are trained to produce the discretized representation as discussed in Section 5.3.3. Making use of the conditional independence assumption (see Section 5.3.3), we output the three discretized distributions. As shown in Figure 5.5, this is done using three softmax output layers – one per channel. They share a common input, but have separate weights and biases. The loss function is given by the sum of the cross-entropy for each of the three softmax outputs.

Point Estimation

Our point estimation output module is shown in Figure 5.6. The hidden-layer from the top of the input module is fed to four single output neurons.[3] Two of these use the sigmoid activation function (range 0:1) to produce the outputs for the saturation and value channels. The other two use the tanh activation function (range -1:1), and produce the intermediate output that we call y_{shue} and y_{chue} for the sine and cosine of the hue channel respectively. The hue can be found as $y_{hue} = \text{atan2}\,(y_{shue}, y_{chue})$. We use the intermediate values when calculating the loss function. During training we use the following loss function for each observation $y^\star$,

[3]Equivalently these four single neurons can be expressed as a layer with four outputs and two different activation functions.

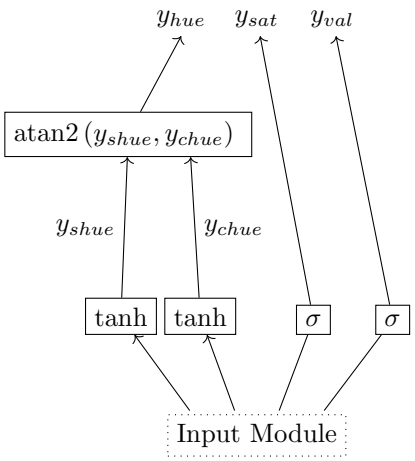


Figure 5.6: The Point Estimate Output Module. Here atan2 is the quadrant preserving arctangent, outputting the angle in turns.

and each corresponding prediction y.

$$\begin{aligned} loss = &\frac{1}{2}\left(\sin(y^{\star}_{hue}) - y_{shue}\right)^2 \\ &+ \frac{1}{2}\left(\cos(y^{\star}_{hue}) - y_{chue}\right)^2 \\ &+ \left(y^{\star}_{sat} - y_{sat}\right)^2 \\ &+ \left(y^{\star}_{val} - y_{val}\right)^2 \end{aligned} \tag{5.1}$$

The mean of this loss is taken over all observations in each mini-batch during training. This loss function is continuous and correctly handles the wrap-around nature of the hue channel (**WhiteRepresentingAnglesSE**).

5.4 Evaluation

5.4.1 Perplexity in Color-Space

Perplexity is a measure of how well the distribution, estimated by the model, matches the reality according to the observations in the test set. Perplexity is commonly used for evaluating language models. Here however, it is being used to evaluate the discretized distribution estimate. It can be loosely thought of as to how well the model's distribution does in terms of the size of an equivalent uniform distribution. Note that this metric does not assume conditional independence of the color channels.

Here τ is the test-set made up of pairs consisting of a color name t, and a color-space point $\tilde{x}$; and $p(\tilde{x} \mid t)$ is the output of the evaluated model. Perplexity is defined as:

$$PP(\tau) = \exp_2\left(\left(\frac{-1}{|\tau|}\sum_{\forall(t,(\tilde{x}))\in\tau}\log_2 p(\tilde{x} \mid t)\right)\right) \tag{5.2}$$

As the perplexity for a high-resolution discretized model will inherently be very large and difficult to read, we define the standardized perplexity: $\frac{PP(\tau)}{n_{res}}$, where n_{res} is the total number of bins in the discretization

scheme. For all the results we present here $n_{res} = 256^3$. This standardized perplexity gives the easily interpretable values *usually* between zero and one. It is equivalent to comparing the relative performance of the model to that of a uniform distribution of the same total resolution. $\frac{PP(\tau)}{n_{res}} = 1$ means that the result is equal to what we would see if we had distributed the probability mass uniformly into all bins in a 3D histogram. $\frac{PP(\tau)}{n_{res}} = 0.5$ means the result is twice as good as if we were to simply use a uniform distribution: it is equivalent to saying that the correct bin is selected as often as it would be had a uniform distribution with half as many bins been used (i.e. larger bins with twice the area). The standardized perplexity is also invariant under different output resolutions. Though for brevity we only present results with 256 bins per channel, our preliminary results for using other resolutions are similar under standardized perplexity.

5.4.2 Angularly Correct Calculations on HSV

We use the HSV color-space (**smith1978color**) throughout this work. In this format: hue, saturation and value all range between zero and one. Note that we measure hue in *turns*, rather than the more traditional degrees, or radians. Having hue measured between zero and one, like the other channels, makes the modeling task more consistent. Were the hue to range between 0 and 2π (radians) or between 0 and 360 (degrees) it would be over-weighted in the loss function and evaluation metrics compared to the other channels. This regular space means that errors on all channels can be considered equally. Unlike many other colors spaces (CIELab, Luv etc.) the gamut is square and all combinations of values from the different channels correspond to realizable colors.

When performing calculations with the HSV color-space, it is important to take into account that hue is an angle. As we are working with the color-space regularized to range between zero and one for all channels, this means that a hue of one and a hue of zero are equivalent (as we measure in turns, in radians this would be 0 and 2π).

The square error of two hue values is thus calculated as:

$$SE(h_1, h_2) = \min\left((h_1 - h_2)^2 , (h_1 - h_2 - 1)^2 \right) \tag{5.3}$$

This takes into account that the error can be calculated clockwise or counter-clockwise; and should be the smaller. Note that the -1 term is related to using units of turns, were we using radians it would be -2π.

The mean of a set of hues ($\{h_1, \ldots, h_N\}$) is calculated as:

$$\bar{h} = \operatorname{atan2}\left(\frac{1}{N}\sum_{i=1}^{i=N} \sin(h_i), \frac{1}{N}\sum_{i=1}^{i=N} \cos(h_i) \right) \tag{5.4}$$

This gives the mean angle.

5.4.3 Non-compositional Baselines

We consider a non-compositional model to establish a baseline on the color modeling part of this task; with the exclusion of the language understanding part.

The non-compositional methods do not process each term in the name; they do not work with the language at all. They simply map from the exact input text (no tokenization) to the pre-calculated distribution or mean of the training data for the exact color name. As there is plenty of training data for most color names (see Section 5.4.4) this is a very effective approach. Strictly speaking, this non-compositional baseline has less information than the neural network models as it does not have the tokenized color name given to it, but only the whole name. However, pragmatically learning to compose the sequence of terms into a meaningful whole is by far the harder part of this task. This non-compositional baseline bypasses the compositional language understanding part of the process. It is as if the input module (as discussed in Section 5.3.4) would perfectly resolve the sequence of terms into a single item. These models can exploit the training observations without the need to determine how to compose the tokens. This is a useful baseline, as our neural models (SOWE, CNN, GRU and LSTM) each differs in how they compose the tokens, and on that this study focuses.

In theory the term-based neural models shoukd out-perform the non-compositional baseline, if they learn a very good compositional understanding of the language. This would require learning how the terms in the color name combine in a way that exceeds the information directly present in the training data per class. *It is this capacity of learning how the terms combine that allow for the models to predict the outputs for combinations of terms that never occur in the training data (Section 5.4.4).* Learning a compositional model that exploits its term based knowledge in such a way that generalizes to get better results than the direct exploitation of the training data (as in the non-compositional baseline), is very difficult and would require very well calibrated control of (over/under)fitting. This is particularly true in the case where there is a large amount of training data for the whole phrase. Conversely, when there is no training data for the whole phrase (as considered in Section 5.4.4) non-compositional models can not function at all.

Non-compositional Baseline for Distribution Estimation: KDE

To define a non-compositional baseline for the distribution estimation tasks, we use kernel-density estimation (KDE) in a formulation for non-parametric estimation (**silverman1986density**) . The KDE effectively produces a smoothed histogram from the training data as processed in Section 5.3.3. It causes adjacent bins to have most similar probabilities, thus matching to the mathematical notion of a continuous random variable. This is applied on-top of the histogram used for the training data. We use the Fast Fourier Transform (FFT) based KDE method of the **silverman1982algorithm**. We use a Gaussian kernel, and select the bandwidth per color description based on leave-one-out cross validation on the training data. A known issue with the FFT-based KDE method is that it has a wrap-around effect near the boundaries, where the probability mass that would be assigned outside the boundaries is instead assigned to the bin on the other side. For the value and saturation channels we follow the standard solution of initially defining additional bins outside the true boundaries, then discarding those bins and rescaling the

probability to one. For the hue channel this wrap-around effect is exactly as desired.

In our evaluations using KDE rather than just the training histograms directly proved much more successful on all distribution estimation tasks. This is because it avoids empty bins, and effectually interpolates probabilities between observations. We found in preliminary investigations that using KDE-based method to be much better than add-one smoothing.

We also investigated the application of KDE to the training data, before training our term-based neural network based distribution models. Results for this can be found in Section 5.8.2. In brief, we found that smoothing the training data does not significantly affect the result of the neural network based models. As discussed in Section 5.6.2, this is because the neural networks are able to learn the smoothness relationship of adjacent bins.

Our KDE-based non-compositional baseline for distribution estimation bypasses the natural language understanding part of the task, and directly uses the standard non-parametric probability estimation method to focus solely on modeling the distributions. Matching its performance indicates that a model is effectively succeeding well at both the natural language understanding component and the distribution estimation component.

Non-Compositional Baseline for Point Estimation: Mean-point

In a similar approach, we also propose a method that directly produces a point estimate from a color name. We define this by taking the mean (centroid) of all the training observations for a given exact color name. The mean is taken in the angularly correct way (as discussed in Section 5.4.2). Taking the mean of all the observations gives the theoretically optimal solution to minimize the squared error on the training data set. As with our direct distribution estimation method, this bypasses the term based language understanding, and directly exploits the training data. It thus represents an approximate upper bound on the point estimation performance of the term based models. Though, as discussed in Section 5.1, the notion of mean and of minimizing the square error is not necessarily the correct way to characterize selecting the optimal point estimate for colors. It is however a consistent way to do so, and so we use it for our evaluations.

5.4.4 Evaluation Strategies and Data

Full Task

We make use of the Munroe dataset as prepared by **mcmahan2015bayesian** from the results of the XKCD color survey. The XKCD color survey (**Munroe2010XKCDdataset**) collected over 3.4 million observations from over 222,500 respondents. McMahan and Stone take a subset from Munroe's full survey, by restricting it to the responses from native English speakers, and removing very rare color names with less than 100

uses. This gives a total of 2,176,417 observations and 829 color names. They also define a standard test, development and train split.

Full Task Corpus Statistics

- In the full corpus 829 unique color names made up of 308 unique terms.
- Training split
  - There are a total of 1,523,108 training observations.
  - The distribution of observations between color names has the following quartile statistics: Q0: 70.0, Q1: 109.0, Q2: 214.0, Q3: 627.0, Q4: 152,953.0.
  - The distribution of observations between terms has the following quartile statistics: Q0: 70.0, Q1: 148.5, Q2: 345.0, Q3: 2,241.75, Q4: 347,173.0.
- Development split
  - There are a total of 108,545 development observations.
  - The distribution of observations between color names has the following quartile statistics: Q0: 5.0, Q1: 7.0, Q2: 15.0, Q3: 45.0, Q4: 10,925.0.
  - The distribution of observations between terms has the following quartile statistics: Q0: 5.0, Q1: 10.0, Q2: 24.5, Q3: 159.25, Q4: 24,754.0.
- Test split
  - There are a total of 544,764 testing observations.
  - The distribution of observations between color names has the following quartile statistics: Q0: 25.0, Q1: 40.0, Q2: 78.0, Q3: 225.0, Q4: 54,627.0.
  - The distribution of observations between terms has the following quartile statistics: Q0: 26.0, Q1: 54.75, Q2: 124.5, Q3: 804.0, Q4: 124,138.0.

Unseen combination Task

A primary interest in using the term based models is to be able to make predictions for never before seen descriptions of colors. For example, based on the learned understanding of `salmon` and of `bright`, from examples like `bright green` and `bright red`, we wish for the system to make predictions about `bright salmon`, even though that description never occurs in the training data. The ability to make predictions, such as these, illustrates term-based natural language understanding. This cannot be done with the non-compositional baseline models, which bypass the term processing step. To evaluate this generalization capacity, we define new sub-datasets for both testing and training. We select the rarest 100 color descriptions from the full dataset, with the restriction that every token in a selected description must still have at least 8 uses

in other descriptions in the training set. The selected examples include multi-token descriptions such as: `bright yellow green` and also single tokens that occur more commonly as modifiers than as stand-alone descriptions such as `pale`.

The unseen combination testing set has only observations from the full test set that do use those rare descriptions. We define a corresponding restricted training set made up of the data from the full training set, excluding those corresponding to the rare descriptions. A restricted development set is created similarly to the training set, containing data from the full (original) validation set, with the exclusion of rare descriptions used in the test set. This was done so that no direct knowledge of the combined terms can leak during early-stopping.

By training on the restricted training set and testing on the unseen combinations, we can assess the model's capacity of compositionality to make predictions for color descriptions not seen during training. A similar approach was used in **acl2018WinnLighter** and in **DBLP:journals/corr/AtzmonBKGC16**. We contrast this to the same models when trained on the full training set to see how much accuracy was lost.

Unseen Combinations Corpus Statistics

- In the unseen combinations testset, there are (by design) 100 unique color names, that is 12.06% of the full set of color names. Thus the number of unique color names in the restricted training set is decreased by 100 names (i.e 12.06% smaller).
- 20,460 observations were removed from the training set . Thus the restricted training set contains 13.43% fewer observations than the full training set.
- 14 terms are used across the 100 color names in the unseen combinations test set. They are `blue`, `bright`, `brown`, `dark`, `deep`, `dull`, `green`, `grey`, `ish`, `light`, `lime`, `olive`, `orange`, `pale`, `pink`, `purple`, `red`, `rose`, `teal`, `very`, `violet`, `y`, `yellow`, and `-`.

