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 15, 2019

THE UNIVERSITY OF WESTERN AUSTRALIA

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 15, 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. On a variety of tasks, these 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:
NRoNL
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:
White2015SentVecMeaning
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:
White2018ColorEst
Location in thesis: Chapter 6
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:
WhiteRefittingSenses
Location in thesis: Chapter 7
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:
novelperspective
Location in thesis: Chapter 8
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:

White2015BOWgen
Location in thesis: Chapter 9
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:
White2016a
Location in thesis: Chapter 10
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:
DataDeps
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:
DataDepsGenerators
Location in thesis: Appendix B
Student contribution to work:
Original author of software. Provided direction, guidance, and code review for its enhancement. Wrote publication.

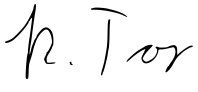
Details of the work:
EmbeddingsPackage
Location in thesis: Appendix C
Student contribution to work:
Original and primary author of software. Wrote publication.

Details of the work:
TensorFlowJulia
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 How Well Sentence Embeddings Capture Meaning **77**

6 Learning of Colors from Color Names: Distribution and Point Estimation **91**

7 Finding Word Sense Embeddings Of Known Meaning **121**

8 NovelPerspective: Identifying Point of View Characters **135**

9 Generating Bags of Words from the Sums of their Word Embeddings **145**

10 Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem **157**

11 Conclusion **175**

III Appendix: Tooling **179**

A DataDeps.jl: Repeatable Data Setup for Replicable Data Science **181**

B DataDepsGenerators.jl: Making Reusing Data Easy by Automatically Generating DataDeps.jl Registration Code **189**

C Embeddings.jl: Easy Access to Pretrained Word Embeddings from Julia **193**

D TensorFlow.jl: An Idiomatic Julia Front End for TensorFlow **195**

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 (**maron1961automatic**). 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* (**ac2018probingsentencevectors**), or *embedding-BOW* (**acl2018bleuopposedmeaning**) 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 (**mikolovSkip**). 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 vocabulary $\mathbb{V}$

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.

arora2016simple's work describes a **arora2016simple**, 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 **wieting2015towards**, they find this to be very competitive when compared with more complex models which take into account word order.

acl2018bleuopposedmeaning 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.

ac2018probingsentencevectors 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 nevertheless 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 6.

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 **grice1975logic**, 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 Chapter 10. 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 Chapter 5 and Chapter 6, 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. Chapter 7 and Chapter 8 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, Chapter 9 and Chapter 10 investigate if it is possible to reverse the conversion from sentence to SOWE. The results in Chapter 9 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 Chapter 10 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 (**bengio1994learning**) and RvNNs (**socher2014recursive**), 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 **NRoNL** (**NRoNL**). 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 Chapter 7. 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 Chapter 5, through a paraphrase grouping tasks. Similarly, the representation of *phrases* is investigated in Chapter 6 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
Chapter 5	Sentences	Paraphrase grouping	Word2Vec (**mikolovSkip**)
Chapter 6	Short Phrases	Color understanding	FastText (**bojanowski2016enriching**)
Chapter 7	Word Senses	Similarity with context & Word sense disambiguation	AdaGram (**AdaGrams**) & Bespoke greedy sense embeddings
Chapter 8	Adj. Contexts	POV character detection	FastText (**bojanowski2016enriching**)
Chapter 9	Sentences	Recovering bags of words	GLoVE (**pennington2014glove**)
Chapter 10	Sentences	Recovering sentences	GLoVE (**pennington2014glove**)

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 Chapter 7. 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 Chapter 8. We conclude with a pair of complementary works in Chapters 9 and 10, 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: Chapter 5. White2015SentVecMeaning

Originally published as: **White2015SentVecMeaning**.

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 (**le2014distributed**) is in line with the observation in the footnotes of the less well-known follow up work of **mesnil2014ensemble**. Where it was found that the performance reported in **le2014distributed** 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 (**DBLP:journals/corr/KirosZSZTUF15**), or machine translation models. Another limitation of the work is that the unfolding recursive autoencoder (**SocherEtAl2011:PoolRAE**) 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 **adi2017Probing**; **ac2018probingsentencevectors**. 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 6. White2018ColorEst

Originally published as: **White2018ColorEst**.

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 (**2016arXiv160603821M**). 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: Chapter 7. WhiteRefittingSenses

Originally published as: **WhiteRefittingSenses**.

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 (**Reisinger2010**). 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 **agirre2006**; 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 (**AdaGrams**), 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 Refitted-Sim, a new similarity measure using this refitting to evaluate the similarity of words in context.

Part II: Chapter 8. novelperspective

Originally published as: **novelperspective**.

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: Chapter 9. White2015BOWgen

Originally published as: **White2015BOWgen**.

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 **adi2017Probing**. 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: Chapter 10. White2016a

Originally published as: **White2016a**.

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 Chapter 9 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 **Horvat2014**. 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 Chapter 9 and Chapter 10, 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 (**Julia**) 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**: "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 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. Chapter 5 investigates this for sentence embeddings. This is done by taking points in the

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

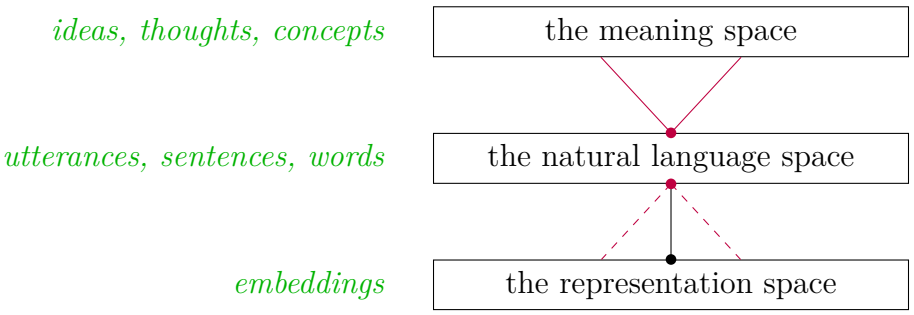


Figure 1.1: The representation space is a computationally manipulate representation of the meaning space. The natural language utterances come from points in the meaning space; though due to ambiguity we can only truly hope to estimate distributions over meanings when interpreting them. A single point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

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. Chapter 7 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 Chapter 7) 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, Chapter 5 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 6 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 distri-

butions 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 Chapters 9 and 10, 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.

Chapter 5 and Chapter 6 both consider representing contiguous linguistic structures, in particular sentences and shore phrases. Chapter 5 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 6 are (once trained) points in a grounded representation space, though that work did not examine it directly. Chapter 7 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. Chapter 8 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

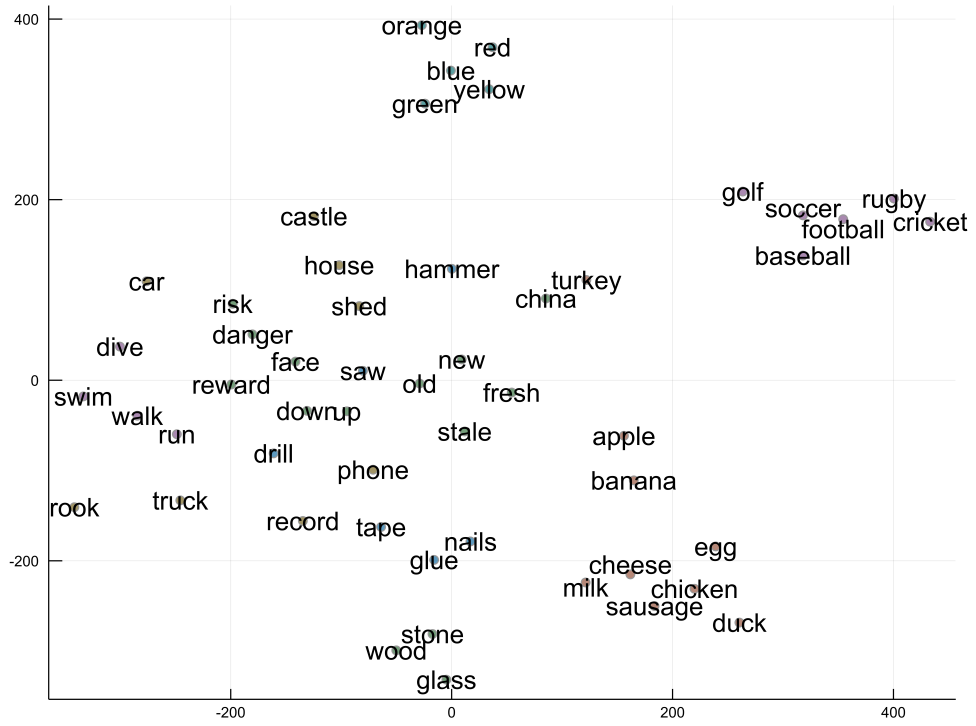
Abstract

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 (**dumais1988using**) and LDA (**blei2003latent**), 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 (**bojanowski2016enriching**). They were originally 300 dimensions but have been reduced to 2 using t-SNE (**maaten2008tsne**) 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 (**rosenfeld2000two**). 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 (**ha2009extending**). 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 (**katz1987estimation**; **kneser1995improved**) 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 (**brown1992class**). 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

NPLM 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

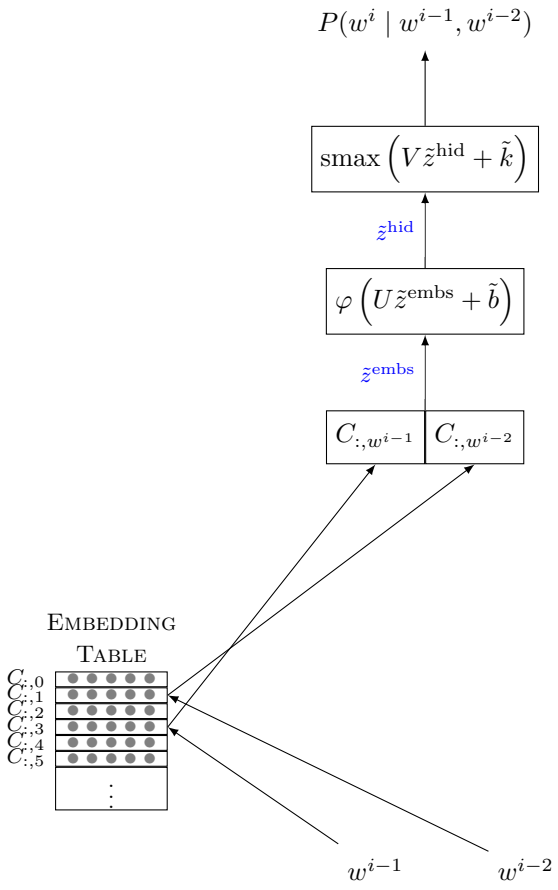
To understand the neural probabilistic language model, let's first consider a simplified neural trigram language model. This model is a simplification of the model introduced by **NPLM**. It follows the same principles, and highlights the most important idea in neural language representations. This is 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

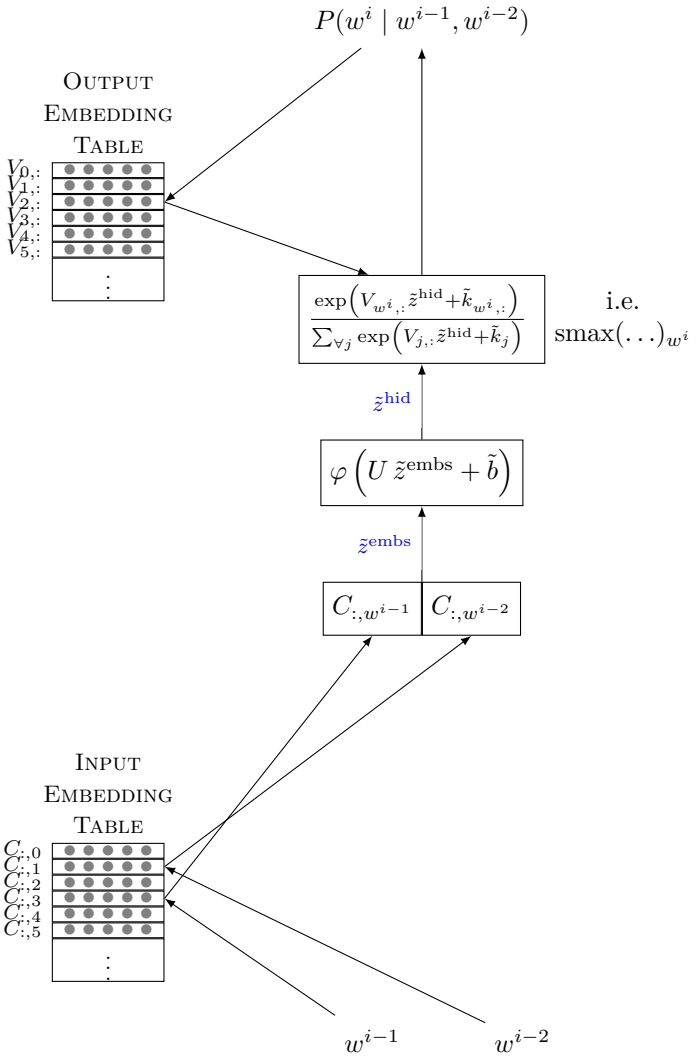
Figure 2.2: The Neural Trigram Language Model



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 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

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



which are looked up during the input.