Order Task

It is believed that the order of words in a color description matters, at least to some extent, for it's meaning. For example, `greenish brown` and `brownish green` are distinct, if similar, colors. To assess the models on their ability to make predictions when order matters we construct the order test set. This is a subset of the full test set containing only descriptions with terms that occur in multiple different orders. There are 76 such descriptions in the full dataset. Each of which has exactly one alternate ordering. This is unsurprising as while color descriptions may have more than 2 terms, normally one or more of the terms is a joining token such as `ish` or `-`. We only construct an order testing set, and not a corresponding training set, as this is an evaluation using the model trained on the full training data.

Order Task Corpus Statistics

- 76 unique color names with 2 possible orders for their terms are used. They makes up 9.17% of the unique color names in the full data set.
- In the full training set (which is used for training for this evaluation) there are 63,048 observations of these color names, making up 4.14% of all training observations.
- 16 terms are used in these ambiguous ordered color names. Namely: `apple`, `blue`, `bright`, `brown`, `green`, `grey`, `ish`, `light`, `orange`, `pink`, `purple`, `red`, `violet`, `y`, `yellow` and `-`.

5.5 Experimental Setup

5.5.1 Implementation

The implementation of all the models was in the Julia programming language (Bezanson et al. 2014). The full implementation can be downloaded from the GitHub repository.[4] The machine learning components makes heavy use of the MLDataUtils.jl[5] and TensorFlow.jl (Malmaud and White 2018) packages, the latter of which was enhanced significantly to allow for this work to be carried out. The discretization and the KDE for the non-compositional baseline is done using KernalDensityEstimation.jl.[6] The training data is managed with DataDeps.jl (White et al. 2018).

5.5.2 Common Network Features

Drop-out (**srivastava2014dropout**) is used on all ReLU layers and on the recurrent units in the RNNs, with threshold of 0.5 during training. The network is optimized using Adam (**kingma2014adam**), and a learning rate of 0.001. Early stopping is checked every 10 epochs using the development dataset. Distribution estimation methods are trained using the full batch (where each observation is a distribution) for every epoch. Point Estimation methods are trained using randomized mini-batches of 2^{16} observations (which are each color-space triples). All hidden-layers, except as otherwise precluded (inside the convolution, and in the penultimate layer of the point estimation networks) have the same width 300, as does the embedding layer.

5.6 Results

5.6.1 Qualitative Results

To get an understanding of the problem and how the models are performing, we consider some of the outputs of the model for particular cases. Figure 5.7 shows examples of distribution estimates, and Figure 5.8 shows

[4] Implementation source is at `https://github.com/oxinabox/ColoringNames.jl`
[5] MLDataUtils.jl is available from `https://github.com/JuliaML/MLDataUtils.jl`
[6] KernalDensityEstimation.jl is available from `https://github.com/JuliaStats/KernelDensity.jl`

Figure 5.7: Some examples of the output distribution estimates from the models trained on the full dataset

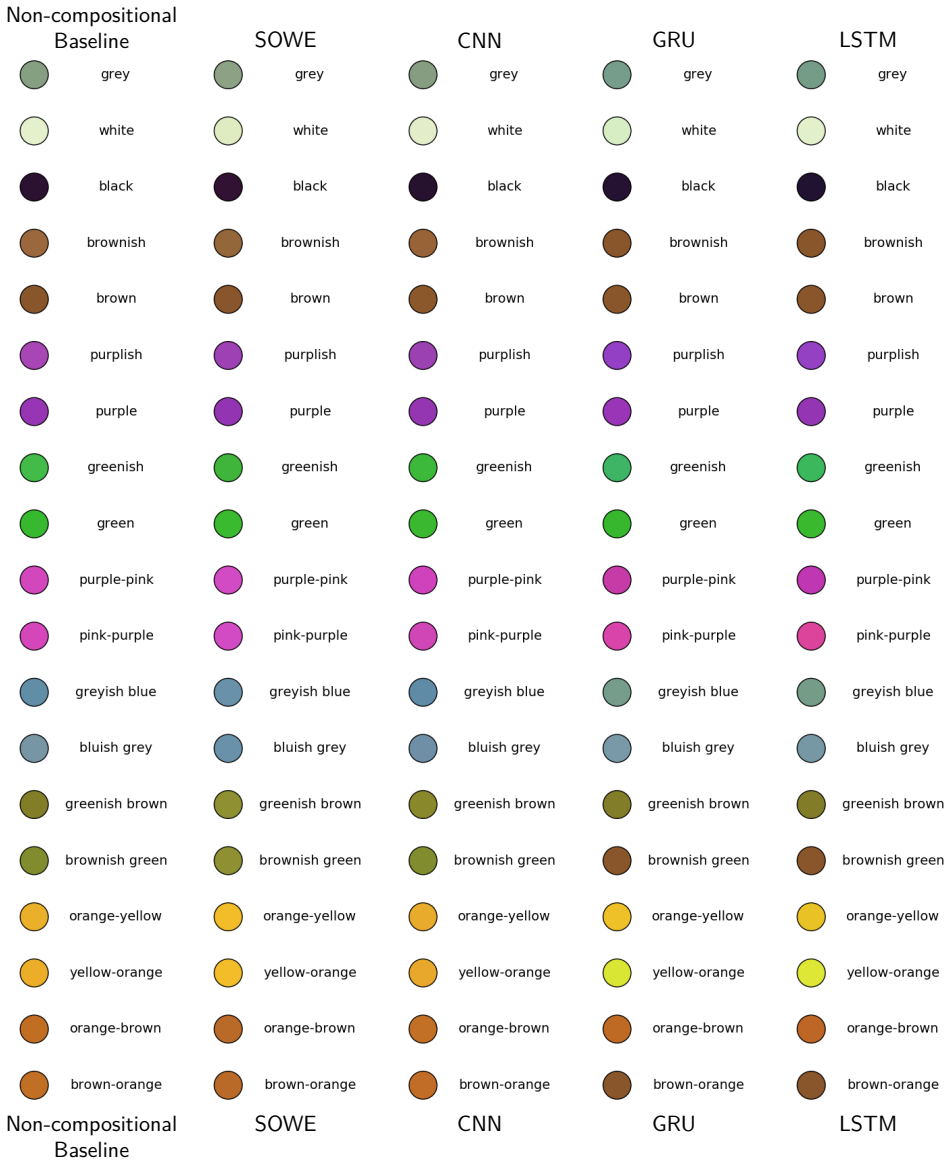


Figure 5.8: Some examples of the output point estimates from the models trained on the full dataset

similar examples for point estimates. Both are taken from models trained on the full training dataset. It can be seen that the models' outputs using term based estimation are generally similar to the non-term-based non-compositional baseline, as is intended. This shows that the models are correctly fitting to estimate the colors. It can be noted that in general the colors are very good, with only a few marked exceptions, discussed in the following sections, particularly around multi-word colors. To the naked eye, it is hard to to distinguish between the outputs of the different models. The general high quality of the estimates aligns with the strong results found in the quantitative evaluations discussed in Section 5.6.2. The example shown in Figures 5.7 and 5.8 serve to indicate that while the quantitative results do show that some of the models perform better than others, the true visual difference is very small.

On the effects of word-order

The different input modules have a different capacity to leverage word-order. This is reflected in Figures 5.7 and 5.8, when considering the pairs of outputs that differ only in word order, such as `purple-pink` and `pink-purple`. The plots presented for the training data and for the non-compositional baseline show that such color name pairs are subtly different but similar. The SOWE model is unable to take into account word order at all, and so produces identical outputs for all orders. The CNN models produce very similar outputs but not strictly identical – spotting the difference requires a very close observation. This is in-line with the different filter sizes allowing the CNN to effectively use n-gram features, and finding that the unigram features are the most useful. Both RNN models (GRU and LSTM) produce estimated distributions that visibly depend on the order of words. It seems that the first term dominates the final output: for example `greenish brown` is more green, and `brownish green` is more brown, contrary to the linguistic understanding. The RNN outputs are more similar to the color described by first term than any later terms. We can see that the first term is not solely responsible for the final output however, as `purple-pink`, `purple` and `purplish` (tokenized as `purple`, `ish`) are all different. It is surprising that the RNNs outputs are dominated by the first term and not the latter terms[7]. This shows that they are functioning to remember the earlier inputs. However, they are struggling to attribute the significance of the word order. Linguistically we would expect the last term to be the most significant: `greenish brown` is a shade of brown, not green. This expectation is reflected in the histogram for the training data. Although, for many of the order swapped colors the training histograms shown are very similar regardless of the order.

On the smoothness of the distribution estimates

In Figure 5.7 it can be seen that the term-based distribution estimation models are much smoother than the corresponding histograms taken from the training data. They are not as smooth as the non-compositional

[7]So much so that we double checked our implementation to be sure that it wasn't processing the inputs backwards.

Table 5.1: The results for the **full distribution estimation task**. Lower perplexity (PP) is better.

Method	$\frac{PP}{256^3}$
Non-compositional Baseline	0.071
SOWE	**0.075**
CNN	0.078
GRU	0.089
LSTM	0.092

baseline which explicitly uses KDE. However, they are much smoother than would be expected, had the output bins been treated independently. Thus it is clear that the models are learning that adjacent bins should have similar output values. This is a common feature of all the training data, no matter which color is being described. This learned effect is in line with the fact that color is continuous, and is only being represented here as discrete. We note in relation to this learned smoothness: that while the models capture the highly asymmetrical shapes of most distributions well, they do not do well at capturing small dips. Larger multi-modes as seen in the achromatic colors such as `white`, `grey`, `black`, `white`, are captured; but smaller dips such as the hue of `greenish` being more likely to be on either side of the green spectrum are largely filled in. In general, it seems clear that additional smoothing of the training data is not required for the neural network based models. This aligns with the results presented in Section 5.8.2.

5.6.2 Quantitative Results

Overall, we see that our models are able to learn to estimate colors based on sequences of terms. From the consideration of all the results shown in Tables 5.1 to 5.6, the CNN and SOWE models perform almost as well as the non-compositional baseline. With the SOWE having a marginal lead for distribution estimation, and the CNN and SOWE being nearly exactly equal for most point estimation tasks. We believe the reason for this is that the SOWE is an easier to learn model from a gradient descent perspective: it is a shallow model with only one true hidden layer. In general the results for the LSTM and GRU were very similar, and both much worse than the non-recurrent models. While it is only marginally behind the SOWE and CNN on the full point estimation task (Table 5.2), on all other tasks for both point estimation and distribution estimation it is significantly worse. This may indicate that it is hard to capture the significant relationships between terms in the sequence. However, as discussed Section 5.6.2 it did learn generally acceptable colors to the human eye, but the quantitative results presented in this section show that it is not as close a match to the population's expectation.

Ordered Task

The performance of SOWE on the order tasks (Tables 5.3 and 5.4) is surprising. For the distribution estimation it outperforms the CNN, and for point estimation it ties with the CNN. The CNN and RNN, can take

Table 5.2: The results for the **full point estimation task**. Lower mean squared error (MSE) is better.

Method	MSE
Non-compositional Baseline	0.066
SOWE	**0.067**
CNN	**0.067**
GRU	0.071
LSTM	0.071
Distribution Mean Non-compositional Baseline	0.066
Distribution Mean SOWE	0.068
Distribution Mean CNN	0.069
Distribution Mean GRU	0.077
Distribution Mean LSTM	0.077

Table 5.3: The results for the **order distribution estimation task**. Lower perplexity (PP) is better. This is a subset of the full test set containing only tests where the order of the words matters.

Method	$\frac{PP}{256^3}$
Non-compositional Baseline	0.053
SOWE	**0.055**
CNN	0.057
GRU	0.124
LSTM	0.125

into account word order, but the SOWE model cannot. The good results for SOWE suggest that the word-order is not very significant for color names. While word order matters, different colors with the same terms in different order are similar enough for most colors that it still performs very well. In theory the models that are capable of using word order have the capacity to ignore it, and thus could achieve a similar result. An RNN can learn to perform a sum of its inputs (the word embeddings), and the CNN can learn to weight all non-unigram filters to zero. In practice we see that for the RNN in particular this clearly did not occur. This can be attributed to the more complex networks being more challenging to train via gradient descent. It seems that color-naming is not a task where word order substantially matters, and thus the simpler SOWE model excels.

Table 5.4: The results for the **order point estimation task**. Lower mean squared error (MSE) is better. This is a subset of the full test set containing only tests where the order of the words matters.

Method	MSE
Non-compositional Baseline	0.065
SOWE	**0.066**
CNN	**0.066**
GRU	0.096
LSTM	0.096
Distribution Mean Non-compositional Baseline	**0.065**
Distribution Mean SOWE	**0.066**
Distribution Mean CNN	0.066
Distribution Mean GRU	0.095
Distribution Mean LSTM	0.088

Unseen Combinations of Terms

The SOWE and CNN models are able to generalize well to making estimates for combinations of color terms that are not seen in training. Tables 5.5 and 5.6 show the results of the model on the test set made up of rare combinations of color names (as described in Section 5.4.4) for the restricted training set (which does not contain those terms). These results on this test set are compared with the same models when trained on the full training set. The non-compositional baseline are unable to produce estimates from the unseen combinations testing set as they do not process the color names term-wise. Performing well on this task is indicative as to if the models are learning how the terms combine to determine the color, as they cannot be simply matching the full color name (term sequence) against one that occurs in training. This is an important test, as due to the combinatorial nature of language, it is common to encounter term sequences in the real world that never occur during training.

On distribution estimation (Table 5.5) the SOWE results are only marginally worse for the restricted training set as they are for the full training set. The CNN results are worse again, but they are still better than the results on the full test-set. The distribution estimates are good on absolute terms, having low evaluated perplexity.

In the point estimation task (Table 5.6) the order is flipped with the CNN outperforming the SOWE model. In-fact the CNN actually performs better with the restricted training set for predicting the unseen test colors, than it does for predicting those colors when they are included in the full training set; though the difference is only marginal. Unlike for distribution estimates, the unseen color point estimates are worse than the overall results from the full task (Table 5.2), though the errors are still small on an absolute scale.