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 **NRoNL**:

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

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:

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

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

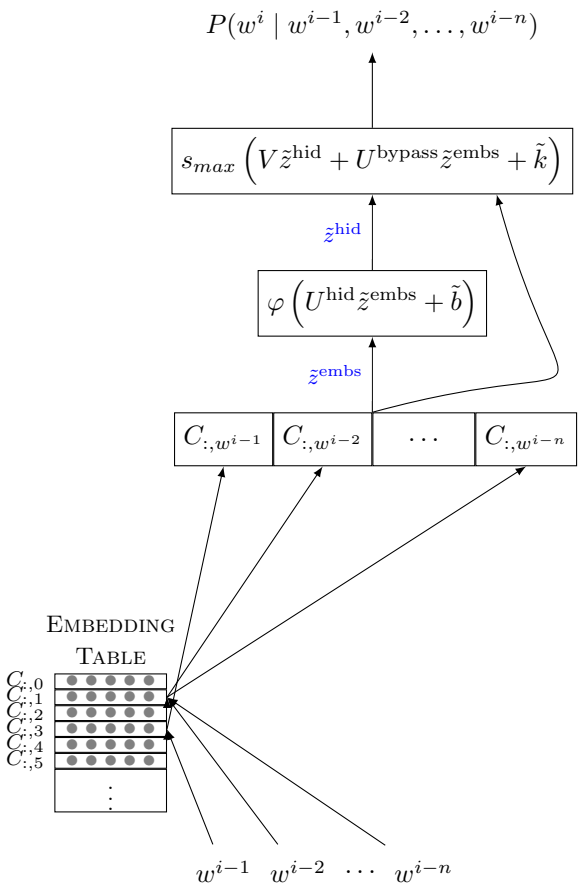
The term $R(W^i{=}w^i) = \exp\left(\tilde{k}_{w^i}\right)$ is linked to the unigram word probabilities: $P(Y = y)$. If $\mathbb{E}(R(Z \mid W_i) = 1$ then the optimal value for $\tilde{k}$ would be given by the log unigram probabilities: $k_{w^i} = \log P(W^i{=}w^i)$. This condition is equivalent to 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

NPLM 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. **NPLM** 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

[1] no pun intended

Figure 2.4: Neural Probabilistic Language Model



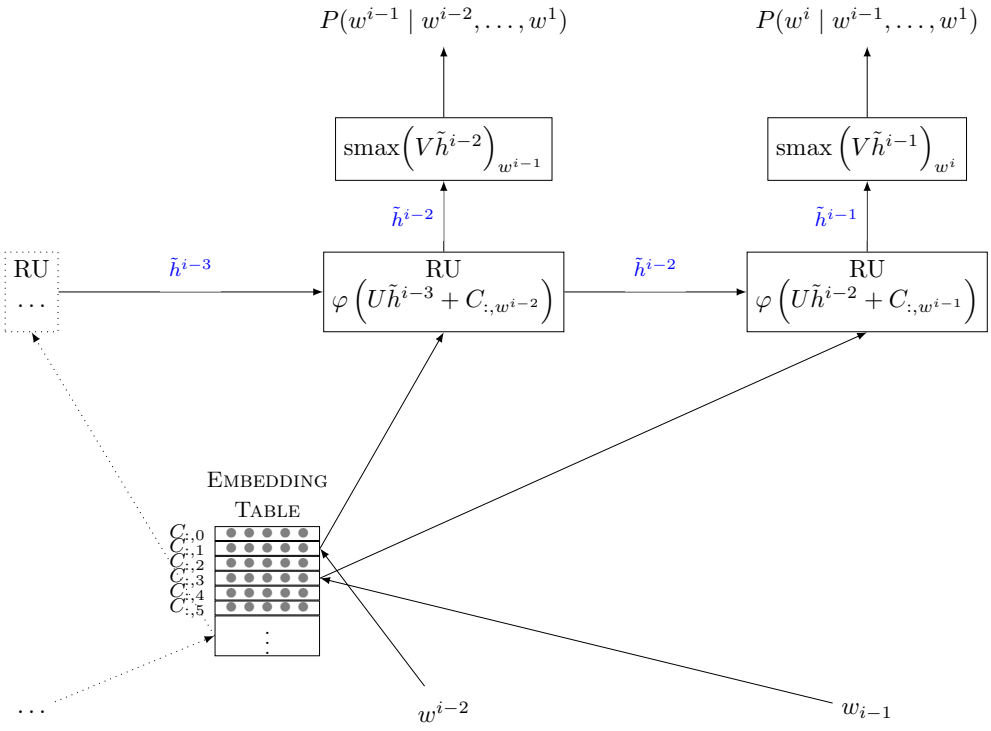
model is described by:

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

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 **schwenk2004efficient**. 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.

Figure 2.5: RNN Language Model. The RU equation shown is the basic RU used in **mikolov2010recurrent**. It can be substituted for a LSTM RU or an GRU as was done in **sundermeyer2012lstm**; **jozefowicz2015empirical**, with appropriate changes.



2.1.2 RNN Language Models

In **mikolov2010recurrent** an RNN is used for language modelling, as shown in Figure 2.5. Using the terminology of Chapter 2 of (**NRoNL**), 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) 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 **sundermeyer2012lstm** and **jozefowicz2015empirical**, 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.

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 **mikolov2013efficient**. 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; 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:

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

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 **mikolov2013efficient**. In this model, the central word is used to predict the words in the context.

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

Figure 2.6: CBOW Language Model

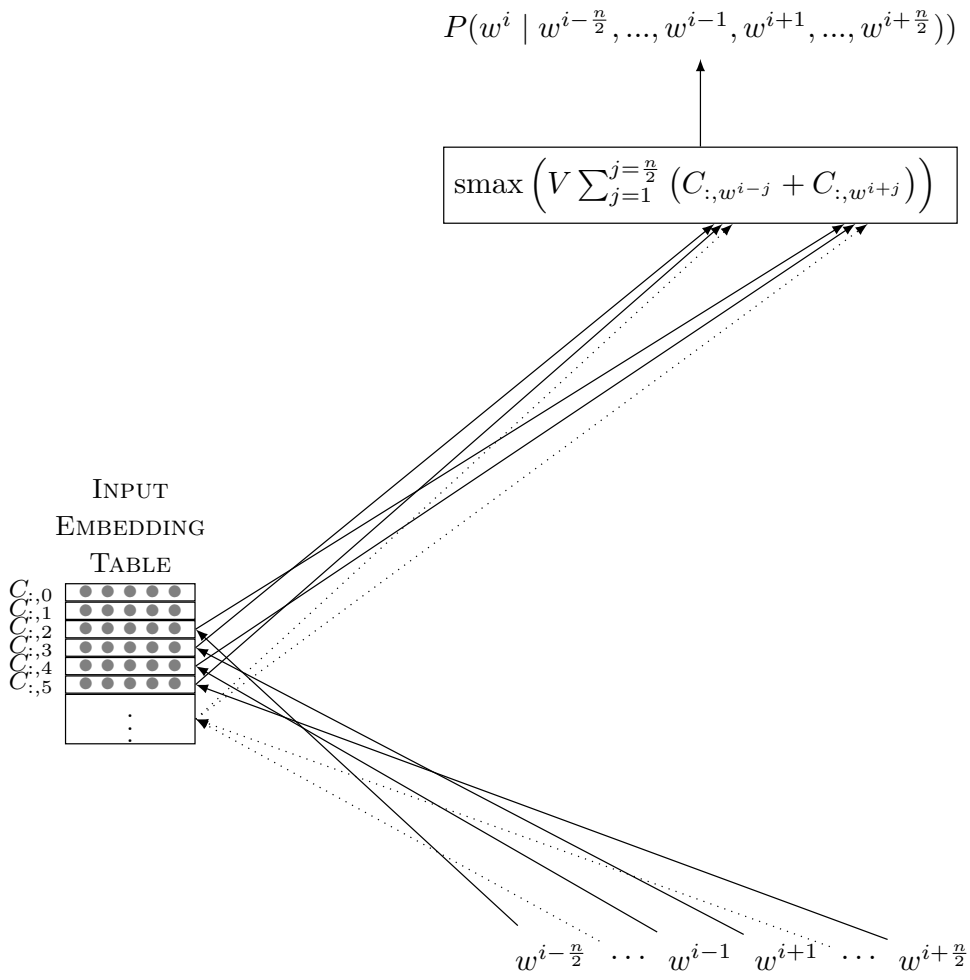


Figure 2.7: Skip-gram language Language Model. Note that the probability $P(w^j \mid w^i)$ is optimised during training for every w^j in a window around the central word w^i. Note that the final layer in this diagram is just a softmax layer, written in in output embedding form.

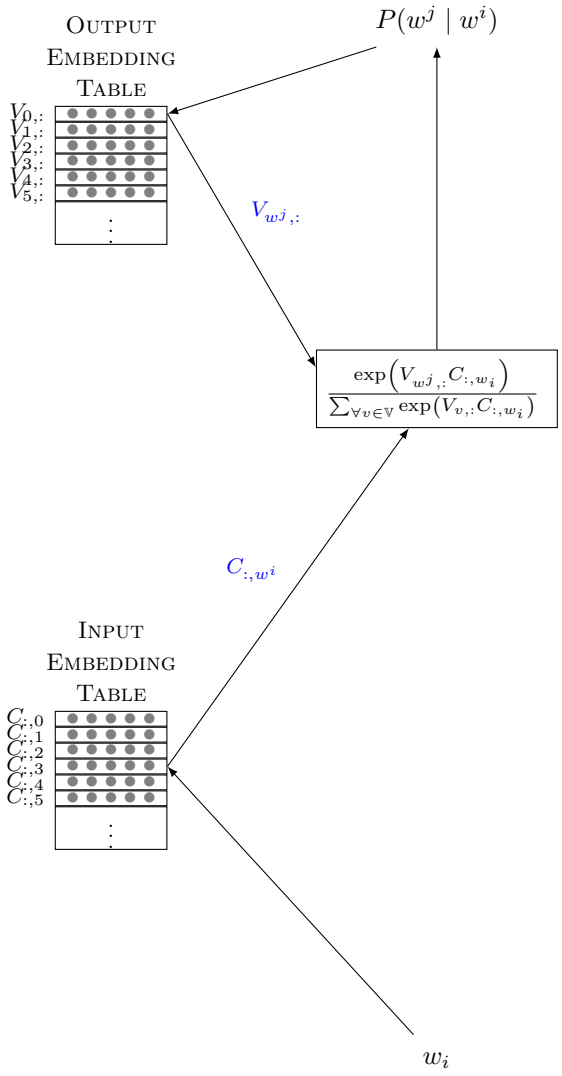
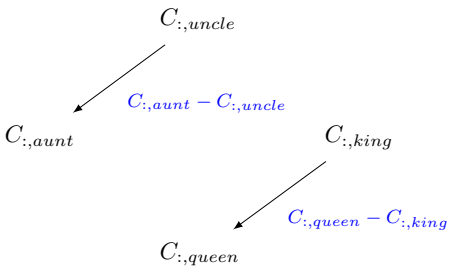


Figure 2.8: Example of analogy algebra



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:

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

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

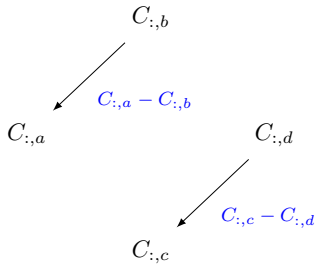
The key term here is the product $V_{w^j,:}\, C_{:,w^i}$. The remainder of Equation (2.16) is to normalise this into a probability. Maximising the probability $P(w^j \mid w^i)$ is equivalent to maximising the dot produce between $V_{w^j,:}$, the output embedding for w^j and $C_{:,w^i}$ the input embedding 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 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`

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



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 **gladkova2016analogy**, 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 **mikolov2013linguisticsubstructures** were relatively poor, but the surprising finding was that this worked at all. **mikolov2013efficient** 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 **pennington2014glove** 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 **pennington2014glove** 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.

pennington2014glove 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 **levy2015lsaisbasicallyskipgramswithexperimentstoprove** 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 **levy2014neural** and in **li2015wordemedingasEMF** with more direct mathematical equivalence to weighed co-occurrence matrix factorisation; Later, **cotterell2017SkipgramisEPCA** showed the equivalence to exponential principal component analysis (PCA). **wordvecispca** 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 **morin2005hierarchical**. Its recent use was popularised by **mikolov2013efficient**, 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 (**huffman1952method**), as was found to be effective by **mikolov2013efficient**. 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 **mikolov2013efficient**, 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.

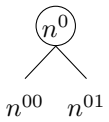
The worse case value for the entropy is $\log_2(|\mathbb{V}|)$. In-fact Huffman encoding is provably optimal in this way. As such this is the minimal number of operations required in the average case.

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 **mikolov2013efficient**). 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

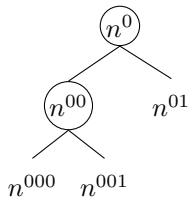
$$P(n^{0i} \mid \tilde{z}) = \sigma((-1)^i V_{0,:}\tilde{z}) \tag{2.29}$$

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 **mikolovSkip**.

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:

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

or to allow it to vary per node: as in the formulation of **mikolovSkip**. 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.

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

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 **mikolovSkip** 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 (**gutmann2012n**
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

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

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 **mikolovSkip**. 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.

mikolovSkip uses a distribution such that

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

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 **mikolovSkip**. 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 (**collobert2008unified**).

2.5.1 Using Word Embeddings as Features

turian2010word 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. **turian2010word** 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 (**gcca**), 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.

faruqui2014improving 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. **translating-unknown-wor** use this as part of a hybrid system to translate out of vocabulary words. **klein2015associating** use it to link word-embeddings with image embeddings.

dhillon2011multi 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 **dhillon2011multi**, though their final process more complex than described here. It is perhaps one of the more unusual 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 **shi2015learningbiligualcofactorisation**, amongst others. Similarly, on the sentence embedding side **zou2013bilingual**, and **socherDTRNN** 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 **Ruder17crosslingreview**.

Chapter 3

Word Sense Representations

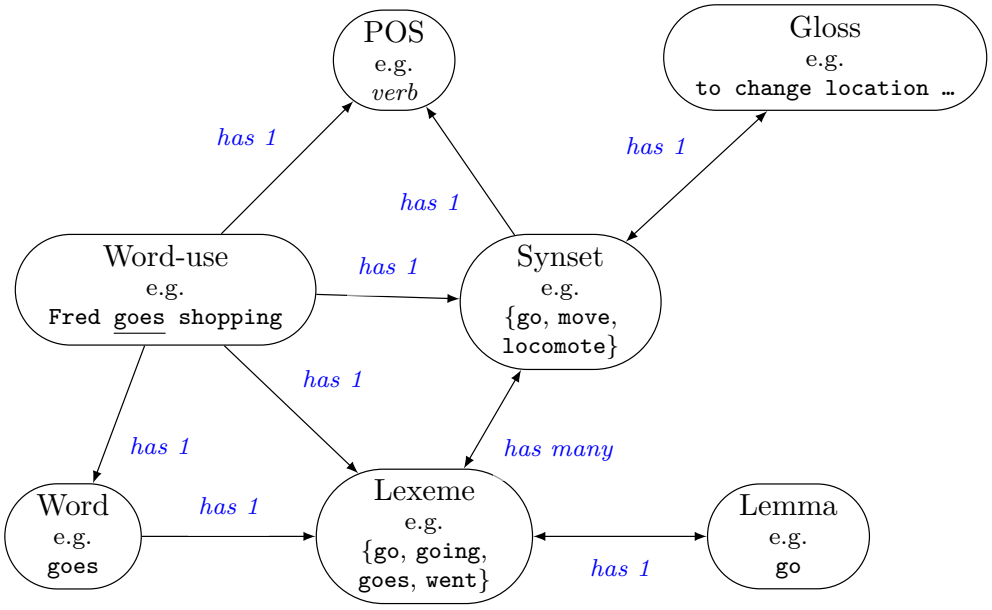
This chapter originally appeared as Chapter 4 of the book "Neural Representations of Natural Language", published by by Springer. The full book can be made available to the examiners on request. This chapter is not to be distributed to the general public and will not appear in the public online archival 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 (**miller1995wordnet**), and the multi-lingual BabelNet (**navigli2010babelnet**). 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 (**imagenet_cvpr09**)).

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 (**zipf1945meaning**). 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 (**zipf1949human**), 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 (**Kilgarriff2004**). 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 (**moro2015semeval**), 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. **iacobacci2015sensembed** use a CBOW language model (**mikolov2013efficient**) to do this. This does, however, run into the aforementioned problem, that there is relatively little training data that has been manually sense annotated. **iacobacci2015sensembed** use a third-party WSD tool, namely BabelFly (**Moro2014**), to annotate the corpus with senses. This allows for existing word representation techniques to be applied.

Chen2014 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

Chen2014 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.

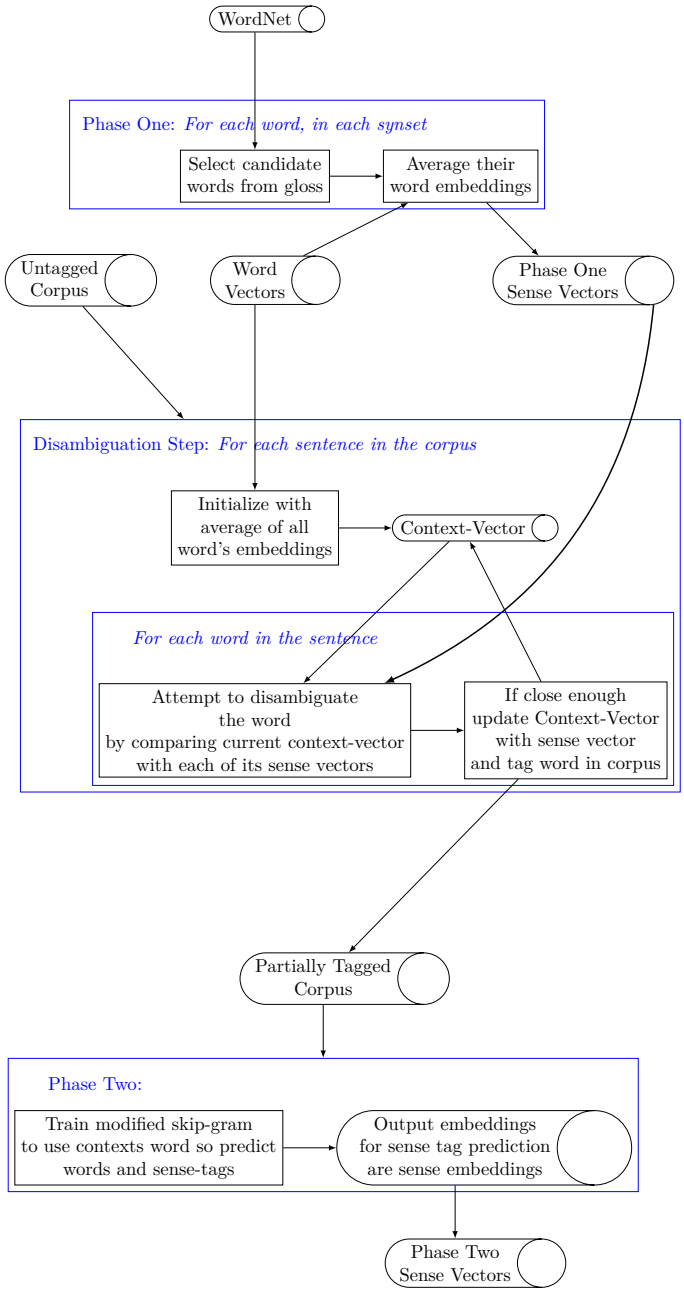
The **first phase** of this method is in essence a word-embedding-based WSD system. When assessed as such, they report that it only marginally exceeds the MFS baseline, though that is not at all unusual for WSD algorithms as discussed above.

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 (**mikolov2013efficient**). 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.

Figure 3.2: The process used by **Chen2014** to create word sense embeddings.



In **phase two** of training **Chen2014** 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 **schwenk2004efficient** 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 **Chen2014**, 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 (**WordSim353**) for context-less, or Stanford's Contextual Word Similarities (SCWS) for similarity with context information (**Huang2012**). 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.

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 **Schutze:1998wordsenseclustering**. That original work is not a neural word sense embedding, however the approach remains the same.

pantel2002WSI and **Reisinger2010** 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:

- **Schutze:1998wordsenseclustering** uses variations of inverse-document-frequency (idf) weighted bags of words, including applying dimensionality reduction to find a dense representation.
- **pantel2002WSI** use the mutual information vectors between words and their contexts.
- **Reisinger2010**, use td-idf or χ^2 weighted bag of words.
- **Huang2012** uses td-idf weighted averages of (their own) single sense word embeddings for all words in the context.
- **kaageback2015neural** 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 (**levy2014neural**)), 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.

- **Schutze:1998wordsenseclustering** uses a group average agglomerative clustering method.
- **pantel2002WSI** use a custom hierarchical clustering method.
- **Reisinger2010** use mixtures of von-Mises-Fisher distributions.
- **Huang2012** use spherical k-means.
- **kaageback2015neural** 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. **Schutze:1998wordsenseclustering** and **kaageback2015neural** use the centroids of their clusters. **Huang2012** 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 **Chen2014** 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. **neelakantan2015efficient** 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.

kaageback2015neural investigated using their weighting function (as discussed in Section 3.3.1) with the online clustering used by **neelakantan2015effici**
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.

tian2014probabilistic 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 **tian2014probabilistic**.

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 **tian2014probabilistic**, 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 (**zipf1945meaning**). 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.

AdaGrams 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 **tian2014probabilistic**. The final output of **AdaGrams** 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

This chapter originally appeared as Chapter 5 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.

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 (**dumais1988using**), probabilistic LSI (**hofmann2000learning**) and LDA (**blei2003latent**).

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. **Mitchell2008** 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 (**White2015BOWgen**).

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: **White2015SentVecMeaning**, **RitterPosition** and **rui2017mvrusemantic**. 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 (**Horvat2014**) and thus also from a SOWE (**White2016a**). 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 (**li2017neural**). 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. **li2017neural** also consider training n-gram embeddings as a bi-product of text classification tasks.

4.1.2 Paragraph Vector Models (Defunct)

le2014distributed 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. **le2014distributed** 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.

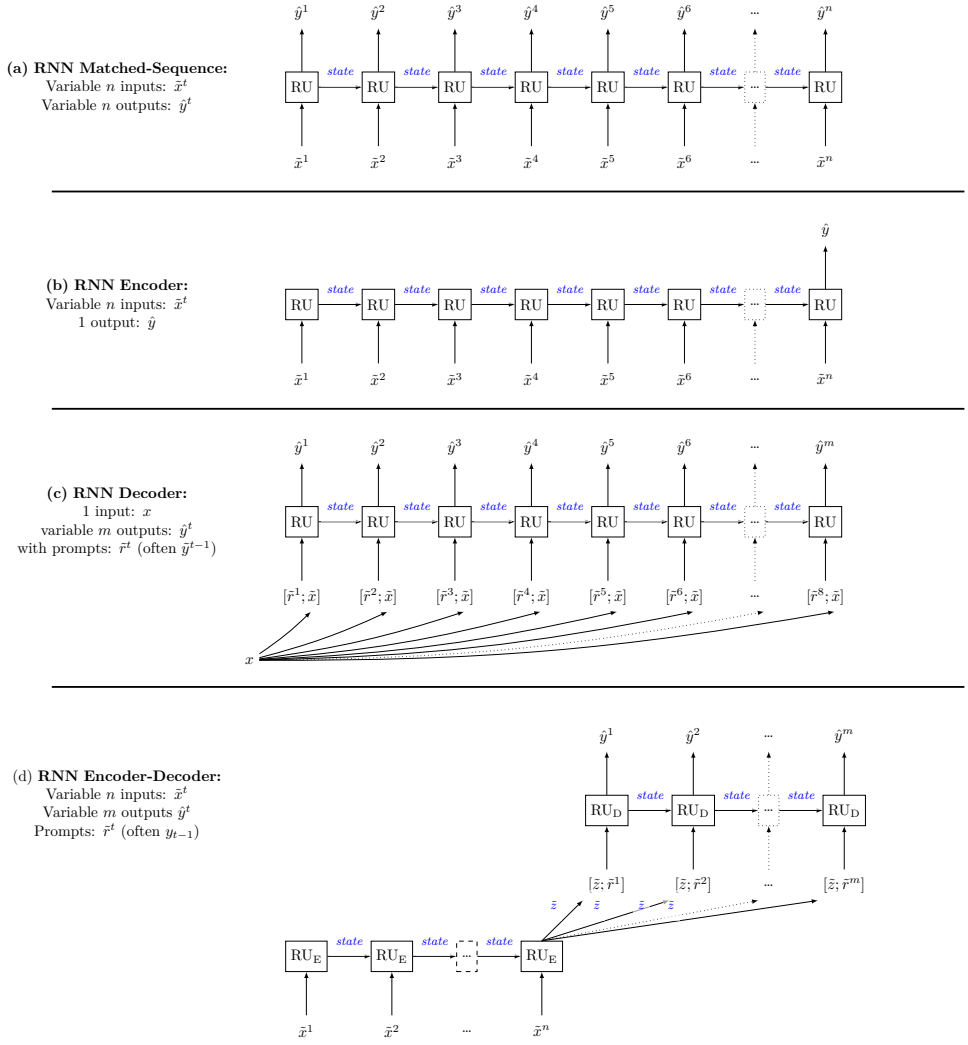
$$\begin{aligned} &P(w^i \mid d^k,\, w^{i-\frac{n}{2}}, ..., w^{i-1}, w^{i+1}, ..., w^{i+\frac{n}{2}}) \\ &\qquad = \text{smax}(WD_{:,d^k} + U \sum_{j=i+1}^{j=\frac{n}{2}} \left(C_{:,w^{i-j}} + C_{:,w^{i+j}}\right)) \end{aligned} \tag{4.1}$$

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.