Over all the performance of the SOWE and CNN remain strong on the unseen combination tasks. The RNN models continue to perform poorly on the unseen combination of terms task for both point and distribution estimation. The SOWE and CNN perform sufficiently well on the unseen combinations that the color estimates they produce would be practically useful. The unseen combination results are comparable to the full dataset results discussed (shown in Tables 5.1 and 5.2), and have very small errors on an absolute scale.

Extracting the mean from the distribution estimates

In the point estimation results discussed so far have been from models trained specifically for point estimation (as described by Section 5.3.5). However, it is also possible to derive the mean from the distribution estimation models. Those results are also presented in Tables 5.2, 5.4 and 5.6. In general these results perform marginally worse (using the MSE metric) than their corresponding modules using the point estimation output module. The only exception to this is the LSTM for both the unseen combination tasks and the order task, for which it was notably better to use the mean from the distribution rather than one directly

Table 5.5: The results for the **unseen combinations distribution estimation task**. Lower perplexity (PP) is better. This uses the unseen test set: a subset of the full test set contain only rare word combinations. In the restricted training set results these rare word combinations were removed from the training and development sets. In the full training set results the whole training and development stet was used, including the rare words that occur in the test set.

Method	Full Training Set $\frac{PP}{256^3}$	Restricted Training Set $\frac{PP}{256^3}$
Non-compositional Baseline	0.050	–
SOWE	**0.050**	**0.055**
CNN	0.052	0.065
GRU	0.117	0.182
LSTM	0.123	0.172

Table 5.6: The results for the **unseen combinations point estimation task**. Lower mean squared error (MSE) is better. This uses the unseen test set: a subset of the full test set contain only rare word combinations. In the restricted training set results these rare word combinations were removed from the training and development sets. In the full training set results the whole training and development stet was used, including the rare words that occur in the test set.

Method	Full Training Set MSE	Restricted Training Set MSE
Non-compositional Baseline	0.062	–
SOWE	**0.065**	0.079
CNN	0.072	**0.070**
GRU	0.138	0.142
LSTM	0.138	0.141
Distribution Mean Non-compositional Baseline	0.062	–
Distribution Mean SOWE	0.073	0.076
Distribution Mean CNN	0.073	0.084
Distribution Mean GRU	0.105	0.152
Distribution Mean LSTM	0.105	0.112

trained. We note that for the non-compositional baseline, the distributions mean is almost identical to the true mean of points, as expected.

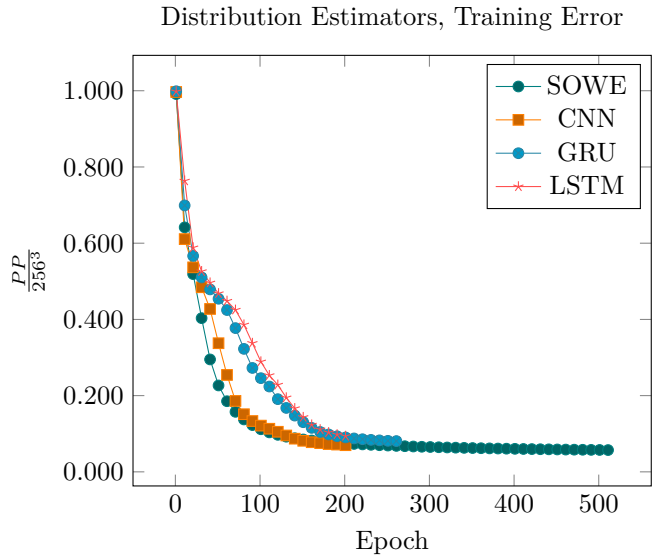
On the differences between the distribution estimation and point estimation training procedure

Beyond the output module there are a few key differences between the point estimation modules and the distribution estimate modules. When training distribution estimation models, all examples of a particular color name is grouped into a single high information training observation using the histogram as the output. Whereas when training for point estimation, each example is processed individually (using minibatches). This means that the distribution estimating models fit to all color names with equal priority. Whereas for point estimates, more frequently used color names have more examples, and so more frequent color names are fit with priority over rarer ones. Another consequence of using training per example using random minibatches, rather than aggregating and training with full batch, is increased resilience to to local minima (**lecun2012efficient**). One of the upsides of the aggregated training used in distribution estimation is that it trains much faster as only a small number of high-information training examples are processed, rather than a much larger number of individual observations.

It may be interesting in future work to consider training the distribution estimates per example using one-hot output representations; thus making the process similar to that used in the point estimate training. It is possible that such a method may have trouble learning the smoothness of the output space (as discussed in Section 5.6.1), as it would not see demonstration of the partial activation of adjacent bins in the training examples. However, this is not certain, much like the point estimation trained on one-hot learns a representation that minimises mean squared error outputting a point between all the training examples, it is reasonable to expect that the distribution estimates will output a smooth histogram as this is near to a minimum for the cross-entropy. With the current model the presence of partial activation of adjacent bins in all examples may be causing the smoothness to be learned primarily in the output layer, and with little respect for the inputs. Such would explain the difficulties in capturing subtler features of the output distribution, such as the depth of the valley between the two peaks in the hue of `greenish` shown in Figure 5.7. Using one-hot examples for training, may help force encoding the knowledge of the nature of continuous distributions deeper into the network allowing the input color name to have a more pronounce effect.

5.6.3 Training set results

To investigate our supposition that the SOWE, is a much easier function to fit via gradient descent, as compared to the CNN or the RNNs, we consider the error rate on the full training set during the training of the models. These plots are shown in Figure 5.9 and Figure 5.10. These plots seem to support the supposition, as the SOWE training error decreases notably faster (it is a steeper curve) in both cases. This corresponds to



Final Training Performance

Method	$\frac{PP}{256^3}$
SOWE	0.057
CNN	0.070
GRU	0.081
LSTM	0.094

Figure 5.9: The training set error of the distribution estimation models, when trained on the full dataset. Note that the plots stop when the model ceased training due to the development set error rising (early stopping).

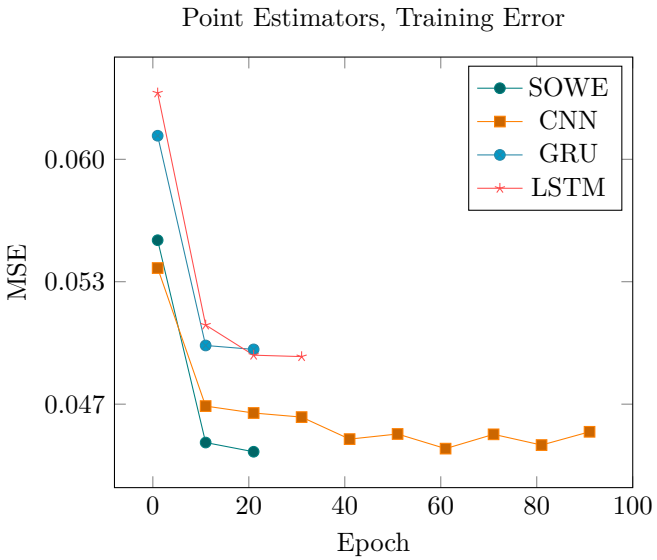
.

a easier error surface in network parameter (weights and biases) space, with fewer points of low gradient, or near local minima. If we compare the final loss of each method on the training set (before it was stopped due to early stopping) against the test set results in Tables 5.1 and 5.2 we find they are similar, particularly for distribution estimation (Table 5.1 and Figure 5.9). While for point estimation (Figure 5.10 and table 5.2), on the test set CNN and SOWE perform similarly, while RNNs perform much worse, despite the fact that in training the performance of CNN is roughly midway between SOWE and the RNNs. In all cases, the absolute error in training has become small relative the to the natural variation in the training set by the time early stopping terminates training. Note that perfect fit is not possible as the training data varies.

5.6.4 Completely Unseen Color Estimation From Embeddings

As an interesting demonstration of how the models function by learning the transformation from the embedding space to the output, we briefly consider the outputs for color-names that do not occur in the training or testing data at all. This is even more extreme than the unseen combination task considered in Tables 5.5 and 5.6 where the terms appeared in training, but not the combination of terms. In the examples shown in Figures 5.11 and 5.12, where the terms never occurred in the training data at all, our models exploit the fact that they work by transforming the word-embedding space to predict the colors. There is no equivalent for this in the direct models. While `Grey` and `gray` never occur in the training data; `grey` does, and it is near-by in the word-embedding space. Similar is true for the other colors that vary by capitalization. We only present a few examples of single term colors here, and no quantitative investigation, as this is merely a matter of interest.

It is particularly interesting to note that the all the models make similar estimations for each color. This occurs both for point estimation and



Final Training Performance

Method	MSE
SOWE	0.132
CNN	0.135
GRU	0.149
LSTM	0.148

Figure 5.10: The training set error of the point estimation models, when trained on the full dataset. Note that the plots stop when the model ceased training due to the development set error rising (early stopping).

for distribution estimation. They do well on the same colors and make similar mistakes on the colors they do poorly at. The saturation of `Gray` is estimated too high, making it appear too blue/purple, this is also true of `grey` though to a much lesser extent. `Purple` and `Green` produce generally reasonable estimates. The hue for `Brown` is estimated as having too much variance, allowing the color to swing into the red or yellowish-green parts of the spectrum. This suggests that in general all models are learning a more generally similar transformation of the space. In general the overall quality of each model seems to be in line with that found in the results for the full tests.

5.7 Conclusion

We have presented four input modules (SOWE, CNN, GRU and LSTM), and two output modules (distribution estimate, and point estimate) that are suitable for using machine learning to make estimates about color based on the terms making up its name. We contrasted these to a non-compositional baseline model for each task which bypassed the term-wise natural language understanding component of the problem. We found the results for SOWE, and CNN were strong, approaching this strong baseline.

It is a note-worthy feature on the current state of short phrase modeling, and the difficulty of compositional natural language understanding that the term-based models are not able to out-perform the non-compositional baseline where training data for the whole phrases was available. The term-based models are effectively given additional information, in the form of the tokenisation, but are unable to fully leverage it in the general case. They are unable to outperform simply ignoring the common sub-phase information when training instances for the whole phrase are available.

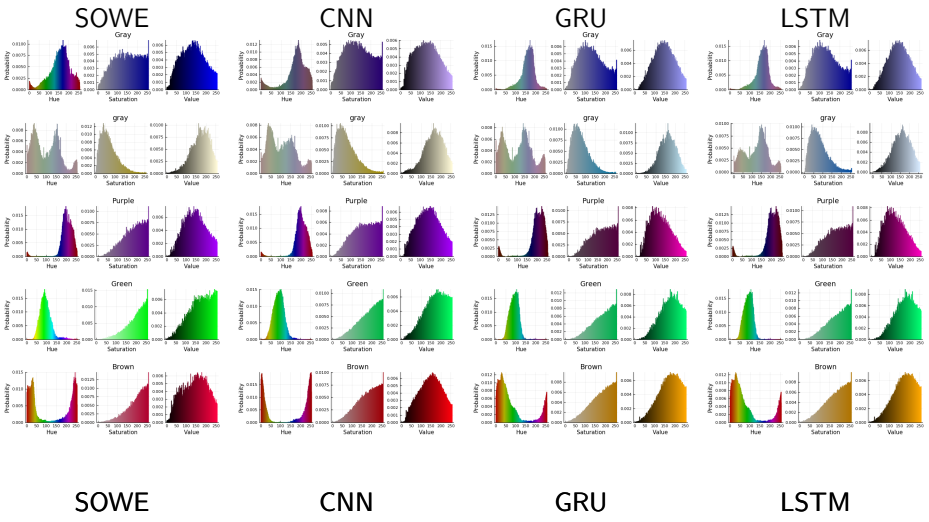


Figure 5.11: Some example distribution estimations for colors names which are completely outside the training data. The terms: `Brown`, `gray`, `Gray`, `Green`, and `Purple`, do not occur in any of the color data; however `brown`, `grey green`, and `purple` do occur.

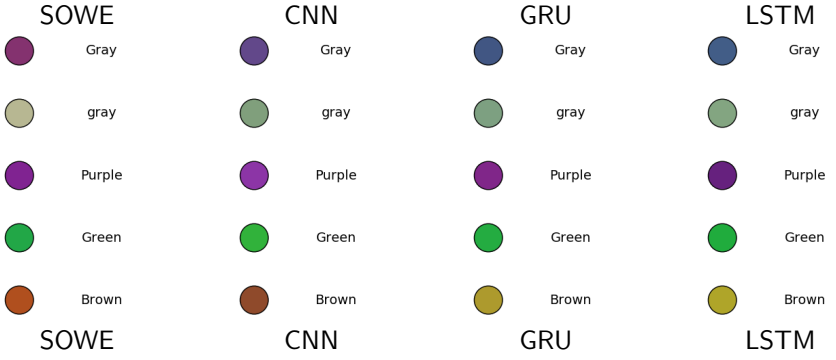


Figure 5.12: Some example point estimates for colors names which are completely outside the training data. The terms: `Brown`, `gray`, `Gray`, `Green`, and `Purple`, do not occur in any of the color data; however `brown`, `grey green`, and `purple` do occur.

A key take away from our results is that using a SOWE should be preferred over an RNN for short phrase natural language understanding tasks when order is not a very significant factor. It is also important to evaluate if order is indeed a significant factor, since on the surface one would expect it to be for color names. One way to evaluate this is to include SOWE as a baseline model in other tasks. While RNNs are the standard type of model for problems with sequential input, such as color names made up of multiple words as we considered here. However, we find both LSTM and GRU performance to be significantly exceeded by SOWE and CNN. SOWE is an unordered model roughly corresponding to a bag of words. CNN similarly roughly correspondents to a bag of ngrams, in our case a bag of all 1,2,3,4 and 5-grams. This means that the CNN can readily take advantage of both fully ordered information, using the filters of length 5, down to unordered information using filters of length 1. RNNs however must fully process the ordered nature of its inputs, as its output comes only from the final node. Between the two RNN models it seems the GRU performs marginally better. It would be interesting to further compare with bidirectional variants of these RNNs.