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

The results of this work are now considered of limited validity. There were failures to reproduce the reported results in the original evaluations which were on sentiment analysis tasks. These were documented

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.



online by several users, including by the second author.[1] A follow up paper, **mesnil2014ensemble** found that reweighed bags of n-grams (**wang2012baselines**) out performed the paragraph vector models. Conversely, **lau2016doc2vecissues** 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 (**wang2012baselines**). On a different short text task, **White2015SentVecMeaning** 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 **NRoNL**. Every RNN learns a representation of all its input and output in its state. We can use RNN

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

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-EtAl:2014:EMNLP2014**, 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

Bowman2015SmoothGeneration 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 (**2014VAE**) 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.

Bowman2015SmoothGeneration 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-EtAl:2014:EMNLP2014**, where the *code* was concatenated to the input at every timestep of the decoder. **Bowman2015SmoothGeneration** investigated such a configuration, and found that it did not yield an improvement in performance.

4.2.2 Skip-thought

DBLP:journals/corr/KirosZSZTUF15 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 **NRoNL**, 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 **Bowman2015SmoothGeneration**. 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.

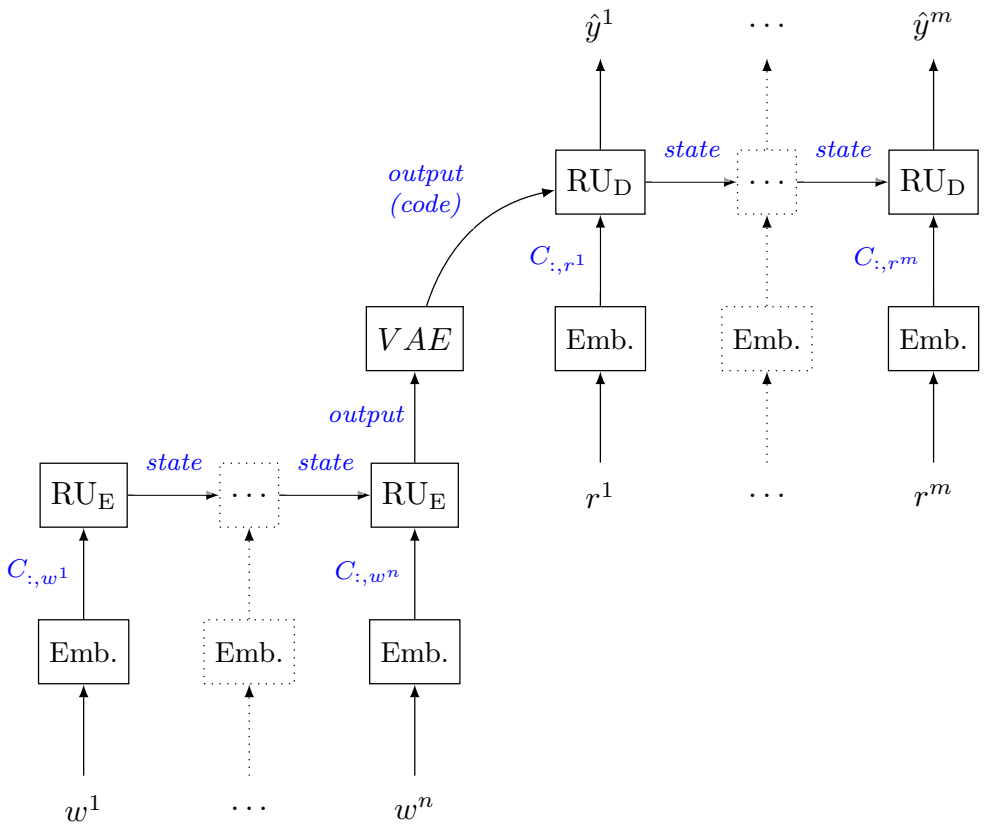
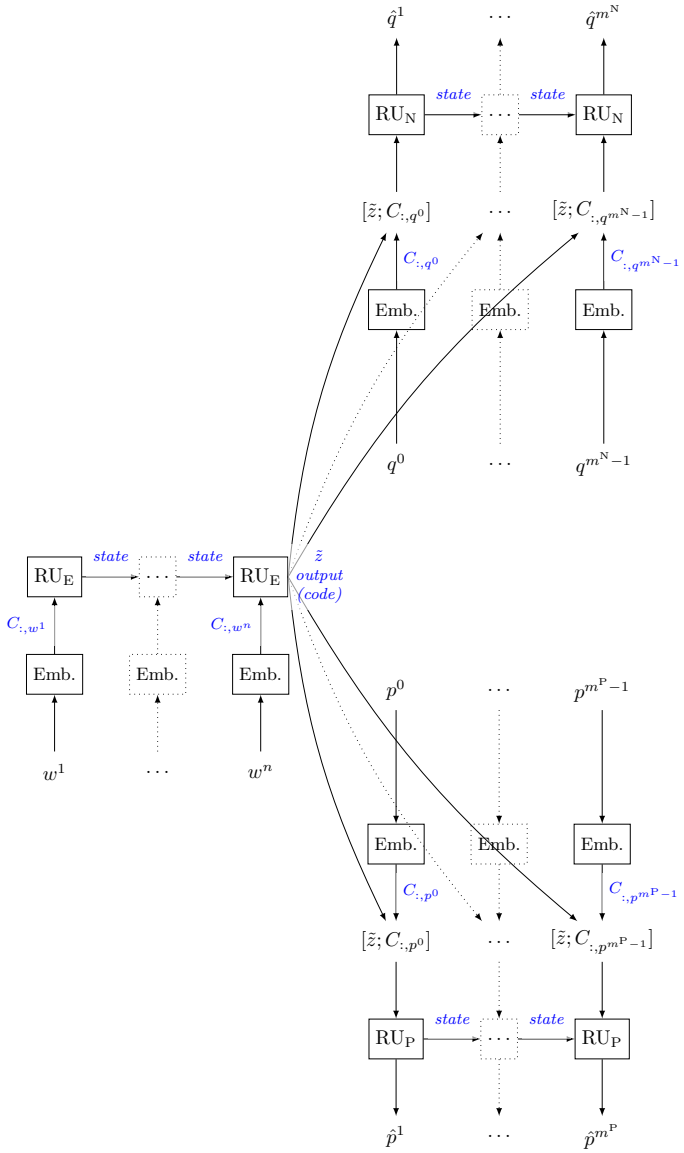


Figure 4.3: The skip-thought model (**DBLP:journals/corr/KirosZSZTUF15**). Note that for the next and previous sentences respectively the outputs are $\hat{q}^i$ and $\hat{p}^i$, and the prompts are q^{i-1} and p^{i-1}. As there is no intent to use the decoders after training, there is no need to worry 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 **socher2014recursive**. It builds on the work of **goller1996BPstructure** and **Pollack199077**, 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.

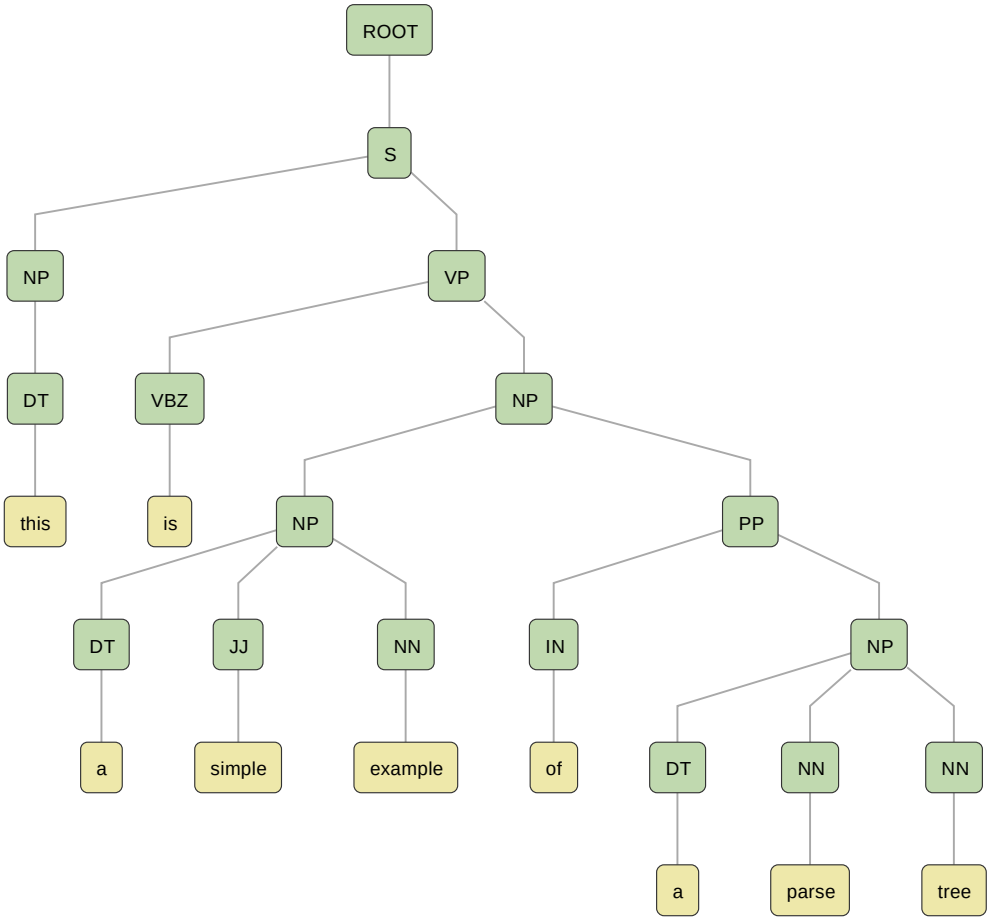
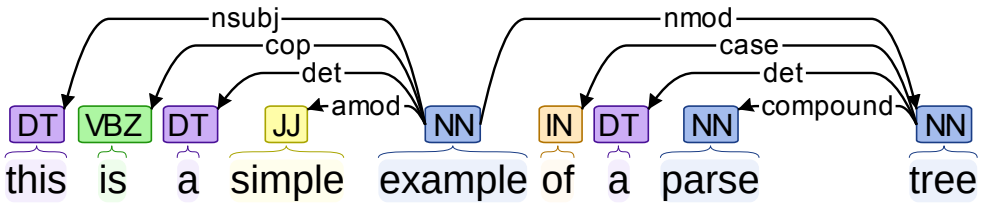


Figure 4.5: A dependency parse tree for the sentence `This is a simple example of a parse tree`, This flattened view may be misleading. `example` is at the peak of the tree, with direct children being: `this`,`is`,`a`,`simple`, and `tree`. `tree` has direct children being: `of`,`a`, and `parse`.



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 **socher2010PhraseEmbedding**, **SocherEtAl2011:RAE**, **SocherEtAl2011:PoolRAE**, **Socher2011ParsingPhrases** and **zhang2014BRAE**. 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 **socherDTRNN**, **iyyer2014neural**, and **iyyer2014generating**. 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 (**socher2010Phrasel**
and for dependency tree structured networks (**stenetorp2013transition**)
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 **socher2010PhraseEmbe**
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. **socher2010PhraseEmbedding** 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. **stenetorp2013transition** 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 **stenetorp2013transition** 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. **bowman2016fast**, 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 **NRoNL**, 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 **SocherEtAl2011:PoolRAE** and **iyyer2014generating** 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 **SocherEtAl2011:RAE** and **zhang2014BRAE** 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 **iyyer2014generating**, 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 **SocherEtAl2011:RAE** this was for sentiment analysis.
- In **SocherEtAl2011:PoolRAE** this was for paraphrase detection.
- In **zhang2014BRAE** 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

SocherMVRNN 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 **rui2017mvrusemantic**. 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 **NRoNL**, 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.

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

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

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

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

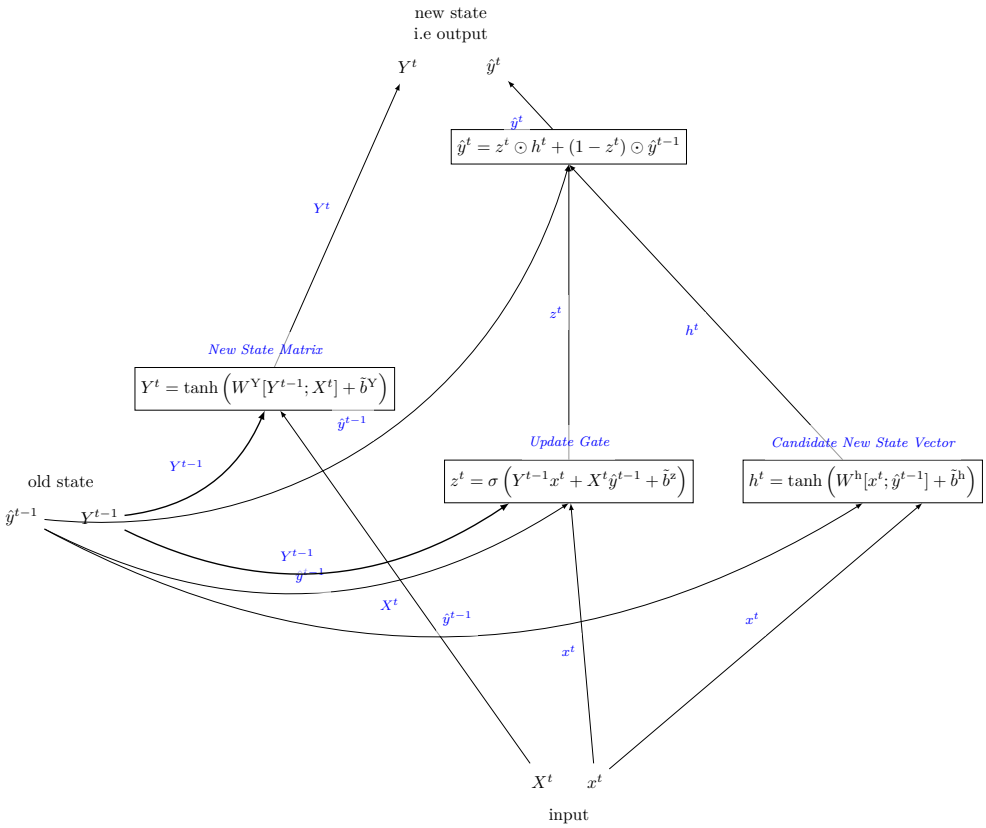
The matrices W^{h}, W^{Y} and the biases $\tilde{b}^{\mathrm{h}}$, $\tilde{b}^{\mathrm{z}}$, $\tilde{b}^{\mathrm{Y}}$ are shared across all time steps/compositions. W^{Y} controls how the next state-composition Y^t matrix is generated from its previous value and the input composition matrix, X^t; W^{h} 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 **SocherMVRNN** is applied in the gate z^t. The method for calculating h^t is similar to that of a normal RU.

The work of **rui2017mvrusemantic**, 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 (**2017arXiv170909360W**). 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. **RitterPosition** 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. **RitterPosition** 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 (**White2016a**), 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

How Well Sentence Embeddings Capture Meaning

This paper was presented at the 20th Australasian Document Computing Symposium, in 2015.

Abstract

Several approaches for embedding a sentence into a vector space have been developed. However, it is unclear to what extent the sentence's position in the vector space reflects its semantic meaning, rather than other factors such as syntactic structure. Depending on the model used for the embeddings this will vary – different models are suited for different down-stream applications. For applications such as machine translation and automated summarization, it is highly desirable to have semantic meaning encoded in the embedding. We consider this to be the quality of *semantic localization* for the model – how well the sentences' meanings coincides with their embedding's position in vector space. Currently the semantic localization is assessed indirectly through practical benchmarks for specific applications.

In this paper, we ground the semantic localization problem through a *semantic classification* task. The task is to classify sentences according to their meaning. A SVM with a linear kernel is used to perform the classification using the sentence vectors as its input. The sentences from subsets of two corpora, the Microsoft Research Paraphrase corpus and the Opinosis corpus, were partitioned according to their semantic equivalence. These partitions give the target classes for the classification task. Several existing models, including URAE, PV–DM and PV–DBOW, were assessed against a bag of words benchmark.

5.1 Introduction

Sentence embeddings are often referred to as semantic vector space representations **iyyer2014generating**. Embedding the meaning of a sentence into a vector space is expected to be very useful for natural language tasks. Vector representation of natural languages enables discourse analysis to take advantage of the array of tools available for computation in vector spaces. However, the embeddings of a sentence may encode a number of factors including semantic meaning, syntactic structure and topic. Since many of these embeddings are learned unsupervised on textual corpora using various models with different training objectives, it is not entirely clear the emphasis placed on each factor in the encoding. For

applications where encoding semantic meaning is particularly desirable, such as machine translation and automatic summarization, it is crucial to be able to assess how well the embeddings capture the sentence's semantics. In other words, for successful application to these areas it is required that the embeddings generated by the models correctly encode meaning such that sentences with the same meaning are co-located in the vector space, and sentences with differing meanings are further away. However, few current models are directly trained to optimize for this criteria.

Currently sentence embeddings are often generated as a byproduct of unsupervised, or semi-supervised, tasks. These tasks include: word prediction **le2014distributed**; recreation of input, as in the auto-encoders of **SocherEtAl2011:RAE**; **SocherEtAl2011:PoolRAE** and **iyyer2014generating**; and syntactic structural classification **SocherEtAl2013:CVG**; **socher2010PhraseEmbedding**. As a result the vector representations of the input sentences learned by these models are tuned towards the chosen optimization task. When employing the embeddings produced as features for other tasks, the information captured by the embeddings often proved to be very useful: e.g. approaching or exceeding previous state-of-the-art results, in sentiment analysis **SocherEtAl2011:RAE**; **SocherMVRNN**; **le2014distributed** and paraphrase detection **SocherEtAl2011:PoolRAE**. However these practical applications do not directly show how well meaning is captured by the embeddings.

This paper provides a new method to assess how well the models are capturing semantic information. A strict definition for the semantic equivalence of sentences is: that each sentence shall entail the other. Such mutually entailing sentences are called *paraphrases*. In this paper we propose to use paraphrases to assess how well the true semantic space aligns with the vector space the models embed into. It thus assesses whether projecting a sentence via the models in to the vector space preserves meaning.

The evaluation corpora were prepared by grouping paraphrases from the Microsoft Research Paraphrase (MSRP) **msrParapharaCorpus** and Opinosis **ganesan2010opinosis** corpora. A semantic classification task was defined which assesses if the model's embeddings could be used to correctly classify sentences as belonging to the paraphrase group with semantically equivalent sentences. Ensuring that sentences of common meaning, but differing form are located in vector space together, is a challenging task and shows a model's semantic encoding strength. This assessment, together with out recent work in the area, allows for a better understanding of how these models work, and suggest new directions for the development in this area.

The assessment proposed in this paper adds to the recent work on semantic evaluation methods, such as the work of Gershamn and Tenenbaum **gershmanphrase** and of Ritter et. al. **RitterPosition**. In particular, the real-world corpus based assessment in this paper is highly complementary to the structured artificial corpus based assessment of Ritter et. al. These methods are discussed in more detail in the next section.

The rest of the paper is organized into the following sections. Section 5.2 discusses the existing models being assessed, the traditional assessment

methods, and the aforementioned more recent semantic correctness based assessnements. Section 5.3 describes the processes by which the models are evaluated using our new method, and the parameters used in the evaluation. Section 5.4 continues into more details on the development of the evaluation corpora for the semantic classification evaluation task. Section 5.5 details the results from evaluating the models and discusses the implications for their semantic consistency. Section 5.6 closes the paper and suggests new directions for development.

5.2 Background

5.2.1 Models

Three well known sentence embedding methods are evaluated in this work. The compositional distributed model of the Unfoldering Recussive Autoencoder (URAE) by Socher et. al. **SocherEtAl2011:PoolRAE**; and the two word content predictive models, Distributed Memory (PV-DM) and Distributed Bag of Words by Le and Mikolov **le2014distributed**. In addition to these advanced sentence embedding models, a simple average of word embeddings, from Mikolov et. al. **mikolovSkip**, is also assessed. These models and their variant forms have been applied to a number of natural language processing tasks in the past, as detailed in the subsequent sections, but not to a real-sentence semantic classification task as described here.

Unfolding Recursive Auto-Encoder (URAE)

The Unfolding Recursive Autoencoder (URAE) **SocherEtAl2011:PoolRAE** is an autoencoder based method. It functions by recursively using a single layer feedforward neural-network to combine embedded representations, following the parse tree. Its optimization target is to be be able to reverse (unfold) the merges and produce the original sentence. The central folding layer – where the whole sentence is collapsed to a single embedding vector – is the sentence representation.

PV-DM

The Distributed Memory Paragraph Vectors (PV-DM) **le2014distributed** method is based on an extension of the Continuous Bag-of-Words word-embedding model **mikolov2013efficient**. It is trained using a sliding window of words to predict the next word. The softmax predictor network is fed a word-embedding for each word in the window, plus an additional sentence embedding vector which is reused for all words in the sentence – called the paragraph vector in **le2014distributed**. These input embeddings can be concatenated or averaged; in the results below they were concatenated. During training both word and sentence vectors are allowed to vary, in evaluation (i.e. inference), the word vectors are locked and the sentence vector is trained until convergence on the prediction task occurs.

PV-DBOW

Distributed Bag of Words Paragraph Vectors (PV-DBOW) **le2014distributed**, is based on the Skip-gram model for word-embeddings, also from **mikolov2013efficient**. In PV-DBOW a sentence vector is used as the sole input to a neural net. That network is tasked with predicting the words in the sentence. At each training iteration, the network is tasked to predict a number of words from the sentence, selected with a specified window size, using the sentence vector being trained as the input. As with PV-DM to infer embedding the rest of the network is locked, and only the sentence vector input allowed to vary, it is then trained to convergence.

Sum and Mean of Word Embeddings (SOWE and MOWE)

Taking the element-wise sum or mean of the word embeddings over all words in the sentence also produces a vector with the potential to encode meaning. Like traditional bag of words no order information is encoded, but the model can take into consideration word relations such as synonymity as encoded by the word vectors. The mean was used as baseline in **le2014distributed**. The sum of word embeddings first considered in **mikolovSkip** for short phrases, it was found to be an effective model for summarization in **KaagebExtractiveSummaristation**. The cosine distance, as is commonly used when comparing distances between embeddings, is invariant between sum and mean of word embeddings. Both sum and mean of word embeddings are computationally cheap models, particularly given pretrained word embeddings are available.

5.2.2 General Evaluation Methods

As discussed in the introduction, current methods of evaluating the quality of embedding are on direct practical applications designed downstream.

Evaluation on a Paraphrase Detection task takes the form of being presented with pairs of sentences and tasked with determining if the sentences are paraphrases or not. The MSRP Corpus, **msrParapharaCorpus** which we used in the semantic classification task, is intended for such use. This pairwise check is valuable, and does indicate to a certain extent if the embeddings are capturing meaning, or not. However, by considering groups of paraphrases, a deeper intuition can be gained on the arrangement of meaning within the vector space.

Sentiment Analysis is very commonly used task for evaluating embeddings. It was used both for the recursive autoencoder in **SocherEtAl2011:RAE** and for the paragraph vector models in **le2014distributed**. Sentiment Analysis is classifying a text as positive or negative, or assigning a score as in the Sentiment Treebank **RvNTN**. Determining the sentiment of a sentence is partially a semantic task, but it is lacking in several areas that would be required for meaning. For example, there is only an indirect requirement for the model to process the subject at all. Sentiment Analysis is a key task in natural language processing, but it is distinct from semantic meaning.

Document Classification is a classic natural language processing task. A particular case of this is topic categorization. Early work in the area goes back to **maron1961automatic** and **borko1963automatic**. Much more recently it has been used to assess the convolution neural networks of **DBLP:journals/corr/ZhangL15**, where the articles of several news corpora were classified into categories such as "Sports", "Business" and "Entertainment". A huge spectrum of different sentences are assigned to the same topic. It is thus too board and insufficiently specific to evaluate the consistency of meanings. Information retrieval can be seen as the inverse of the document classification task.

Information Retrieval is the task of identifying the documents which most match a query. Such document selection depends almost entirely on topic matching. Suitable results for information retrieval have no requirement to agree on meaning, though text with the same meaning are more likely to match the same queries.

The evaluation of semantic consistency requires a task which is fine grained, and preserving meaning. Document Classification and Information Retrieval are insufficiently fine-grained. Sentiment Analysis does not preserve meaning, only semantic orientation. Paraphrase Detection is directly relevant to evaluating semantic constancy, however it is a binary choice based on a pairwise comparison – a more spatial application is desirable for evaluating these vector spaces. Thus the current down-steam application tasks are not sufficient for assessing semantic consistency – more specialized methods are required.