In a broader context, we envisage the distribution learned for a color name can be used as a prior probability, and when combining with additional context information, a likelihood can be estimated for particular uses. This additional information could take the from of other words, such as estimating the distribution for a `brown dog`, as compared to a `brown tree`, or from other sources. A particularly interesting related avenue for investigation would condition the model not only on the words used but also on the speaker. The original source of the data, **Munroe2010XKCDdataset**, includes some demographic information which is not explored as a model input in any published model (to the best of our knowledge). It is expected that color-term usage may vary with subcultures.

5.8 Appendix

5.8.1 On the Conditional Independence of Color Channels given a Color Name

As discussed in the main text, we conducted a superficial investigation into the truth of our assumption that given a color name, the distributions of the hue, value and saturation are statistically independent.

We note that this investigation is, by no means, conclusive though it is suggestive. The investigation focusses around the use of the Spearman's rank correlation. This correlation measures the monotonicity of the relationship between the random variables. A key limitation is that the relationship may exist but be non-monotonic. This is almost certainly true for any relationship involving channels, such as hue, which wrap around. In the case of such relationships Spearman's correlation will underestimate the true strength of the relationship. Thus, this test is of limited use in proving conditional independence. However, it is a quick test to perform and does suggest that the conditional independence assumption may not be so incorrect as one might assume.

In Monroe Color Dataset the training data given by $V \subset \mathbb{R}^3 \times T$, where $\mathbb{R}^3$ is the value in the color-space under consideration, and T is the natural language space. The subset of the training data for the description $t \in T$ is given by $V_{|t} = \{(\tilde{v}_i, t_i) \in V \mid t_i = t\}$. Further let $T_V = \{t_i \mid (\tilde{v}, t_i) \in V$ be the set of color names used in the training set. Let $V_{\alpha|t}$ be the α channel component of $V_{|t}$, i.e. $V_{\alpha|t} = \left\{v_\alpha \mid ((v_1, v_2, v_3), t) \in V_{|t}\right\}$.

The set of absolute Spearman's rank correlations between channels a and b for each color name is given by $S_{ab} = \left\{\left|\rho(V_{a|t}, V_{b|t})\right| \; t \in T_V\right\}$.

Table 5.7: The third quartile for the pairwise Spearman's correlation of the color channels given the color name.

Color-Space	$Q3(S_{12})$	$Q3(S_{13})$	$Q3(S_{23})$	max
HSV	0.1861	0.1867	0.1628	0.1867
HSL	0.1655	0.2147	0.3113	0.3113
YCbCr	0.4005	0.4393	0.3377	0.4393
YIQ	0.4088	0.4975	0.4064	0.4975
LCHab	0.5258	0.411	0.3688	0.5258
DIN99d	0.5442	0.4426	0.4803	0.5442
DIN99	0.5449	0.4931	0.5235	0.5449
DIN99o	0.5608	0.4082	0.5211	0.5608
RGB	0.603	0.4472	0.5656	0.603
Luv	0.5598	0.6112	0.4379	0.6112
LCHuv	0.6124	0.4072	0.3416	0.6124
HSI	0.2446	0.2391	0.6302	0.6302
CIELab	0.573	0.4597	0.639	0.639
xyY	0.723	0.5024	0.4165	0.723
LMS	0.968	0.7458	0.779	0.968
XYZ	0.9726	0.8167	0.7844	0.9726

We consider the third quartile of that correlation as the indicative statistic in Table 5.7. That is to say for 75% of all color names, for the given color-space, the correlation is less than this value.

Of the 16 color-spaces considered, it can be seen that the HSV exhibits the strongest signs of conditional independence – under this (mildly flawed) metric. More properly put, it exhibits the weakest signs of non-independence. This includes being significantly less correlated than other spaces featuring circular channels such as HSL and HSI.

Our overall work makes the conditional independence assumption, much like n-gram language models make the Markov assumption. The success of the main work indicates that the assumption does not cause substantial issues.

5.8.2 KDE based smoothing of Training Data

It can be seen that smoothing has very little effect on the performance of any of the neural network based distribution estimation models. All four term based models (SOWE, CNN, LSTM, GRU all perform very similarly whether or not the training data is smoothed. This is seen consistently in all the distribution estimation tasks. Contrast Tables 5.8 to 5.10 to the tables for the unsmoothed results Tables 5.1, 5.3 and 5.5.

If however, smoothing is not applied to the operational upper bound, it works far worse. In Tables 5.8 to 5.10 the Direct result refers to using the training histograms almost directly, without any smoothing or term-based input processing. This is the same as the operational upper bound, minus the KDE. It works very poorly (by comparison). This is because the bins values are largely independent: a very high probability in one bin does not affect the probability of the adjacent bin – which by chance of sampling may be lower than would be given by the true distribution.

This is particularly notable in the case of the direct, full training set result on the unseen combinations task reported in Table 5.10. As these were

Table 5.8: The results for the **full distribution estimation task** using smoothed training data. Lower perplexity (PP) is better. This corresponds to the main results in Table 5.1.

Method	$\frac{PP}{256^3}$
Direct	0.164
Operational Upper Bound	0.071
SOWE-smoothed	0.075
CNN-smoothed	0.079
GRU-smoothed	0.088
LSTM-smoothed	0.090

Table 5.9: The results for the **order distribution estimation task** using smoothed training data. Lower perplexity (PP) is better. This is a subset of the full test set containing only tests where the order of the words matters. This corresponds to the main results in Table 5.3.

Method	$\frac{PP}{256^3}$
Direct	0.244
Operational Upper Bound	0.053
SOWE-smoothed	0.055
CNN-smoothed	0.058
GRU-smoothed	0.122
LSTM-smoothed	0.120

some of the rarest terms in the training set, several did not coincide with any bins for observations in testing set. This is because without smoothing it results in estimating the probability based on bins unfilled by any observation. We do cap that empty bin probability at $\epsilon_{64} \approxeq 2 \times 10^{-16}$ to prevent undefined perplexity. We found capping the lower probability for bins like this to be far more effective than add-on smoothing.

Conversely, on this dataset the neural network models do quite well, with or without smoothing. As the network can effectively learn the smoothness, not just from the observations of one color but from all of the observations. It learns that increasing the value of one bin should increase the adjacent ones. As such smoothing does not need to be applied to the training data.

Table 5.10: The results for the **unseen combinations distribution estimation task** using smoothed training data. Lower perplexity (PP) is better. This uses the extrapolation subset of the full test set. In the extrapolating results certain rare word combinations were removed from the training and development sets. In the non-extrapolating results the full training and development stet was used. This corresponds to the main results in Table 5.5.

Method	Full Training Set $\frac{PP}{256^3}$	Restricted Training Set $\frac{PP}{256^3}$
Direct	175.883	–
Operational Upper Bound	0.050	–
SOWE-smoothed	0.050	0.056
CNN-smoothed	0.053	0.063
GRU-smoothed	0.112	0.183
LSTM-smoothed	0.119	0.162

Chapter 6

Conclusion

Current research in natural language understanding relies on the creation of representations of natural language that can be readily manipulated by computer algorithms for purposes of making inferences about meaning. This thesis has focused on one particular type of representation: linear combinations of embeddings. This is a very simple representation, closely related to a bag of words. There is a machine learning adage: that given enough data and a model with sufficiently high representational capacity any problem can be solved. However, we seem to have found a sweet spot, where a model seemingly without sufficiently high representational capacity, never-the-less performs excellently on tasks with the amount of data that we have. It seems clear that there will always exist low-medium resource settings where linear combinations of embeddings will remain an ideal method for many practical problems.

The research presented here on linear combinations of embeddings has shown that this simple input representation technique is surprisingly powerful. This power is related to the fact that surface level information plays a significant role in practically giving human understandable meaning to a natural language utterance. Word content is the most obvious surface level information, and is effectively captured by a LCOWE. The LCOWE represented this in a dense, but informative vector. While the LCOWE loses word order information, it preserves the aggregated content very well, making it very useful for the tasks considered in this research.

We considered a number of tasks to identify the utility of this representation. **??** investigated classifying paraphrases as a means to investigate the quality of SOWE as a sentence embedding method. Chapter 5 defined models for color estimation from short phrases. **??** considered if we could use weighted combinations of sense embeddings to better capture the sense used in a particular example. **??** considered taking the mean of the embeddings adjacent to named entity tokens across a fictional text as a feature to characterize how the named entity token was being used. We followed up these practical demonstrations of capacity, with further investigations into what can be recovered from the SOWE in the important area of sentence representations. **??** demonstrated a method that could partially recover bags of words from a given SOWE. **??** extended this work by attempting to order those bags of words into sentences. This demonstrated that a surprising amount of information is still available in

the summed embeddings; which helps to explain why they work so well.

Linear combinations of embedding are not perfect for representing all meaning, as they do not encode any information about word order. It is thus clear that there exists sentences and phrases that are ambiguous when represented this way. However, we note that such sentences are rare: often there is only one likely ordering, particularly in any given text with a restricted domain. Most sentences are relatively short; multiple similarly likely word ordering occur more often in longer sentences. Many reorderings are paraphrases, or near paraphrases, particularly when done at the clause level. Though some orderings, such as noun swaps of nouns with similar ontological classification (e.g. Agents, Objects) do exist at almost all lengths: many are paraphrases `The banana is next to the orange` *vs.* `The orange is next to the banana`; and others are similar in meaning: `The banana is to the left of the orange` *vs.* `The orange is to the left of the banana`. It is desirable that such sentences are nearby in a representational of the semantic space.

6.1 Future work

6.1.1 Adversarial Test cases

A limitation of the LCOWE representations is that they have no ability to represent word order. This is in-contrast to RNNs and other commonly used neural network based representations of multi-word natural language input. It is possibly to construct adversarial test cases, that no LCOWE can succeed on. This can be done by selecting sentences with multiple reasonable word orders with very different meanings. It is worth consideration, that such adversarial test cases allow advancement of the state of the art to increase the capacity of models to represent all possible inputs. However, they do not necessarily advance the practical state of the art in representing real inputs that occur in a particular domain. Thus it is is essential to understand how common such adversarial test cases are in practice.

Future work in this area requires not just the construction of adversarial examples; but of the determination of how common they are in practice. Adversarial examples are not ubiquitous in real world tasks. It is important not to succeed on only these cases, while failing on the more common simple cases.

It is also important to consider how challenging an adversarial test case is. In Chapter 5, the ordered task which was to make predictions for colors for which the different words in the name could appear in different orders to describe different colors. For example `bluish green` and `greenish blue` are different colors. However, they are very *similar* colors. As such the error from discarding word order, is less than the error from using a more complicated model such as an RNN. Such a more complex model is harder to train, and those practical difficulties can dominate over a small amount of theoretical lack of capacity.

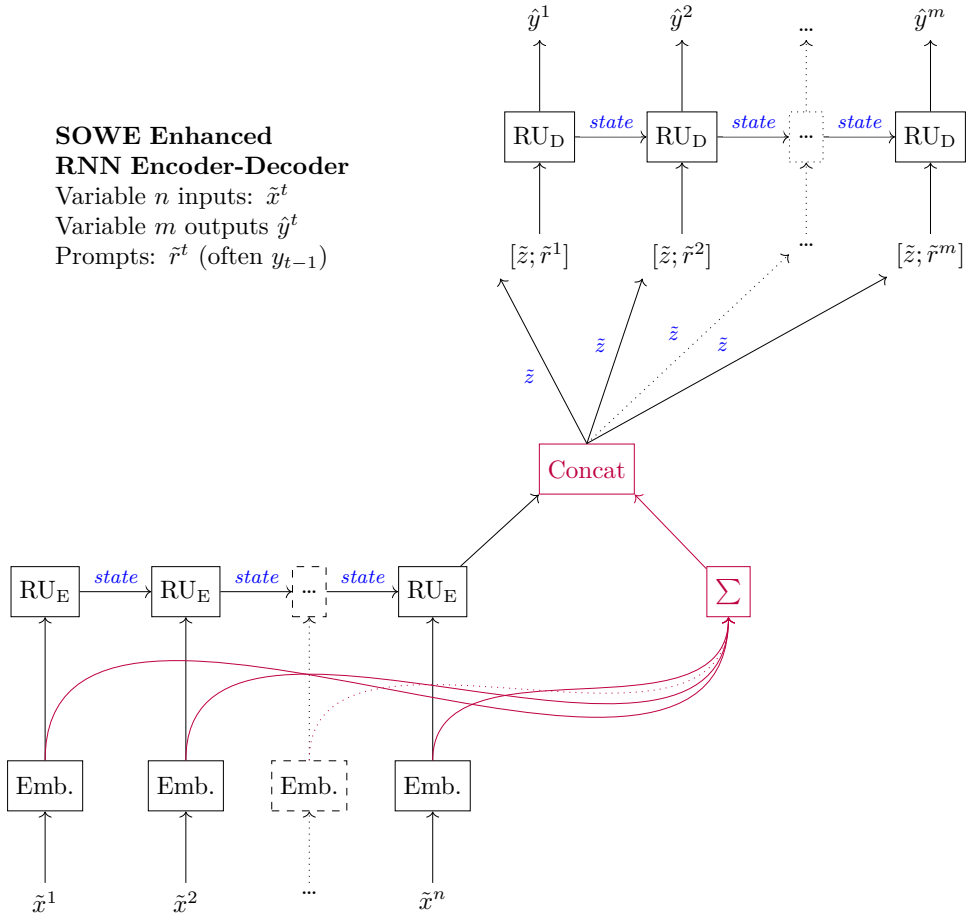


Figure 6.1: An encoder-decoder model with a SOWE encoder bypass layer added (shown in red).

6.1.2 Language Models and Orderless Representations

There is a complementary aspect to LCOWE and language models. While LCOWE have no capacity to handle word order, but they have an excellent ability to capture word content; whereas pure language models have no ability to capture word content, but have an excellent ability to capture word order. Language modelling based models incorporating a representation stage, such as encoder-decoders (Cho et al. 2014), do not capture word content as well as LCOWE (Conneau et al. 2018). They do, however, have state of the art order representation.

An interesting combination of the two, would be an encoder model, where the coding layer, is augmented by concatenating the final RNN output, with a sum of word embeddings, for all the input words to the encoder. An example of this for a encoder-decoder is shown in Figure 6.1. This would effectively allow a bypass of the encoder RNN. A similar bypass of intermediate layers has been used in feed-forward networks including the notable neural probabilistic language model (Bengio et al. 2003). The significant advantage of bypassing the RNN encoder is that it allows the model to weight the value of the orderful representation of the RNN output, against the unordered representation of the SOWE and learn use which ever is better for the task. Further, having explicit access to the surface level features in the SOWE, should help encourage the orderful encoder to learn more important deeper features.

A coding layer featuring components from a encoder capturing order-features, and a SOWE capturing surface features can be expected to perform better at both representations than either alone. This expectation is due to the the weighting above the shared layer will train to weight each feature for what it is better at, and thus during gradient decent the weights for the encoder would be decreased for surface information that is better obtained from the SOWE. It would thus allow each part of the network to *focus* on what it is best at, thus creating better representations./ This thesis has shown that SOWE can excel at surface level tasks (and that more tasks that expected are surface level). On deeper tasks where structure becomes more important ordered representations out perform it (Conneau et al. 2018). By combining the two we expect to get the best of both worlds, and produce truly excellent models for natural language understanding.

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2016). "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, pp. 265–283. ISBN: 978-1-931971-33-1.

Adi, Yossi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg (2017). "Fine-grained analysis of sentence embeddings using auxiliary prediction tasks". In: *Proceedings of ICLR Conference Track*.

Agirre, Eneko and Aitor Soroa (2007). "Semeval-2007 Task 02: Evaluating Word Sense Induction and Discrimination Systems". In: *Proceedings of the 4th International Workshop on Semantic Evaluations*. SemEval '07. Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 7–12.

Agirre, Eneko, David Martínez, Oier López De Lacalle, and Aitor Soroa (2006). "Evaluating and optimizing the parameters of an unsupervised graph-based WSD algorithm". In: *Proceedings of the first workshop on graph based methods for natural language processing*. Association for Computational Linguistics, pp. 89–96.

Arora, Sanjeev, Yingyu Liang, and Tengyu Ma (2017). "A simple but tough-to-beat baseline for sentence embeddings". In: *Proceedings of ICLR Conference Track*.

Bartunov, Sergey, Dmitry Kondrashkin, Anton Osokin, and Dmitry P. Vetrov (2015). "Breaking Sticks and Ambiguities with Adaptive Skip-gram". In: *CoRR* abs/1502.07257.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Janvin (2003). "A Neural Probabilistic Language Model". In: *The Journal of Machine Learning Research*, pp. 137–186.

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah (2014). "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1, pp. 65–98. DOI: 10.1137/141000671.

Bird, Steven and Edward Loper (2004). "NLTK: the natural language toolkit". In: *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, p. 31.

Blei, David M, Andrew Y Ng, and Michael I Jordan (2003). "Latent dirichlet allocation". In: *the Journal of machine Learning research* 3, pp. 993–1022.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov (2017). "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146.

Booth, Wayne C (1961). *The rhetoric of fiction*. University of Chicago Press.

Borko, Harold and Myrna Bernick (1963). "Automatic document classification". In: *Journal of the ACM (JACM)* 10.2, pp. 151–162.

Bowman, Samuel R, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts (2016a). "A fast unified model for parsing and sentence understanding". In: *arXiv preprint arXiv:1603.06021*.

Bowman, Samuel R, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio (2016b). "Generating Sentences from a Continuous Space". In: *International Conference on Learning Representations (ICLR) Workshop*.

Brown, Peter F, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai (1992). "Class-based n-gram models of natural language". In: *Computational linguistics* 18.4, pp. 467–479.

Chen, Tianqi, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang (2015). "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems". In: *arXiv preprint arXiv:1512.01274*.

Chen, Xinxiong, Zhiyuan Liu, and Maosong Sun (2014). "A Unified Model for Word Sense Representation and Disambiguation." In: *EMNLP*. Citeseer, pp. 1025–1035.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734.

Collobert, Ronan and Jason Weston (2008). "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 160–167.

Conneau, Alexis, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni (2018). "What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 2126–2136.

Cotterell, Ryan, Adam Poliak, Benjamin Van Durme, and Jason Eisner (2017). "Explaining and Generalizing Skip-Gram through Exponential Family Principal Component Analysis". In: *EACL 2017* 175.

Cífka, Ondřej and Ondřej Bojar (2018). "Are BLEU and Meaning Representation in Opposition?" In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1362–1371.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*.

Dhillon, Paramveer, Dean P Foster, and Lyle H Ungar (2011). "Multi-view learning of word embeddings via cca". In: *Advances in Neural Information Processing Systems*, pp. 199–207.

Dinu, Georgiana and Marco Baroni (2014). "How to make words with vectors: Phrase generation in distributional semantics". In: *Proceedings of ACL*, pp. 624–633.

Dolan, William B. and Chris Brockett (2005). "Automatically Constructing a Corpus of Sentential Paraphrases". In: *Third International Workshop on Paraphrasing (IWP2005)*. Asia Federation of Natural Language Processing.

Drummond, Chris (2009). "Replicability is not reproducibility: nor is it good science". In: *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*.

Dumais, Susan T, George W Furnas, Thomas K Landauer, Scott Deerwester, and Richard Harshman (1988). "Using latent semantic analysis to improve access to textual information". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. Acm, pp. 281–285.

Elson, David K., Nicholas Dames, and Kathleen R. McKeown (2010). "Extracting Social Networks from Literary Fiction". In: *Proceedings of the 48th Annual Meet-*

ing of the Association for Computational Linguistics. ACL '10. Uppsala, Sweden: Association for Computational Linguistics, pp. 138–147.

Fan, Rong-En, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin (2008). "LIBLINEAR: A Library for Large Linear Classification". In: *Journal of Machine Learning Research* 9, pp. 1871–1874.

Farhadi, Ali, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth (2010). "Every picture tells a story: Generating sentences from images". In: *Computer Vision–ECCV 2010*. Springer, pp. 15–29.

Faruqui, Manaal and Chris Dyer (2014). "Improving vector space word representations using multilingual correlation". In: Association for Computational Linguistics.

Finkelstein, Lev, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin (2001). "Placing search in context: The concept revisited". In: *Proceedings of the 10th international conference on World Wide Web*. ACM, pp. 406–414.

Francis, W Nelson and Henry Kucera (1979). "Brown corpus manual". In: *Brown University*.

Fu, X., K. Huang, E. E. Papalexakis, H. A. Song, P. P. Talukdar, N. D. Sidiropoulos, C. Faloutsos, and T. Mitchell (2016). "Efficient and Distributed Algorithms for Large-Scale Generalized Canonical Correlations Analysis". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 871–876. DOI: 10.1109/ICDM.2016.0105.

Ganesan, Kavita, ChengXiang Zhai, and Jiawei Han (2010). "Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions". In: *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, pp. 340–348.

Gershman, Samuel J and Joshua B Tenenbaum (2015). "Phrase similarity in humans and machines". In: *Proceedings of the 37th Annual Conference of the Cognitive Science Society*.

Gladkova, Anna, Aleksandr Drozd, and Satoshi Matsuoka (2016). "Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't." In: *SRW@ HLT-NAACL*, pp. 8–15.

Goller, Christoph and Andreas Kuchler (1996). "Learning task-dependent distributed representations by backpropagation through structure". In: *Neural Networks, 1996., IEEE International Conference on*. Vol. 1. IEEE, pp. 347–352.

Goodman, Alyssa, Alberto Pepe, Alexander W. Blocker, Christine L. Borgman, Kyle Cranmer, Merce Crosas, Rosanne Di Stefano, Yolanda Gil, Paul Groth, Margaret Hedstrom, David W. Hogg, Vinay Kashyap, Ashish Mahabal, Aneta Siemiginowska, and Aleksandra Slavkovic (2014). "Ten Simple Rules for the Care and Feeding of Scientific Data". In: *PLOS Computational Biology* 10.4, pp. 1–5. DOI: 10.1371/journal.pcbi.1003542.

Grave, Edouard, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov (2018). "Learning Word Vectors for 157 Languages". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Grice, H Paul (1975). "Logic and conversation". In: *Speech Acts* 3, pp. 41–58.

Guevara, Emiliano (2010). "A regression model of adjective-noun compositionality in distributional semantics". In: *Proceedings of the 2010 Workshop on Geometrical Models of Natural Language Semantics*. Association for Computational Linguistics, pp. 33–37.

Gujral, Biman, Huda Khayrallah, and Philipp Koehn (2016). "Translation of Unknown Words in Low Resource Languages". In: *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA)*.

Gutmann, Michael U and Aapo Hyvärinen (2012). "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics". In: *Journal of Machine Learning Research* 13.Feb, pp. 307–361.

Ha, Le Quan, Philip Hanna, Ji Ming, and F Jack Smith (2009). "Extending Zipf's law to n-grams for large corpora". In: *Artificial Intelligence Review* 32.1, pp. 101–113.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hofmann, Thomas (2000). "Learning the similarity of documents: An information-geometric approach to document retrieval and categorization". In: *Advances in neural information processing systems*, pp. 914–920.

Horvat, Matic and William Byrne (2014). "A Graph-Based Approach to String Regeneration." In: *EACL*, pp. 85–95.

Huang, Eric H, Richard Socher, Christopher D Manning, and Andrew Y Ng (2012). "Improving word representations via global context and multiple word prototypes". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, pp. 873–882.

Huffman, David A (1952). "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9, pp. 1098–1101.

Iacobacci, Ignacio, Mohammad Taher Pilehvar, and Roberto Navigli (2015). "SensEmbed: learning sense embeddings for word and relational similarity". In: *Proceedings of ACL*, pp. 95–105.

Imani, M. B., S. Chandra, S. Ma, L. Khan, and B. Thuraisingham (2017). "Focus location extraction from political news reports with bias correction". In: *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1956–1964. DOI: 10.1109/BigData.2017.8258141.

Innes, Mike (2018). "Flux: Elegant Machine Learning with Julia". In: *Journal of Open Source Software*. DOI: 10.21105/joss.00602.

Innes, Mike, David Barber, Tim Besard, James Bradbury, Valentin Churavy, Simon Danisch, Alan Edelman, Stefan Karpinski, Jon Malmaud, Jarrett Revels, Viral Shah, Pontus Stenetorp, and Deniz Yuret (2017). "On Machine Learning and Programming Languages". In: *SysML Conference*.

Iyyer, Mohit, Jordan Boyd-Graber, and Hal Daumé III (2014). "Generating Sentences from Semantic Vector Space Representations". In: *NIPS Workshop on Learning Semantics*.

Iyyer, Mohit, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III (2014). "A neural network for factoid question answering over paragraphs". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 633–644.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever (2015). "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342–2350.

Jurgens, David A, Peter D Turney, Saif M Mohammad, and Keith J Holyoak (2012). "Semeval-2012 task 2: Measuring degrees of relational similarity". In: *Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics, pp. 356–364.

Kågebäck, Mikael, Olof Mogren, Nina Tahmasebi, and Devdatt Dubhashi (2014). "Extractive summarization using continuous vector space models". In: *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, pp. 31–39.

Kågebäck, Mikael, Fredrik Johansson, Richard Johansson, and Devdatt Dubhashi (2015). "Neural context embeddings for automatic discovery of word senses". In: *Proceedings of NAACL-HLT*, pp. 25–32.

Karp, Richard M (1972). *Reducibility among combinatorial problems*. Springer.

Katz, Slava M (1987). "Estimation of probabilities from sparse data for the language model component of a speech recognizer". In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 35.3, pp. 400–401.

Kilgarriff, Adam (2004). "How Dominant Is the Commonest Sense of a Word?" In: *Text, Speech and Dialogue: 7th International Conference, TSD 2004, Brno, Czech Republic, September 8-11, 2004. Proceedings*. Ed. by Petr Sojka, Ivan Kopecek, Karel Pala, Petr Sojka, Ivan Kopecek, and Karel Pala. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 103–111. ISBN: 978-3-540-30120-2. DOI: 10.1007/978-3-540-30120-2_14.

Kingma, D. P and M. Welling (2014). "Auto-Encoding Variational Bayes". In: *The International Conference on Learning Representations (ICLR)*. arXiv: 1312.6114 [stat.ML].

Kiros, Ryan, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler (2015). "Skip-Thought Vectors". In: *CoRR* abs/1506.06726.

Klein, Benjamin, Guy Lev, Gil Sadeh, and Lior Wolf (2015). "Associating neural word embeddings with deep image representations using fisher vectors". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4437–4446.

Kneser, Reinhard and Hermann Ney (1995). "Improved backing-off for m-gram language modeling". In: *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*. Vol. 1. IEEE, pp. 181–184.

Landgraf, Andrew J. and Jeremy Bellay (2017). "word2vec Skip-Gram with Negative Sampling is a Weighted Logistic PCA". In: *CoRR* abs/1705.09755.

Lau, Jey Han and Timothy Baldwin (2016). "An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation". In: *ACL 2016*, p. 78.

Le, Quoc and Tomas Mikolov (2014). "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1188–1196.

Levy, Omer and Yoav Goldberg (2014). "Neural word embedding as implicit matrix factorization". In: *Advances in neural information processing systems*, pp. 2177–2185.

Levy, Omer, Yoav Goldberg, and Ido Dagan (2015). "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 3, pp. 211–225. ISSN: 2307-387X.

Li, Bofang, Tao Liu, Zhe Zhao, Puwei Wang, and Xiaoyong Du (2017). "Neural Bag-of-Ngrams." In: *AAAI*, pp. 3067–3074.