5.2.3 Evaluations of Semantic Consistency

Semantic consistency for word embeddings is often measured using the analogy task. In an analogy the meta-relation: `A is to B as C is to D`. Mikolov et. al.**mikolov2013linguisticsubstructures** demonstrated that the word-embedding models are semantically consistent by showing that the semantic relations between words were reflected as a linear offset in the vector space. That is to say, for embeddings $\tilde{x}_a$, $\tilde{x}_b$, $\tilde{x}_c$, $\tilde{x}_d$ corresponding to words A, B, C and D, respectively; it was tested that if for a strong relationship matching between A/B and C/D, then the offset vector would be approximately equal: $\tilde{x}_b - \tilde{x}_a \approxeq \tilde{x}_d - \tilde{x}_c$. Rearranging this in word space gets the often quoted example of $\texttt{King} - \texttt{Man} + \texttt{Woman} \approxeq \texttt{Queen}$, As man is to woman, king is to queen. In the rating task as described by **jurgens2012semeval**, the goal is to rank such analogous word pairs based on the degree the relation matches. Thus to evaluate the word-embedding model using this task, it was a matter of sorting closeness of the corresponding offset vectors. Surprisingly strong results were found on this task**mikolov2013linguisticsubstructures**. It was thus demonstrated that word embeddings were not simply semantically consistent, but more so that this consistency was displayed as local linearity. This result gives confidence in the semantic quality of the word embeddings. However, this relationship analogy test cannot be performed for sentence embeddings.

Gershman et. al. **gershmanphrase**, compares the distances of modified sentences in vector space, to the semantic distances ascribed to them

by human raters. Like the analogy task for word vectors, this task requires ranking the targets based on the vector distance, however instead of rating on the strength of relationships it measures simply the similarities of the sentences to an original base sentence for each group. In that evaluation 30 simple base sentences of the form `A [adjective1] [noun1] [prepositional phrase] [adjective2] [noun2]` were modified to produce 4 difference derived sentences. The derived sentences were produced by swapping the nouns, swapping the adjectives, reversing the positional phrase (so `behind` becomes `in front of`), and a paraphrase by doing all of the aforementioned changes. Human raters were tasked with sorting the transformed sentences in similarity to the base sentence. This evaluation found that the embedding models considered did not agree with the semantic similarity rankings placed by humans. While the sentence embedding models performed poorly on the distance ranking measure, it is also worth considering how they perform on a meaning classification task.

A meaning classification task was recently proposed by Ritter et. al. **RitterPosition**, to classify sentences based on which spatial relationship was described. The task was to classify the sentence as describing: *Adhesion to Vertical Surface*, *Support by Horizontal Surface*, *Full Containment*, *Partial Containment*, or *Support from Above*. In this evaluation also, the sentences took a very structured form: `There is a [noun1] [on/in] the [noun2]`. These highly structured sentences take advantage of the disconnection between word content and the positional relationship described to form a task that must be solved by a compositional understanding combining the understanding of the words. *"The apple is on the refrigerator"* and *"The magnet is on the refrigerator"* belong to two separate spatial categories, even though the word content is very similar. Surprisingly, the simple model of adding word vectors outperformed compositional models such as the recursive autoencoder. The result does have some limitation due to the highly artificial nature of the sentences, and the restriction to categorizing into a small number of classes based only on the meaning in terms of positional relationship. To generalize this task, in this paper we consider real world sentences being classed into groups according to their full semantic meaning.

5.3 Methodology

To evaluate how well a model's vectors capture the meaning of a sentence, a semantic classification task was defined. The task is to classify sentences into classes where each shares the same meaning. Each class is thus defined as a paraphrase groups. This is a far finer-grained task than topic classification. It is a multiclass classification problem, rather than the binary decision problem of paraphrase detection. Such multiclass classification requires the paraphrase groups to be projected into compact and distinct groups in the vector space. A model which produces such embeddings which are thus easily classifiable according to their meaning can been thus seen to have good semantic localization.

This semantic classification does not have direct practical application – it is rare that the need will be to quantify sentences into groups with

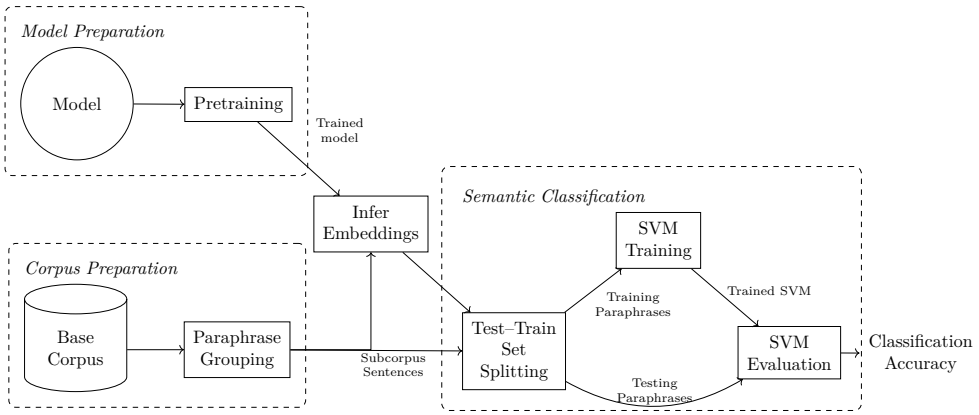


Figure 5.1: Process Diagram for the Evaluation of Semantic Consistency via our method

the same prior known meaning. Rather it serves as a measure to assess the models general suitability for other tasks requiring a model with consistency between meaning and embedding.

To evaluate the success at the task three main processes are involved, as shown in Figure 5.1: Corpus Preparation, Model Preparation, and the Semantic Classification task itself.

5.3.1 Corpus Preparation

The construction of each of the corpora is detailed more fully in the next section. In brief: Two corpora were constructed by selecting subsets of the Microsoft Research Paraphrase (MSRP) **msrParapharaCorpus** and of the Opinosis **ganesan2010opinosis** corpora. The corpora were partitioned into groups of paraphrases – sentences with the same meaning. Any paraphrase groups with less than three sentences were discarded. The paraphrase grouping was carried out manually for Opinosis, and automatically for the MSRP corpus using the existing paraphrase pairings. The paraphrase groups divide the total semantic space of the corpora into discrete classes, where each class contains sentences sharing the same meaning.

It is by comparing the ability of the models to produce embeddings which can be classified back into these classes, that we can compare the real semantic space partitions to their corresponding vector embedding space regions.

5.3.2 Model Preparation and Inferring Vectors

Prior to application to semantic classification, as with any task the models had to be pretrained. Here we use the term *pretraining* to differentiate the model training from the classifier training. The pretraining is not done using the evaluation corpora as they are both very small. Instead other data are used, and the inference/evaluation procedure given for each method was then used to produce the vectors for each sentence. The model parameters used are detailed below.

Unfolding Recursive Auto-Encoder (URAE)

In this evaluation we make use of the pretrained network that Socher et. al. have graciously made available[1], full information is available in the paper**SocherEtAl2011:PoolRAE**. It is initialized on the unsupervised Collobert and Weston word embeddings**collobert2008unified**, and training on a subset of 150,000 sentences from the gigaword corpus. It produces embeddings with 200 dimensions. This pretrained model when used with dynamic pooling and other word based features performed very well on the MSRP corpus paraphrase detection. However in the evaluation below the dynamic pooling techniques are not used as they are only directly suitable for enhancing pairwise comparisons between sentences.

Paragraph Vector Methods (PV-DM and PV-DBOW)

Both PV-DM and PV-DBOW, were evaluated using the GenSim implementation **rehurek_lrec** from the current *develop* branch[2]. Both were trained on approximately 1.2 million sentences from randomly selected Wikipedia articles, and the window size was set to 8 words, and the vectors were of 300 dimensions.

Sum and Mean of Word Embeddings (SOWE and MOWE)

The word embeddings used for MOWE were taken from the Google News pretrained model[3] based on the method described in **mikolovSkip**. This has been trained on 100 million sentences from Google News. A small portion of the evaluation corpus did not have embeddings in the Google News model. These tokens were largely numerals, punctuation symbols, proper nouns and unusual spellings, as well as the stop-words: “and”, “a” and “of”. These words were simply skipped. The resulting embeddings have 300 dimensions, like the word embeddings they were based on.

Bag of Words (BOW and PCA BOW)

A bag of words (BOW) model is also presented as a baseline. There is a dimension in each vector embedding for the count of each token, including punctuation, in the sentence. In the Opinosis and MSRP subcorpora there were a total of 1,085 and 2,976 unique tokens respectively, leading to BOW embeddings of corresponding dimensionality. As it is a distributional rather than distributed representation, the BOW model does not need any pretraining step. For comparison to the lower dimensional models Principle Component Analysis (PCA) was applied to the BOW embeddings to produce an additional baseline set of embeddings of 300 dimensions – in line with PV-DM, PV-DBOW, SOWE, and MOWE models. It does not quite follow the steps shown in Figure 5.1, as the PCA pretraining step is performed on the training embeddings

[1] http://www.socher.org/index.php/Main/DynamicPoolingAndUnfoldingRecursiveAutoencodersForParaphraseDetection

[2] https://github.com/piskvorky/gensim/tree/develop/

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

only during the SVM classification process, and it is used to infer the PCA BOW embeddings during the testing step. This avoids unfair information transfer where the PCA would otherwise be about to choose representations optimized for the whole set, including the test data. It was found that when the PCA model was allowed to cheat in this way it performed a few percentage points better. The bag of words models do not have any outside knowledge.

5.3.3 Semantic Classification

The core of this evaluation procedure is in the semantic classification step. A support vector machine (SVM), with a linear kernel, and class weighting was applied to the task of predicting which paraphrase group each sentence belongs to. Classification was verified using 3-fold cross-validation across different splits of the testing/training data, the average results are shown in this section. The splits were in proportion to the class size. For the smallest groups this means there were two training cases and one test case to classify.

In this paper, only a linear kernel was used, because a more powerful kernel such as RBF may be able to compensate for irregularities in the vector space, which makes model comparison more difficult. Scikit-learn **scikit-learn** was used to orchestrate the cross-validation and to interface with the LibLinear SVM implementation **LIBLIBEAR**. As the linear SVM's classification success depends on how linearly separable the input data is, thus this assessed the quality of the localization of the paraphrase groupings embeddings.

5.4 Corpus Construction

5.4.1 Microsoft Research Paraphrased Grouped Subcorpus

The MSRP corpus is a very well established data set for the paraphrase detection task **msrParapharaCorpus**. Sentences are presented as pairs which are either paraphrases, or not. A significant number of paraphrases appear in multiple different pairings. Using this information, groups of paraphrases can be formed.

The corpus was partitioned according to sentence meaning by taking the symmetric and transitive closures the set of paraphrase pairs. For example if sentences A, B, C and D were present in the original corpus as paraphrase pairs: A, B, D, A and B, C then the paraphrase group $\{A, B, C, D\}$ is found. Any paraphrase groups containing less than 3 phrases were discarded. The resulting sub-corpus has the breakdown as shown in Figure 5.2.

5.4.2 Opinosis Paraphrase Grouped Subcorpus

The Opinosis Corpus**ganesan2010opinosis** was used as secondary source of original real-world text. It is sourced from several online review sites:

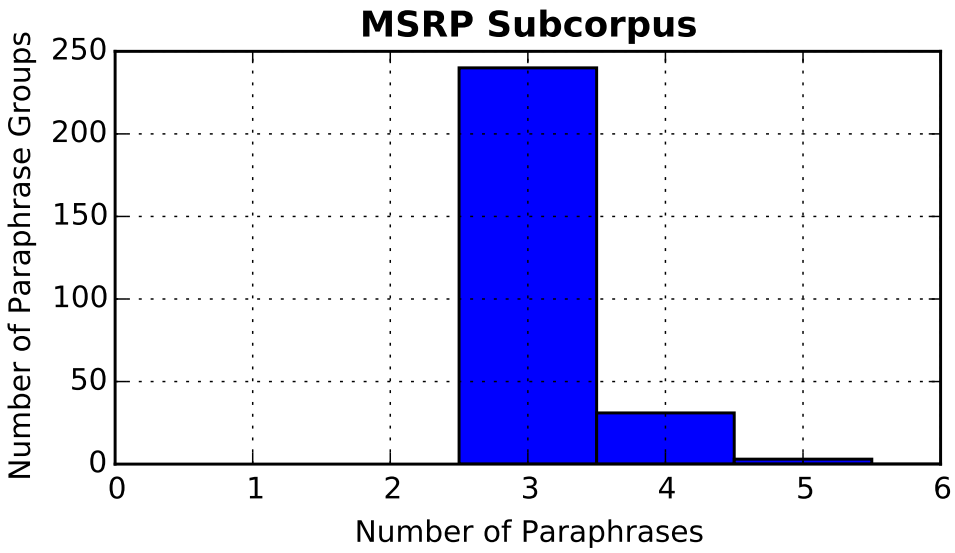


Figure 5.2: Break down of how many paraphrases groups are present in the MSRP subcorpus of which sizes.It contains a total of 859 unique sentences, broken up into 273 paraphrase groups.