Li, Yitan, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen (2015). "Word Embedding Revisited: A New Representation Learning and Explicit Matrix Factorization Perspective." In: *IJCAI*, pp. 3650–3656.

Lubin, Miles and Iain Dunning (2015). "Computing in operations research using Julia". In: *INFORMS Journal on Computing* 27.2, pp. 238–248.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov, pp. 2579–2605.

Malmaud, Jonathan and Lyndon White (2018). "TensorFlow.jl: An Idiomatic Julia Front End for TensorFlow". In: *Journal of Open Source Software*. DOI: 10.21105/joss.01002.

Maron, Melvin Earl (1961). "Automatic indexing: an experimental inquiry". In: *Journal of the ACM (JACM)* 8.3, pp. 404–417.

Meindl, Bernhard and Matthias Templ (2012). "Analysis of commercial and free and open source solvers for linear optimization problems". In: *Eurostat and Statistics Netherlands*.

Mesnil, Grégoire, Tomas Mikolov, Marc'Aurelio Ranzato, and Yoshua Bengio (2014). "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews". In: *arXiv preprint arXiv:1412.5335*.

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig (2013). "Linguistic Regularities in Continuous Space Word Representations." In: *HLT-NAACL*, pp. 746–751.

Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur (2010). "Recurrent neural network based language model." In: *Interspeech*. Vol. 2, p. 3.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013a). "Distributed representations of words and phrases and their compositionality". In: *Advances in Neural Information Processing Systems*, pp. 3111–3119.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013b). "Efficient estimation of word representations in vector space". In: *arXiv:1301.3781*.

Miller, George A (1995). "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11, pp. 39–41.

Mitchell, Jeff and Mirella Lapata (2008). "Vector-based Models of Semantic Composition." In: *ACL*, pp. 236–244.

Monroe, W., N. D. Goodman, and C. Potts (2016). "Learning to Generate Compositional Color Descriptions". In: *ArXiv e-prints*. arXiv: 1606.03821 [cs.CL].

Morin, Frederic and Yoshua Bengio (2005). "Hierarchical probabilistic neural network language model". In: *Proceedings of the international workshop on artificial intelligence and statistics*. Citeseer, pp. 246–252.

Moro, Andrea and Roberto Navigli (2015). "SemEval-2015 Task 13: Multilingual All-Words Sense Disambiguation and Entity Linking". In: *Proceedings of SemEval-2015*.

Moro, Andrea, Alessandro Raganato, and Roberto Navigli (2014). "Entity Linking meets Word Sense Disambiguation: a Unified Approach". In: *Transactions of the Association for Computational Linguistics (TACL)* 2, pp. 231–244.

Nation, I (2006). "How large a vocabulary is needed for reading and listening?" In: *Canadian Modern Language Review* 63.1, pp. 59–82.

Navigli, Roberto, Kenneth C. Litkowski, and Orin Hargraves (2007). "SemEval-2007 Task 07: Coarse-grained English All-words Task". In: *Proceedings of the 4th International Workshop on Semantic Evaluations*. SemEval '07. Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 30–35.

Navigli, Roberto and Simone Paolo Ponzetto (2010). "BabelNet: Building a very large multilingual semantic network". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, pp. 216–225.

Neelakantan, Arvind, Jeevan Shankar, Alexandre Passos, and Andrew McCallum (2015). "Efficient non-parametric estimation of multiple embeddings per word in vector space". In: *arXiv preprint arXiv:1504.06654*.

Nocedal, Jorge (1980). "Updating quasi-Newton matrices with limited storage". In: *Mathematics of computation* 35.151, pp. 773–782.

Pantel, Patrick and Dekang Lin (2002). "Discovering word senses from text". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 613–619.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu (2002). "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pp. 311–318.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pp. 1532–1543.

Pollack, Jordan B. (1990). "Recursive distributed representations". In: *Artificial Intelligence* 46.1, pp. 77 –105. ISSN: 0004-3702. DOI: http://dx.doi.org/10.1016/0004-3702(90)90005-K.

Rehůrek, Radim and Petr Sojka (2010). "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, pp. 45–50.

Reisinger, Joseph and Raymond J Mooney (2010). "Multi-prototype vector-space models of word meaning". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 109–117.

Ritter, Samuel, Cotie Long, Denis Paperno, Marco Baroni, Matthew Botvinick, and Adele Goldberg (2015). "Leveraging Preposition Ambiguity to Assess Compositional Distributional Models of Semantics". In: *The Fourth Joint Conference on Lexical and Computational Semantics*.

Rosenfeld, Ronald (2000). "Two decades of statistical language modeling: Where do we go from here?" In: *Proceedings of the IEEE* 88.8, pp. 1270–1278. DOI: 10.1109/5.880083.

Ruder, Sebastian (2017). "A survey of cross-lingual embedding models". In: *CoRR* abs/1706.04902.

Schmaltz, A., A. M. Rush, and S. M. Shieber (2016). "Word Ordering Without Syntax". In: *ArXiv e-prints*. arXiv: 1604.08633 [cs.CL].

Schütze, Hinrich (1998). "Automatic Word Sense Discrimination". In: *Comput. Linguist.* 24.1, pp. 97–123. ISSN: 0891-2017.

Schwenk, Holger (2004). "Efficient training of large neural networks for language modeling". In: *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*. Vol. 4. IEEE, pp. 3059–3064.

Shi, Tianze, Zhiyuan Liu, Yang Liu, and Maosong Sun (2015). "Learning Cross-lingual Word Embeddings via Matrix Co-factorization." In: *ACL (2)*, pp. 567–572.

Socher, Richard (2014). "Recursive Deep Learning for Natural Language Processing and Computer Vision". PhD thesis. Stanford University.

Socher, Richard, Christopher D Manning, and Andrew Y Ng (2010). "Learning continuous phrase representations and syntactic parsing with recursive neural networks". In: *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pp. 1–9.

Socher, Richard, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning (2011a). "Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection". In: *Advances in Neural Information Processing Systems 24*.

Socher, Richard, Cliff C Lin, Chris Manning, and Andrew Y Ng (2011b). "Parsing natural scenes and natural language with recursive neural networks". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 129–136.

Socher, Richard, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning (2011c). "Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Socher, Richard, Brody Huval, Christopher D Manning, and Andrew Y Ng (2012). "Semantic compositionality through recursive matrix-vector spaces". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pp. 1201–1211.

Socher, Richard, John Bauer, Christopher D. Manning, and Andrew Y. Ng (2013a). "Parsing With Compositional Vector Grammars". In: *ACL*.

Socher, Richard, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts (2013b). "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631. Citeseer, p. 1642.

Socher, Richard, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng (2014). "Grounded compositional semantics for finding and describing images with sentences". In: *Transactions of the Association for Computational Linguistics* 2, pp. 207–218.

Stenetorp, Pontus (2013). "Transition-based Dependency Parsing Using Recursive Neural Networks". In: *Deep Learning Workshop at the 2013 Conference on Neural Information Processing Systems (NIPS)*. Lake Tahoe, Nevada, USA.

Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney (2012). "LSTM neural networks for language modeling". In: *Thirteenth Annual Conference of the International Speech Communication Association*.

Tengi, Randee I (1998). "WordNet: an electronic lexical database, The MIT Press, Cambridge, Massachusetts". In: ed. by Christiane (réd.) Fellbaum. Chap. Design and implementation of the WordNet lexical database and searching software, p. 105.

Tian, Fei, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tie-Yan Liu (2014). "A Probabilistic Model for Learning Multi-Prototype Word Embeddings." In: *COLING*, pp. 151–160.

Turian, Joseph, Lev Ratinov, and Yoshua Bengio (2010). "Word representations: a simple and general method for semi-supervised learning". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, pp. 384–394.

Tymoczko, T., J. Henle, and J.M. Henle (1995). *Sweet Reason: A Field Guide to Modern Logic*. Textbooks in Mathematical Sciences. Key College. ISBN: 9780387989303.

Vandewalle, P., J. Kovacevic, and M. Vetterli (2009). "Reproducible research in signal processing". In: *IEEE Signal Processing Magazine* 26.3, pp. 37–47. ISSN: 1053-5888. DOI: 10.1109/MSP.2009.932122.

Véronis, Jean (1998). "A study of polysemy judgements and inter-annotator agreement". In: *Programme and advanced papers of the Senseval workshop*, pp. 2–4.

Wang, Rui, Wei Liu, and Chris McDonald (2017). "A Matrix-Vector Recurrent Unit Model for Capturing Compositional Semantics in Phrase Embeddings". In: *International Conference on Information and Knowledge Management*.

Wang, Sida and Christopher D Manning (2012). "Baselines and bigrams: Simple, good sentiment and topic classification". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, pp. 90–94.

Webster, W.F. (1900). *English: Composition and Literature*. Houghton Mifflin Company.

White, L., R. Togneri, W. Liu, and M. Bennamoun (2017). "Learning Distributions of Meant Color". In: *ArXiv e-prints*. arXiv: 1709.09360 [cs.CL].

White, L., R. Togneri, W. Liu, and M. Bennamoun (2018). "DataDeps.jl: Repeatable Data Setup for Replicable Data Science". In: *ArXiv e-prints*. arXiv: 1808.01091 [cs.SE].

White, Lyndon and David Ellison (2018). "Embeddings.jl: easy access to pretrained word embeddings from Julia". In: *Journal of Open Source Software (Under Review)*.

White, Lyndon and Sebastin Santy (2018). "DataDepsGenerators.jl: making reusing data easy by automatically generating DataDeps.jl registration code". In: *Journal of Open Source Software*.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2015). "How Well Sentence Embeddings Capture Meaning". In: *Proceedings of the 20th Australasian Document Computing Symposium*. ADCS '15. Parramatta, NSW, Australia: ACM, 9:1–9:8. ISBN: 978-1-4503-4040-3. DOI: 10.1145/2838931.2838932.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016a). "Generating Bags of Words from the Sums of their Word Embeddings". In: *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016b). "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem". In: *IEEE International Conference on Data Mining: High Dimensional Data Mining Workshop (ICDM: HDM)*. DOI: 10.1109/ICDMW.2016.0113.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018a). "DataDeps.jl: Repeatable Data Setup for Reproducible Data Science". In: *Journal of Open Research Software (Under Review)*.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "Finding Word Sense Embeddings Of Known Meaning". In: *19th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

White, Lyndon., Roberto. Togneri, Wei. Liu, and Mohammed Bennamoun (2018). "Learning of Colors from Color Names: Distribution and Point Estimation". In: *Computational Lingustics (Under Review)*.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018a). *Neural Representations of Natural Language*. Studies in Computational Intelligence (Book). Springer Singapore. ISBN: 9789811300615.

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "NovelPerspective: Identifying Point of View Characters". In: *Proceedings of ACL 2018, System Demonstrations*. Association for Computational Linguistics.

Wieting, John, Mohit Bansal, Kevin Gimpel, and Karen Livescu (2016). "Towards Universal Paraphrastic Sentence Embeddings". In: *International Conference on Learning Representations (ICLR)*.

Wilson, Greg, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, and Paul Wilson (2014). "Best Practices for Scientific Computing". In: *PLOS Biology* 12.1, pp. 1–7. DOI: 10.1371/journal.pbio.1001745.

Wohlgenannt, Gerhard, Ekaterina Chernyak, and Dmitry Ilvovsky (2016). "Extracting social networks from literary text with word embedding tools". In: *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pp. 18–25.

Wren, Jonathan D (2008). "URL decay in MEDLINE: a 4-year follow-up study". In: *Bioinformatics* 24.11, pp. 1381–1385.

Yin, Wenpeng and Hinrich Schütze (2015). "Learning Word Meta-Embeddings by Using Ensembles of Embedding Sets". In: eprint: 1508.04257.

Yogatama, Dani, Fei Liu, and Noah A Smith (2015). "Extractive Summarization by Maximizing Semantic Volume". In: *Conference on Empirical Methods in Natural Language Processing*.

Yuret, Deniz (2016). "Knet: beginning deep learning with 100 lines of Julia". In: *Machine Learning Systems Workshop at NIPS 2016*.

Zanzotto, Fabio Massimo, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh Manandhar (2010). "Estimating linear models for compositional distributional semantics". In: *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, pp. 1263–1271.

Zhang, Chiyuan (2014). *Mocha.jl: Deep Learning framework for Julia*. URL: https://github.com/pluskid/Mocha.jl.

Zhang, Jiajun, Shujie Liu, Mu Li, Ming Zhou, and Chengqing Zong (2014). "Bilingually-constrained Phrase Embeddings for Machine Translation". In: ACL.

Zhang, Xiang and Yann LeCun (2015). "Text Understanding from Scratch". In: *CoRR* Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics.

Zhang, Yue and Stephen Clark (2015). "Discriminative Syntax-based Word Ordering for Text Generation". In: *Comput. Linguist.* 41.3, pp. 503–538. ISSN: 0891-2017. DOI: 10.1162/COLI_a_00229.

Zhu, Yukun, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler (2015). "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *Proceedings of the IEEE international conference on computer vision*, pp. 19–27.

Zipf, George Kingsley (1945). "The meaning-frequency relationship of words". In: *The Journal of general psychology* 33.2, pp. 251–256.

Zipf, G.K. (1949). *Human behavior and the principle of least effort: an introduction to human ecology*. Addison-Wesley Press.

Zou, Will Y, Richard Socher, Daniel M Cer, and Christopher D Manning (2013). "Bilingual Word Embeddings for Phrase-Based Machine Translation." In: *EMNLP*, pp. 1393–1398.

Part III

Appendix: Tooling

Appendix A

DataDeps.jl: Repeatable Data Setup for Replicable Data Science

This paper is currently under review for the Journal of Open Research Software.