Tripadvisor, Edmunds.com, and Amazon.com, and contains single sentence statements about hotels, cars and electronics. The advantage of this as a source for texts is that comments on the quality of services and products tend to be along similar lines. The review sentences are syntactically simpler than sentences from a news-wire corpus, and also contain less named entities. However, as they are from more casual communications, the adherence to grammar and spelling may be less formal.

Paraphrases were identified using the standard criterion: bidirectional entailment. For a paraphrase group $\mathcal{S}$ of sentences: $\forall s_1, s_2 \in \mathcal{S}, \quad s_1 \vDash s_2 \quad \wedge s_2 \vDash s_1$, every sentence in the group entails the every other sentence in the group. A stricter interpretation of bidirectional entailment was used, as compared to the "mostly bidirectional entailment" used in the MSRP corpus. The grouping was carried out manually. Where it was unclear as to the group a particular phrase should belong to it was left out of the corpus entirely. The general guidelines were as follows.

- Tense, Transitional Phrases, and Discourse and Pragmatic Markers were ignored.
- Statement intensity was coarsely quantized.
- Approximately equal quantitative and qualitative values were treated as synonymous.
- Sentences with entities mentioned explicitly were grouped separately from similar statements where they were implied.
- Sentences with additional information were grouped separately from those without that information.

The final point is the most significant change from the practices apparent in the construction of the MSRP corpus. Sentences with differing or additional information were classified as non-paraphrases. This requirement comes from the definition of bidirectional entailment. For example, *"The staff were friendly and polite."*, *"The staff were polite."* and *"The staff*

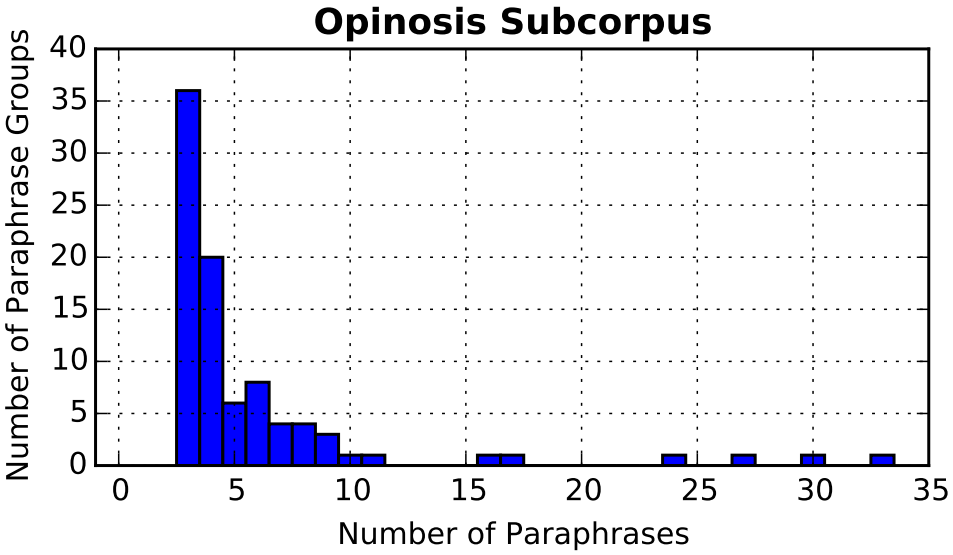


Figure 5.3: Break down of how many paraphrases groups are present in the Opinosis subcorpus of which sizes. It contains a total of 521 unique sentences, broken up into 89 paraphrase groups.

were friendly." are in three separate paraphrase groups. The creators of the MSRP corpus, however, note "...the majority of the equivalent pairs in this dataset exhibit 'mostly bidirectional entailments', with one sentence containing information 'that differs' from or is not contained in the other." **msrParapharaCorpus**. While this does lead to more varied paraphrases; it strays from the strict linguistic definition of a paraphrase, which complicates the evaluation of the semantic space attempted here. This stricter adherence to bidirectional entailment resulted in finer separation of groups, which makes this a more challenging corpus.

After the corpus had been broken into paraphrase groups some simple post-processing was done. Several artifacts present in the original corpus were removed, such as substituting the ampersand symbol for `&amp`. Any paraphrase groups containing identical sentences were merged, and duplicates removed. Finally, any group with less than three phrases was discarded. With this complete the breakdown is as in Figure 5.3.

Further information on the construction of the corpora in this section, and download links are available online.[4]

5.5 Results and Discussion

5.5.1 Classification Results and Discussion

The results of performing the evaluation method described in Section 5.3 are shown in Table 5.1.

While the relative performance of the models is similar between the corpora, the absolute performance differs. On the absolute scale, all the models perform much better on the MSRP subcorpus than on the Opinosis subcorpus. This can be attributed to the significantly more

[4] http://white.ucc.asn.au/resources/paraphrase_grouped_corpora/

	MSRP Subcorpus	Opinosis Subcorpus
PV-DM	78.00%	38.26%
PV-DBOW	89.93%	32.19%
URAE	51.14%	20.86%
MOWE	97.91%	**69.30%**
SOWE	98.02%	68.75%
BOW	**98.37%**	65.23%
PCA BOW	97.96%	54.43%

Table 5.1: The semantic classification accuracy of the various models across the two evaluation corpora.

distinct classes in the MSRP subcorpus. The Opinosis subcorpus draws a finer line between sentences with similar meanings. As discussed earlier, for example there is a paraphrase group for *"The staff were polite."*, another for *"The staff were friendly."*, and a third for *"The staff were friendly and polite."*. Under the guidelines used for paraphrases in MSRP, these would all have been considered the same group. Secondly, there is a much wider range of topics in the MSRP. Thus the paraphrase groups with different meanings in MSRP corpus are also more likely to have different topic entirely than those from Opinosis. Thus the the ground truth of the semantics separability of phrases from the MSRP corpus is higher than for Opinosis, making the semantic classification of the Opinosis subcorpus is a more challenging task.

The URAE model performs the worst of all models evaluated. In **KaagebExtractiveSummaristation** is was suggested that the URAE's poor performance at summarizing the Opinosis corpus could potentially be attributed to the less formally structured product reviews – the URAE being a highly structured compositional model. However, here it also performed poorly on the MSRP – which it was created for **SocherEtAl2011:PoolRAE**. The exact same model from **SocherEtAl2011:PoolRAE** was used here – though this did put it at a dimensional disadvantage over the other models having 200 dimensions to the other's 300. The key difference from **SocherEtAl2011:PoolRAE**, beyond the changing to a multiclass classification problem, was the lack of the complementary word-level features as used in the dynamic pooling layer. This suggests the model could benefit from such world level features – as the very strong performance of the word-based model indicates.

The word based models, MOWE, SOWE, BOW and PCA BOW, performed very well. This suggests that word choice is a very significant factor in determining meaning; so much so that the models which can make use of word order information, URAE and PV-DM, were significantly outperformed by methods which made more direct use of the word content.

The very high performance of the BOW maybe attributed to its very high dimensionality, though the MOWE and SOWE performed similarly. The PCA step can be considered as being similar to choosing an optimal set of words to keep so as to maximum variability in the bag of words. It loses little performance, even though decreasing vector size by an order of magnitude – particularly on the easier MSRP dataset.

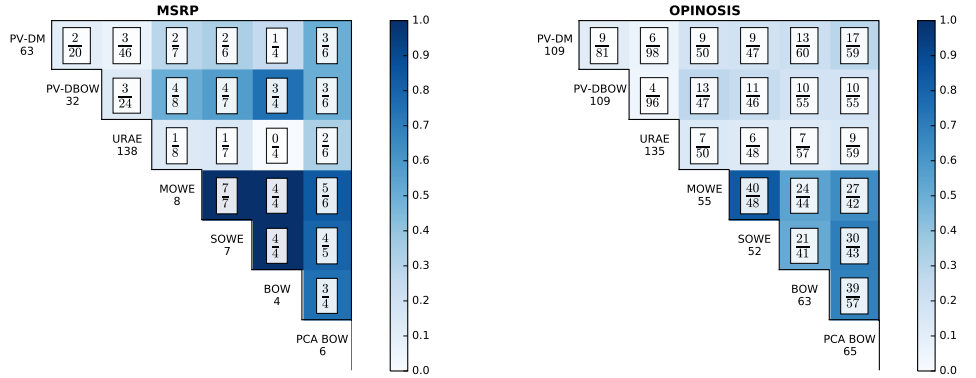


Figure 5.4: The misclassification agreement between each of the models for the MSRP (left) and Opinosis (right) subcorpora. Below each model name is the total mistakes made. The denominator of each fraction is the number of test cases incorrectly classified by both models. The numerator is the portion of those misclassifications which were classified in the same (incorrect) way by both models. The shading is in-proportion to that fraction.

5.5.2 Model Agreement

The misclassifications of the models can be compared. By selecting one of the test/train folds from the classification task above, and comparing the predicted classifications for each test-set sentence, the similarities of the models were assessed. The heatmaps in Figure 5.4 show the agreement in errors. Here misclassification agreement is given as an approximation to $P(m_1(x) = m_2(x) \,|\, m_1(x) \neq y \wedge m_2(x) \neq y)$, for a randomly selected sentence x, with ground truth classification y, where the models m_1 and m_2 are used to produce classifications. Only considering the cases where both models were incorrect, rather than simple agreement, avoids the analysis being entirely dominated by the agreement of the models with the ground truth.

The word based models showed significant agreement. Unsurprisingly MOWE and SOWE have almost complete agreement in both evaluations. The other models showed less agreement – while they got many of the same cases wrong the models produced different misclassifications. This overall suggests that the various full sentence models are producing substantially dissimilar maps from meaning to vector space. Thus it seems reasonable that using a ensemble approach between multiple sentence models and one word-based model would produce strong results. Yin and Schütze **Yin2015** found this successful when combining different word embedding models.

5.5.3 Limitations

This evaluation has some limitations. As with all such empirical evaluations of machine learning models, a more optimal choice of hyperparameters and training data will have an impact on the performance. In particular, if the model training was on the evaluation data the models would be expected to be better able to position their embedding. This was however unfeasible due to the small sizes of the datasets used for evaluation, and would not reflect real word application of the models to

data not prior seen. Beyond the limitation of the use of the datasets is their contents.

The paraphrase groups were not selected to be independent of the word content overlap – they were simply collected on commonality of meaning from real world sourced corpora. This is a distinct contrast to the the work of Ritter et. al.**RitterPosition** discussed in Section 5.2.3 where the classes were chosen to not have meaningful word overlap. However our work is complementary to theirs, and our findings are well aligned. The key difference in performance is the magnitude of the performance of the sum of word embeddings (comparable to the mean of word embeddings evaluated here). In **RitterPosition** the word embedding model performed similarly to the best of the more complex models. In the results presented above we find that the word embedding based model performs significantly beyond the more complex models. This can be attributed to the word overlap in the paraphrase groups – in real-world speech people trying to say the same thing do in-fact use the same words very often.

5.6 Conclusion

A method was presented, to evaluate the semantic localization of sentence embedding models. Semantically equivalent sentences are those which exhibit bidirectional entailment – they each imply the truth of the other. Paraphrases are semantically equivalent. The evaluation method is a semantic classification task – to classify sentences as belonging to a paraphrase group of semantically equivalent sentences. The datasets used were derived from subsets of existing sources, the MRSP and the Opinosis corpora. The relative performance of various models was consistent across the two tasks, though differed on an absolute scale.

The word embedding and bag of word models performed best, followed by the paragraph vector models, with the URAE trailing in both tests. The strong performance of the sum and mean of word embeddings (SOWE and MOWE) compared to the more advanced models aligned with the results of Ritter et. al.**RitterPosition**. The difference in performance presented here for real-word sentences, were more marked than for the synthetic sentence used by Ritter et. al. This may be attributed to real-world sentences often having meaning overlap correspondent to word overlap – as seen also in the very strong performance of bag of words. Combining the result of this work with those of Ritter et. al., it can be concluded that summing word vector representations is a practical and surprisingly effective method for encoding the meaning of a sentence.

Chapter 6

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 (RNN) and convolution (CNN), to estimate colors from sequences of terms. We consider both point as well as 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". Surprisingly, the sum of word embeddings generally performs the best on almost all evaluations.

6.1 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 bushland. 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, indispensable for *executable semantic parsing*. 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 here is to address these linguistic phenomena around the short-phrase descriptions of a color, which can be represented in a color-space such as HSV (**smith1978color**). Issues of illumination and perceived color based on context are considered out of the scope.

6.1.1 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. Having said that, we do produce point estimate results for completeness, though we argue the true usefulness of such estimates is limited.