Abstract

We present DataDeps.jl: a julia package for the reproducible handling of static datasets to enhance the repeatability of scripts used in the data and computational sciences. It is used to automate the data setup part of running software which accompanies a paper to replicate a result. This step is commonly done manually, which expends time and allows for confusion. This functionality is also useful for other packages which require data to function (e.g. a trained machine learning based model). DataDeps.jl simplifies extending research software by automatically managing the dependencies and makes it easier to run another author's code, thus enhancing the reproducibility of data science research.

A.1 Introduction

In the movement for reproducible sciences there have been two key requests upon authors: **1.** Make your research code public, **2.** Make your data public (Goodman et al. 2014). In practice this alone is not enough to ensure that results can be replicated. To get another author's code running on a your own computing environment is often non-trivial. One aspect of this is data setup: how to acquire the data, and how to connect it to the code.

DataDeps.jl simplifies the data setup step for software written in Julia (Bezanson et al. 2014). DataDeps.jl follows the unix philosophy of doing one job well. It allows the code to depend on data, and have that data automatically downloaded as required. It increases replicability of any scientific code that uses static data (e.g. benchmark datasets). It provides simple methods to orchestrate the data setup: making it easy to create software that works on a new system without any user effort. While it has been argued that the direct replicability of executing the author's code is a poor substitute for independent reproduction (Drummond 2009), we maintain that being able to run the original code is

important for checking, for understanding, for extension, and for future comparisons.

Vandewalle, Kovacevic, and Vetterli (2009) distinguishes six degrees of replicability for scientific code. The two highest levels require that "The results can be easily reproduced by an independent researcher with at most 15 min of user effort". One can expend much of that time just on setting up the data. This involves reading the instructions, locating the download link, transferring it to the right location, extracting an archive, and identifying how to inform the script as to where the data is located. These tasks are automatable and therefore should be automated, as per the practice "Let the computer do the work" (Wilson et al. 2014).

DataDeps.jl handles the data dependencies, while Pkg[1] and BinDeps.jl,[2] (etc.) handle the software dependencies. This makes automated testing possible, e.g., using services such as TravisCI[3] or AppVeyor.[4] Automated testing is already ubiquitous amongst julia users, but rarely for parts where data is involved. A particular advantage over manual data setup, is that automation allow scheduled tests for URL decay (Wren 2008). If the full deployment process can be automated, given resources, research can be fully and automatically replicated on a clean continuous integration environment.

A.1.1 Three common issues about research data

DataDeps.jl is designed around solving common issues researchers have with their file-based data. The three key problems that it is particularly intended to address are:

Storage location: Where do I put it? Should it be on the local disk (small) or the network file-store (slow)? If I move it, am I going to have to reconfigure things?

Redistribution: I don't own this data, am I allowed to redistribute it? How will I give credit, and ensure the users know who the original creator was?

Replication: How can I be sure that someone running my code has the same data? What if they download the wrong data, or extract it incorrectly? What if it gets corrupted or has been modified and I am unaware?

A.2 DataDeps.jl

A.2.1 Ecosystem

DataDeps.jl is part of a package ecosystem as shown in Figure A.1. It can be used directly by research software, to access the data they depend upon for e.g. evaluations. Packages such as MLDatasets.jl[5] pro-

[1] `https://github.com/JuliaLang/Pkg.jl`
[2] `https://github.com/JuliaLang/BinDeps.jl`
[3] `https://travis-ci.org/`
[4] `https://ci.appveyor.com/`
[5] `https://github.com/JuliaML/MLDatasets.jl`

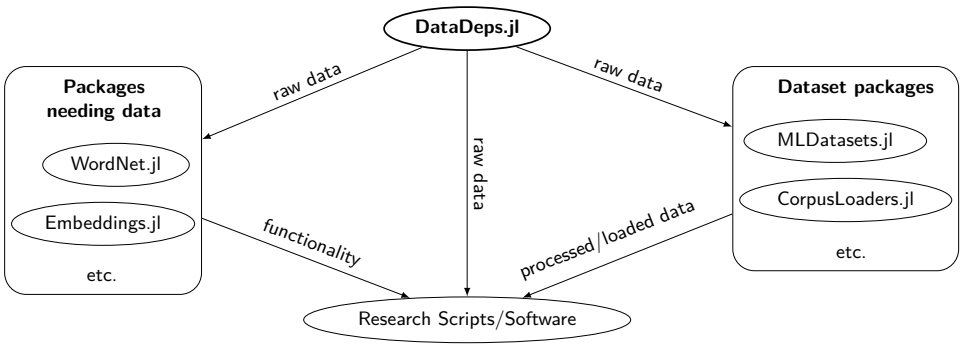


Figure A.1: The current package ecosystem depending on DataDeps.jl.

vide more convenient accesses with suitable preprocessing for commonly used datasets. These packages currently use DataDeps.jl as a back-end. Research code also might use DataDeps.jl indirectly by making use of packages, such as WordNet.jl[6] which currently uses DataDeps.jl to ensure it has the data it depends on to function (see Appendix A.4.1); or Embeddings.jl which uses it to load pretrained machine-learning models. Packages and research code alike depend on data, and DataDeps.jl exists to fill that need.

A.2.2 Functionality

Once the dependency is declared, data can accessed by name using a datadep string written `datadep"Name"`. This can treated just like a filepath string, however it is actually a string macro. At compile time it is replaced with a block of code which performs the operation shown in Figure A.2. This operation always returns an absolute path string to the data, even that means the data must be download and placed at that path first.

DataDeps.jl solves the issues in Appendix A.1.1 as follows:

Storage location: A data dependency is referred to by name, which is resolved to a path on disk by searching a number of locations. The locations search is configurable.

Redistribution: DataDeps.jl downloads the package from its original source so it is not redistributed. A prompt is shown to the user before download, which can be set to display information such as the orignal author and any papers to cite etc.

Replication: when a dependency is declared, the creator specified the URL to fetch from and post fetch processing to be done (e.g. extraction). This removed the chance for human error. To ensure the data is exactly as it was originally checksum is used.

DataDeps.jl is primarily focused on public, static data. For researchers who are using private data, or collecting that data while developing the scripts, a manual option is provided; which only includes the **Storage Location** functionality. They can still refer to it using the `datadep"Name"`, but it will not be automatically downloaded. During publication the re-

[6] `https://github.com/JuliaText/WordNet.jl`

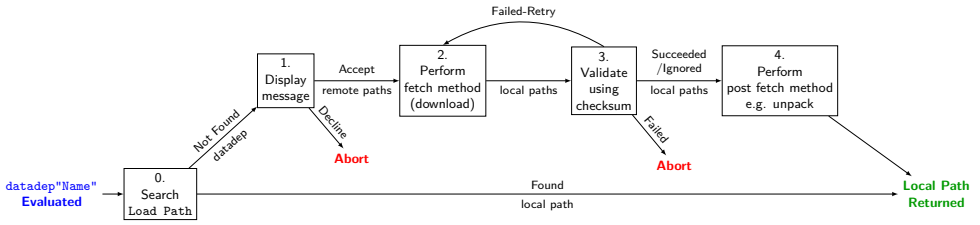


Figure A.2: The process that is executed when a data dependency is accessed by name.

searcher can upload their data to an archival repository and update the registration.

A.2.3 Similar Tools

Package managers and build tools can be used to create adhoc solutions, but these solution will often be harder to use and fail to address one or more of the concerns in Appendix A.1.1. Data warehousing tools, and live data APIs; work well with continuous streams of data; but they are not suitable for simple static datasets that available as a collection of files.

Quilt[7] is a more similar tool. In contrast to DataDeps.jl, Quilt uses one centralised data-store, to which users upload the data, and they can then download and use the data as a software package. It does not directly attempt to handle any **Storage Location**, or **Redistribution** issues. Quilt does offer some advantages over DataDeps.jl: excellent convenience methods for some (currently only tabular) file formats, and also handling data versioning. At present DataDeps.jl does not handle versioning, being focused on static data.

A.2.4 Quality Control

Using AppVeyor and Travis CI testing is automatically performed using the latest stable release of Julia, for the Linux, Windows, and Mac environments. The DataDeps.jl tests include unit tests of key components, as well as comprehensive system/integration tests of different configurations of data dependencies. These latter tests also form high quality examples to supplement the documentation for users to looking to see how to use the package. The user can trigger these tests to ensure everything is working on their local machine by the standard julia mechanism: running `Pkg.test(``DataDeps'')` respectively.

The primary mechanism for user feedback is via Github issues on the repository. Bugs and feature requests, even purely by the author, are tracked using the Github issues.

[7] `https://github.com/quiltdata/quilt`

A.3 Availability

A.3.1 Operating system

DataDeps.jl is verified to work on Windows 7+, Linux, Mac OSX.

A.3.2 Programming language

Julia v0.6, and v0.7 (1.0 support forthcoming).

A.3.3 Dependencies

DataDeps.jl's dependencies are managed by the julia package manager. It depends on SHA.jl for the default generation and checking of checksums; on Reexport.jl to reexport SHA.jl's methods; and on HTTP.jl for determining filenames based on the HTTP header information.

List of contributors

- Lyndon White (The University of Western Australia) Primary Author
- Christof Stocker (Unaffiliated), Contributor, significant design discussions.
- Sebastin Santy (Birla Institute of Technology and Science), Google Summer of Code Student working on DataDepsGenerators.jl

A.3.4 Software location:

Name: oxinabox/DataDeps.jl
Persistent identifier: `https://github.com/oxinabox/DataDeps.jl/`
Licence: MIT
Date published: 28/11/2017
Documentation Language English
Programming Language Julia
Code repository GitHub

A.4 Reuse potential

DataDeps.jl exists only to be reused, it is a "backend" library. The cases in which is should be reused are well discussed above. It is of benefit to any application, research tool, or scientific script that has a dependency on data for it's functioning or for generation of its result.

DataDeps.jl is extendible via the normal julia methods of subtyping, and composition. Additional kinds of `AbstractDataDep` can be created, for example to add an additional validation step, while still reusing the behaviour defined. Such new types can be created in their own packages, or contributed to the open source DataDeps.jl package.

Julia is a relatively new language with a rapidly growing ecosystem of packages. It is seeing a lot of up take in many fields of computation sciences, data science and other technical computing. By establishing tools like DataDeps.jl now, which support the easy reuse of code, we hope to promote greater resolvability of packages being created later. Thus in turn leading to more reproducible data and computational science in the future.

A.4.1 Case Studies

Research Paper: White et al. (2016a) We criticize our own prior work here, so as to avoid casting aspersions on others. We consider it's limitations and how it would have been improved had it used DataDeps.jl. Two version of the script were provided[8] one with just the source code, and the other also including 3GB of data. It's license goes to pains to explain which files it covers and which it does not (the data), and to explain the ownership of the data. DataDeps.jl would avoid the need to include the data, and would make the ownership clear during setup. Further sharing the source code alone would have been enough, the data would have been downloaded when (and only if) it is required. The scripts themselves have relative paths hard-coded. If the data is moved (e.g. to a larger disk) they will break. Using DataDeps.jl to refer to the data by name would solve this.

Research Tool: WordNet.jl WordNet.jl is the Julia binding for the WordNet tool (Miller 1995). As of PR #8[9] it now uses DataDeps.jl. It depends on having the WordNet database. Previously, after installing the software using the package manager, the user had to manually download and set this up. The WordNet.jl author previously had concerns about handling the data. Including it would inflate the repository size, and result in the data being installed to an unreasonable location. They were also worried that redistributing would violate the copyright. The manual instructions for downloading and extracting the data included multiple points of possible confusion. The gzipped tarball must be correctly extracted. The user must know to pass in the *grand-parent* directory of the database files. Using DataDeps.jl all these issues have now been solved.

Acknowledgements

Thank particularly to Christof Stocker, the creator of MLDatasets.jl (and numerous other packages), in particular for his bug reports, feature requests and code reviews; and for the initial discussion leading to the creation of this tool.

Competing interests

The authors declare that they have no competing interests.

[8] Source code and data provided at `http://white.ucc.asn.au/publications/White2016BOWgen/`
[9] `https://github.com/JuliaText/WordNet.jl/pull/8`

A.5 Concluding Remarks

DataDeps.jl aims to help solve reproducibility issues in data driven research by automating the data setup step. It is hoped that by supporting good practices, with tools like DataDeps.jl, now for the still young Julia programming language better scientific code can be written in the future .

Appendix B

DataDepsGenerators.jl: Making Reusing Data Easy by Automatically Generating DataDeps.jl Registration Code

This paper is currently under review for the Journal of Open Source Software.

B.1 Summary

DataDepsGenerators.jl is a tool written to help users of the Julia programming language (Bezanson et al. 2014), to observe best practices when making use of published datasets. Using the metadata present in published datasets, it generates the code for the data dependency registration blocks required by DataDeps.jl (White et al. 2018). These registration blocks are effectively executable metadata, which can be resolved by DataDeps.jl to download the dataset. They include a message that is displayed to the user whenever the data set is automatically downloaded. This message should include provenance information on the dataset, so that downstream users know its original source and details on its processing.

DataDepsGenerators.jl attempts to use the metadata available for a dataset to capture and record:

- The dataset name.
- A URL for a website about the dataset.
- The names of the authors and maintainers
- The creation date, publication date, and the date of the most recent modification.
- The license that the dataset is released under.

- The formatted bibliographic details of any paper about or relating to the dataset.
- The formatted bibliographic details of how to cite the dataset itself.
- A list of URLs where the files making up the dataset can be downloaded.
- A corresponding list of file hashes, such as MD5 or SHA256, to validate the files after download.
- A description of the dataset.

Depending on the APIs supported by the repository some of this information may not be available. DataDepsGenerators.jl makes a best-effort attempt to acquire as much provenance information as possible. Where multiple APIs are supported, it makes use of all APIs possible, merging their responses to fill any gaps. It thus often produces higher quality and more comprehensive dataset metadata than is available from any one source.

DataDepsGenerators.jl leavages many different APIs to support a very large number of repositories. By current estimates tens of millions of datasets are supported, from hundreds of repositories. The APIs supported include:

- DataCite / CrossRef
  - This is valid for the majority of all dataset with a DOI.
- DataOne
  - This supports a number of data repositories used in the earth sciences.
- FigShare
  - A popular general purpose data repository.
- DataDryad
  - A data repository particularly popular with evolutionary biology and ecology.
- UCI ML repository
  - A data repository commonly used for small-medium machine learning benchmark datasets.
- GitHub
  - Most well known for hosting code; but is fairly regularly used to host versioned datasets.
- CKAN
  - This is the system behind a large number of government open data initiatives;
  - such as Data.Gov, data.gov.au, and the European Data Portal
- Embedded JSON-LD fragments in HTML pages.
  - This is commonly used on many websites to describe their datasets.
  - Including many of those listed above.
  - But also Zenodo, Kaggle Datasets, all DataVerse sites and many others.

DataDepsGenerators.jl as the name suggests, generates static code which the user can add into their project's julia source code to make use of with DataDeps.jl. There are a number of reasons why static code generation is preferred over directly using the APIs. - On occasion the information reported by the APIs is wrong or incomplete. By generating code

that the user may edit they may tweak the details as required. - The process of accessing the APIs requires a number of heavy dependencies, such as HTML and JSON parsers. If these APIs were to be access directly by a project, it would require adding this large dependency tree to the project. - It is important to know if a dataset has changed. As such retrieving the file hash and last modification date would be pointless if they are updated automatically. Finally: having the provenance information recorded in plain text, makes the dataset metadata readily accessible to anyone reading the source code; without having to run the project's application.

The automatic downloading of data is important to allow for robustly replicable scientific code. The inclusion of provenance information is required to give proper credit and to allow for good understanding of the dataset's real world context. DataDepsGenerators.jl makes this easy by automating most of the work.

B.1.1 Other similar packages

In the R software ecosystem there is the `suppdata` [@suppdata] package. `suppdata` is a package for easily downloading supplementary data files attached to journal articles. It is thus very similar in purpose: to make research data more accessible. It is a direct download tool, rather than DataDepsGenerators.jl's approach of generating metadata that is evaluated to preform the download. While there is some overlap, in that both support FigShare and Dryad, `suppdata` primarily supports journals rather than data repositories.

When it comes to accessing data repositories, there exists several R packages which only support a single provider of data. These vary in their support for different functionality. They often support things beyond the scope of DataDepsGenerators.jl, to search, or upload data to the supported repository. Examples include:

- `rdryad` for DataDryad
- `rfigshare` for FigShare
- `ckanr` for CKAN
- `rdatacite` for DataCite
- `rdataone` for DataOne

To the best of our knowledge at present their is not any unifying R package that supports anywhere near the range of data repostories supported by DataDepsGenerators.jl. Contemporaneously, with the creation of DataDepsGenerator.jl, the was proposed package to acquire data based on a DOI. While this has yet to eventuate into usable software, several of the discussions relating to it were insightful, and contributed to the functionality of DataDepsGenerators.jl.

To the best of our knowledge at present there does not exist a unifying R package that supports anywhere near the range of data repositories supported by DataDepsGenerators.jl. Contemporaneously, during the creation of DataDepsGenerator.jl, there was another R package (`doidata`) that was proposed in order to acquire data based on a DOI. While this has

yet to eventuate into usable software, several of the discussions relating to it were insightful, and contributed to the functionality of DataDepsGenerators.jl.

B.1.2 Acknowledgements

This work was largely carried out as a Google Summer of Code project, as part of the NumFocus organisation. It also benefited from funding from Australian Research Council Grants DP150102405 and LP110100050.

We also wish to thank the support teams behind the APIs and repositories listed above. In the course of creating this tool we thoroughly exercised a number of APIs. In doing so we encountered a number of bugs and issues; almost all of which have now been fixed, by the attentive support and operation staff of the providers.

Appendix C

Embeddings.jl: Easy Access to Pretrained Word Embeddings from Julia

This paper is currently under review for the Journal of Open Source Software.

C.1 Summary

Embeddings.jl is a tool to help users of the Julia programming language (Bezanson et al. 2014) make use of pretrained word embeddings for NLP. Word embeddings are a very important feature representation in natural language processing. The use of embeddings pretrained on very large corpora can be seen as a form of transfer learning. It allows knowledge of lexical semantics derived from the distributional hypothesis– that words occurring in similar contexts have similar meaning– to be injected into models which may have only limited amounts of supervised, task oriented training data.

Many creators of word embedding methods have generously made sets of pretrained word representations publicly available. Embeddings.jl exposes these as a standard matrix of numbers and a corresponding array of strings. This lets Julia programs use word embeddings easily, either on their own or alongside machine learning packages such as Flux (Innes 2018). In such deep learning packages, it is common to use word embeddings as an input layer of an LSTM or other neural network, where they may be kept invariant or used as initialization for fine-tuning on the supervised task. They can be summed to represent a bag of words, concatenated to form a matrix representation of a sentence or document, or used otherwise in a wide variety of natural language processing tasks.

Embeddings.jl makes use of DataDeps.jl (White et al. 2018), to allow for convenient automatic downloading of the data when and if required. It also uses the DataDeps.jl prompt to ensure the user of the embeddings has full knowledge of the original source of the data, and which papers to cite etc.

It currently provides access to

- multiple sets of word2vec embeddings (Mikolov et al. 2013b) for English
- multiple sets of GLoVE (Pennington, Socher, and Manning 2014) embeddings for English
- multiple sets of FastText embeddings (Bojanowski et al. 2017; Grave et al. 2018) for several hundred languages

It is anticipated that as more pretrained embeddings are made available for more languages and using newer methods, the Embeddings.jl package will be updated to support them.

Appendix D

TensorFlow.jl: An Idiomatic Julia Front End for TensorFlow

This paper is currently under review for the Journal of Open Source Software.

D.1 Summary

TensorFlow.jl is a Julia (Bezanson et al. 2014) client library for the TensorFlow deep-learning framework (Abadi et al. 2015; Abadi et al. 2016). It allows users to define TensorFlow graphs using Julia syntax, which are interchangable with the graphs produced by Google's first-party Python TensorFlow client and can be used to perform training or inference on machine-learning models.

Graphs are primarily defined by overloading native Julia functions to operate on a TensorFlow.jl `Tensor` type, which represents a node in a TensorFlow computational graph. This overloading is powered by Julia's powerful multiple-dispatch system, which in turn allows allows the vast majority of Julia's existing array-processing functionality to work as well on the new `Tensor` type as they do on native Julia arrays. User code is often unaware and thereby reusable with respect to whtether its inputs are TensorFlow tensors or native Julia arrays by utilizing *duck-typing*.

TensorFlow.jl has an elegant, idiomatic Julia syntax. It allows all the usual infix operators such as `+`, `-`, `*` etc. It works seamlessly with Julia's broadcast syntax as well, such as the `.*` operator. This `*` can correspond to matrix multiplication while `.*` corresponds to element-wise multiplication, while Python clients needs distinct `@` (or `matmul`) and `*` (or `multiply`) functions. It also allows Julia-style indexing (e.g. `x[:, ii + end÷2]`), and concatenation (e.g. `[A B]`, `[x; y; 1]`). Its goal is to be idiomatic for Julia users, while still preserving all the power and maturity of the TensorFlow system. For example, it allows Julia code to operate on TPUs by virtue of using the same TensorFlow graph syntax as Python's TensorFlow client, even though there is no native Julia TPU compiler.

TensorFlow.jl to carefully balance between matching the Python TensorFlow API and Julia conventions. In turn, the Python TensorFlow client is itself designed to closely mirror numpy. Some examples are shown in the table below.

Julia	**Python TensorFlow**	**TensorFlow.jl**
1-based indexing	0-based indexing	1-based indexing
Column Major	Row Major	Row Major
Explicit broadcasting	Implicit broadcasting	Implicit or explicit broadcasting
Last index at `end`, 2nd last in `end-1`	Last index at `-1`, second last in `-2`	last index at `end`, 2nd last in `end-1`
Operations in Julia ecosystem namespaces. (`SVD` in `LinearAlgebra`, `erfc` in `SpecialFunctions`, `cos` in `Base`)	All operations TensorFlow's namespaces (`SVD` in `tf.linalg`, `erfc` in `tf.math`, `cos` in `tf.math`, and all reexported from `tf`)	All hand imported Operations in the Julia ecosystems namespaces. (`SVD` in `LinearAlgebra`, `erfc` in `SpecialFunctions`, `cos` in `Base`) Ops that have no other place are in `TensorFlow`. Automatically generated ops are in `Ops`
Container types are parametrized by number of dimensions and element type	N/A: does not have a parametric type system	Tensors are parametrized by element type, enabling easy specialization of algorithms for different types.

Defining TensorFlow graphs in the Python TensorFlow client can be viewed as metaprogramming, in the sense that a host language (Python) is being used to generate code in a different embedded language (the TensorFlow computational graph) (Innes et al. 2017). This often comes with some awkwardness, as the syntax and the semantics of the embedded language by definition do not match the host language or there would be no need for two languages to begin with. Using TensorFlow.jl is similarly a form of meta-programming for the same reason. However, the flexibility and meta-programming facilities offered by Julia's macro system makes Julia especially well-suited as a host language, as macros implemented in TensorFlow.jl can syntactically transform idiomatic Julia code into Julia code that constructs TensorFlow graphs. This permits users to reuse their knowledge of Julia, while users of the Python TensorFlow client essentially need to learn both Python and TensorFlow.

One example of our ability to leverage the increased expressiveness of Ju-

lia is using `@tf` macro blocks implemented in TensorFlow.jl to automatically name nodes in the TensorFlow computatioal graph. Nodes in a TensorFlow graph have names; these correspond to variable names in a traditional programming language. Thus every operation, variable and placeholder takes a `name` parameter. In most TensorFlow bindings, these must be specified manually resulting in a lot of code that includes duplicate information such as `x = tf.placeholder(tf.float32, name="x")` or they are defaulted to an uninformative value such as `Placeholder_1`. In TensorFlow.jl, prefixing a lexical block (such as a `function` or a `begin` block) with the `@tf` macro will cause the `name` parameter on all operations occurring on the right-hand side of an assignment to be filled in using the left-hand side. For example, the TensorFlow.jl equivalent of the above example is `@tf x = placeholder(Float32)`. Note how `x` is named only once instead of twice, as is redundantly required in the Python example. Since all nodes in the computational graph can automatically be assigned the same name as the corresponding Julia variable with no additional labor from TensorFlow.jl users, users get for free more intuitive debugging and graph visualisation.

to

Another example of the use of Julia's metaprogramming is in the automatic generation of Julia code for each operation defined by the official TensorFlow C implementation (for example, convolutions of two TensorFlow tensors). The C API can be queried to return definitions of all operations as protobuffer descriptions, which includes the expected TensorFlow type and arity of its inputs and outputs, as well as documentation. This described the operations at a sufficient level to generate the Julia code to bind to the functions in the C API and automatically generate a useful docstring for the function,. One challenge in this is that such generated code must correct the indices to be 1-based instead of 0-based to accord with Julia convention. Various heuristics are employed by TensorFlow.jl to guess which input arguments represent indicies and so should be converted.

TensorFlow.jl ships by default with bindings for most operations, but any operation can be dynamically imported at runtime using `@tfimport OperationName`, which will generate the binding and load it immediately. Additionally, for operations that correspond to native Julia operations (for example, `sin`), we overload the native Julia operation to call the proper binding.

We also use Julia's advanced parametric type system to enable elegant implementations of array operations not easily possible in other client libraries. TensorFlow.jl represents all nodes in the computational graph as parametric `Tensor` types which are parametrised by their element type, e.g. `Tensor{Int}`, `Tensor{Float64}` or `Tensor{Bool}`. This allows Julia's dispatch system to be used to simplify defining some bindings. For example, indexing a `Tensor` with an `Int`-like Tensor will ultimately create a node corresponding to a TensorFlow "gather" operation, and indexing with a `Bool`-like Tensor will correspond to a "boolean_mask" operation. It is also used to cast inputs in various functions to compatible shapes.

D.1.1 Challenges

A significant difficulty in implementing the TensorFlow.jl package for Julia is that in the upstream TensorFlow version 0 and 1 distributions, the C API is primarily designed for the execution of pretrained models and does not include many conveients for the definition of training of graphs.

The C API primarily exposes low-level array operations such as matrix multiplication or reductions. Graidnet descent ptimizers, RNNs functionality, and (until recently) shape-inference all required reimplementation on the Julia side. Most challengingly, the symbolic differentiation implemented in the `gradients` function is not available from the C API for all operations. To work around this, we currently use Julia's Python interop library to generate the gradient nodes using the Python client for those operations not supported by the C API. This requires serializing and deserializing TensorFlow graphs on both the Julia and Python side.

This has been improving over time, both due to Google moving more functionality from the Python TensorFlow client to the C API which can reused by Julia, and with more reimplementations of other aspects of the Python client from our own volunteer efforts. There nevertheless remains a large number of components from the upstream `contrib` submodule that remain unimplemented, including various efforts around probabilistic programming.

D.1.2 Other deep learning frameworks in Julia

Julia also has bespoke neural network packages such as Mocha (Zhang 2014), Knet (Yuret 2016) and Flux (Innes 2018), as well as bindings to other frameworks such as MxNet (Chen et al. 2015). While not having the full-capacity to directly leverage some of the benefits of the language and its ecosystem present in the pure julia frameworks such as Flux, TensorFlow.jl provides an interface to one of the most mature and widely deployed deep learning environments. It thus trivially supports technologies such as TPUs and visualization libraries like TensorBoard. It also gains the benefits from the any optimisations made in the graph execution engine of the underlying TensorFlow C library, which includes extensive support for automatically distributing computations over multiple host machines which each have multiple GPUs.

D.1.3 Acknowledgements

- We gratefully acknowledge the 30 contributors to the TensorFlow.jl Github repository.
- We especially thank Katie Hyatt for contributing tests and documentation.
- We thank members of Julia Computing and the broader Julia Community for various discussions, especially Mike Innes and Keno Fischer.