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 mean as a point estimate will select a point in the valley between the peaks, which is less likely. Similarly for an asymmetrical distribution, where the mean will be off to one side of the peak. Conversely, using the distribution mode, will select the highest (most likely) peaks, but will on average be more incorrect (as measured by the mean squared error). The correct trade-off, if a point estimate is required, is dependent 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 Section 6.6.1). While by definition the mean color point minimizes the squared error, it may not actually be a meaningful point estimate. Given these issues, producing a point estimate has only limited value and estimating a distribution is more general task. However we do consider the point estimation task, as

it allows contrast in assessing the input module (SOWE/CNN/RNN) of our proposed methods across the two different output modules (distribution/point estimation).

The 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/KawakamiDRS16**; 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. To our knowledge The generation of probability distributions in color-space based on sequences of terms making up the color name, has not been considered at all by any prior work. There has been several works on the reverse problem (**mcmahan2015bayesian**; **meomcmahanstone:color**; **2016arXiv160603821M**): the generation of a textual name for a color from a point in a color-space. From the set of work on the reverse problem there is a clear trend on data-driven approaches in recent years where more color names and observations are used.

6.1.2 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 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 connected **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, three different color-name embedding learning models are investigated for use in the **input module**: Sum Of Word Embeddings (SOWE), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). 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).

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 displays which were 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. To qualify our estimate of the distribution, 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.

We conclude that the 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 RNN performing significantly worse (see Section 6.6). 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 the models for their general use in short phrase understanding.

To the best of knowledge, this is the first of such investigation in term-wise mapping color names to values that can generate a visual representation, which bring the future of natural language controlled computer graphics generation and executable semantic parsing one step closer to reality.

6.2 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** which propose methods for the automatic mapping of colors to and from these small manually defined sets of colors. 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.

More recent works, including the work presented in this article, function with a much larger number of colors, larger vocabularies, and larger pools of respondents. In particular making uses of the large Munroe dataset **Munroe2010XKCDdataset**, as we do here. This allows a data driven approach towards the modelling.

mcmahan2015bayesian and **meomcmahanstone:color** present color naming methods. Their work primarily focuses on mapping from colors to to their exact names, the reverse of our task. 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.

2016arXiv160603821M 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 to the creation of a conditional language model. 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 **2016arXiv160603821** but functioning on a character, rather than a word level. The work by Kawakami et al. also includes a method for generating colors. However they only consider the generation of point estimated, 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 modelling 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 modelling: the character based modelling 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 is calculated, over a high dimensional color-space from **2016arXiv160603821M**, 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 point estimates of how colors change based on the use of comparative terms such as `lighter`. The network's input is word embedding for a comparative adjective, and a point in RGB color-space; the model outputs a point in predicted RGB space that is the modified version of the input color point according to the given adjective. For example mapping from green to a darker green is: $((164, 227, 77), \text{darker}) \mapsto (141, 190, 61)$. The color adjectives may have up to two words, to allow for expressions such as `more neon`. This is allowed by taking as the fixed sized input two embeddings – when only one input is required, the other is replaced by zero vector. Their training and evaluation is based on data sourced from

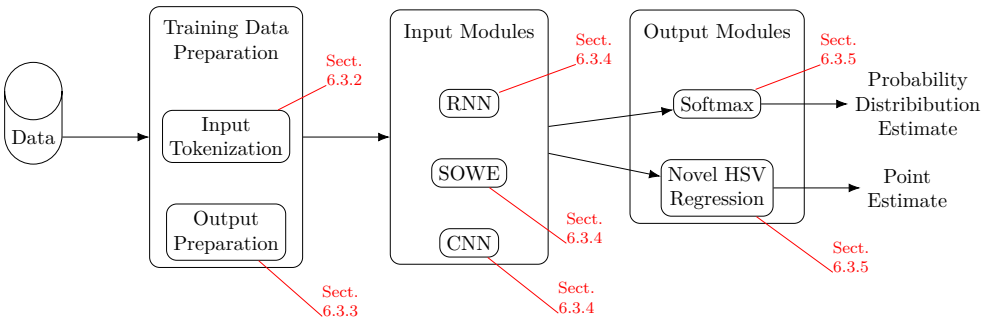


Figure 6.1: The overall architecture of our systems

the Munroe dataset.

The work presented here closes the gap, that while we have language models conditional upon color, we do not have color models conditional upon language.

6.3 Method

6.3.1 System Architecture

Our overall system architecture for all models is shown in Figure 6.1. This shows how color names are transformed into distribution or point estimates over the HSV color-space.

6.3.2 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` 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 6.3.2). 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 modelling 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 (**bojanowski2016enrichin**

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 6.6.3.

6.3.3 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 conditionality 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

[1] Methods do exist to generate embeddings for out of vocabulary words (like `blurple`), particularly with FastText embeddings (**bojanowski2016enriching**). But we do not investigate those here.

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

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 6.8.1). 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 6.3.3), 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 when using histograms over a truely continuous estimation method, such as a Gaussian mixture model. Although such models may have other advantages (such as the a priori 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/OordDZSVGKSK16** apply a similar discretization step and found that 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 create 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. This is

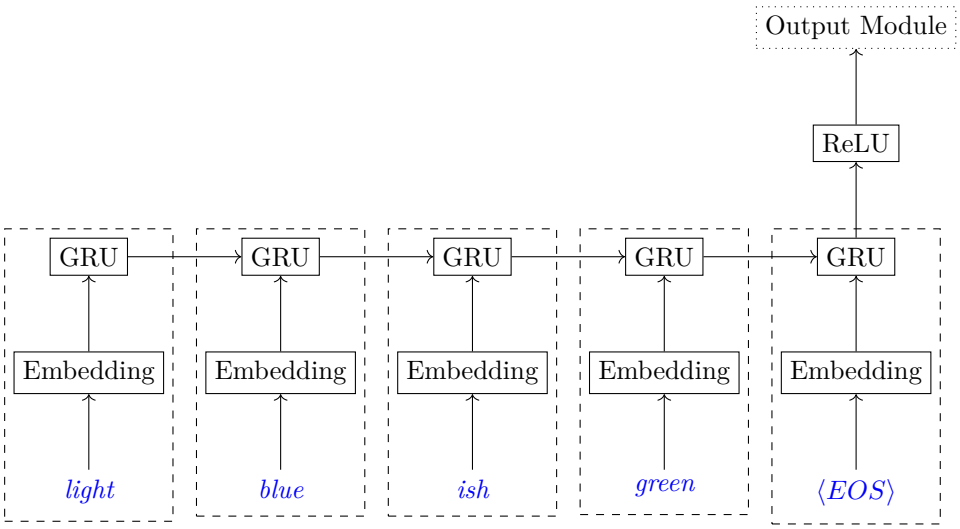


Figure 6.2: The RNN Input module for the example input `light greenish blue`. Each dashed box represents 1 time-step.

the form of tabulation commonly used during the first step of performing kernel density estimation, prior to the application of the kernel.

6.3.4 Color Name Representation Learning (Input Modules)

For each of the models we investigate 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 6.3.5). In all models the input and output modules are trained concurrently as a single system.

Recurrent Neural Network(RNN)

A Recurrent Neural Network is a common choice for this kind of task, due to the variable length of the input. The general structure of this network, shown in Figure 6.2 is similar to **2016arXiv160603821M**, 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 efficiently learned. The embedding is used as the input for a Gated Recurrent Unit (GRU). The stream was 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).

We make use of a GRU (**cho2014properties**), which we found to marginally out-perform the basic RNN in preliminary testing. The small improvement is unsurprising, as the color names have at most 5 terms, so longer short term memory is not required.

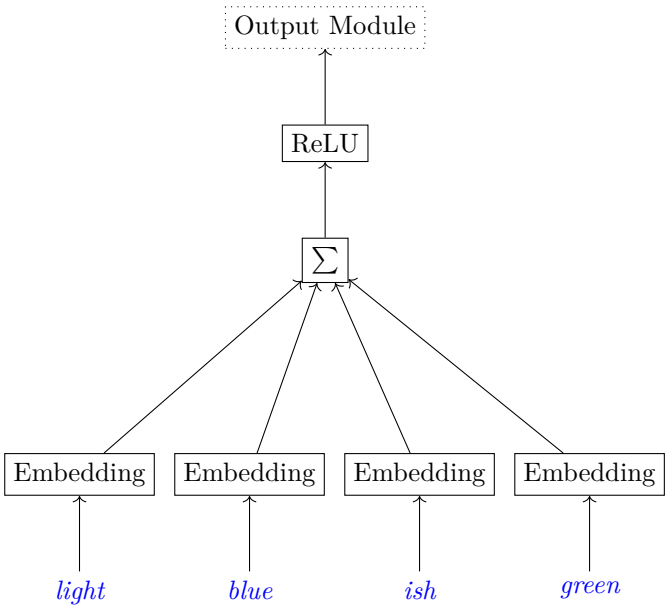


Figure 6.3: The SOWE input module for the example input `light bluish green`

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 **White2015SentVecMeaning**; **novelperspective**). 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 6.3.

Convolutional Neural Network(CNN)

A convolutional neural network (shown in Figure 6.4) 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.

6.3.5 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 6.3.3. Making use of the conditional independence

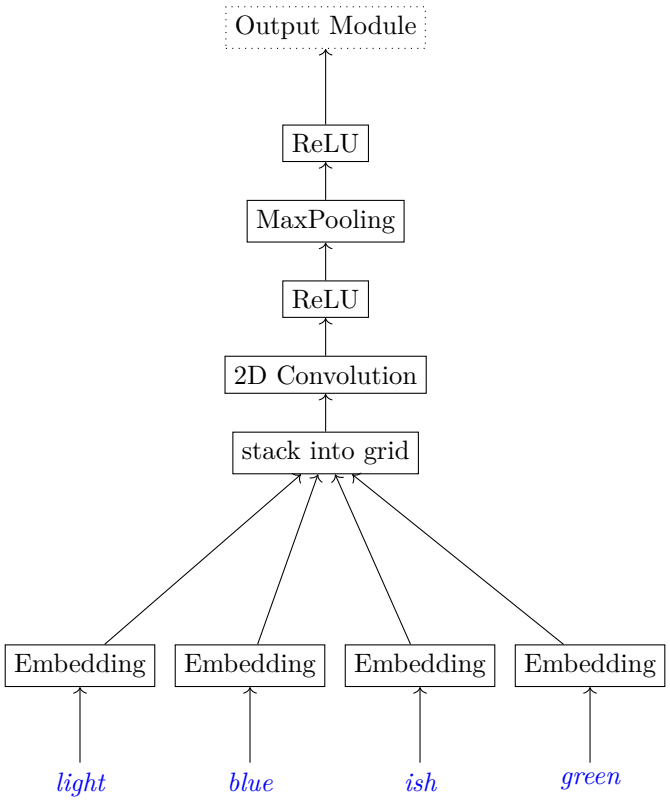


Figure 6.4: The CNN input module for the example input `light greenish blue`

assumption (see Section 6.3.3), we output the three discretized distributions. As shown in Figure 6.5, 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 6.6. The hidden-layer from the top of the input module is fed to four single output neu-

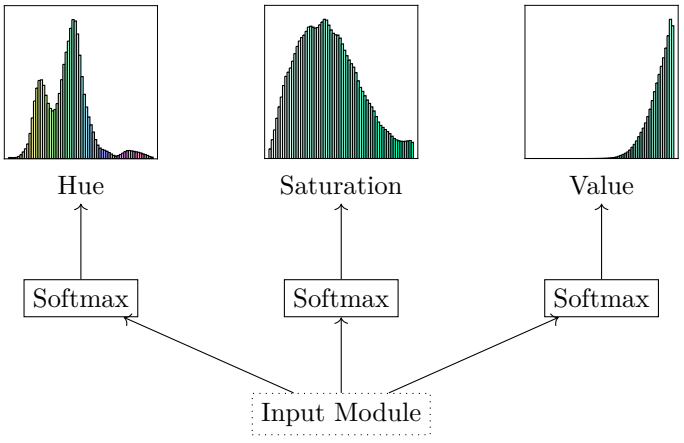


Figure 6.5: The Distribution Output Module

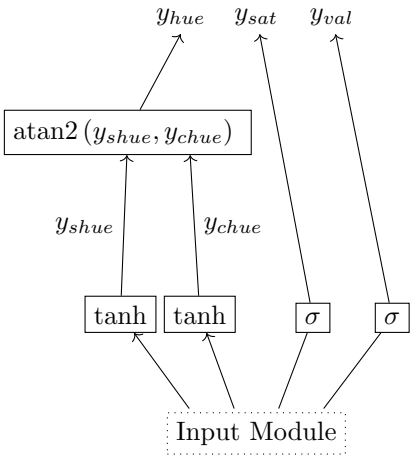


Figure 6.6: The Point Estimate Output Module. Here atan2 is the quadrant preserving arctangent, outputting the angle in turns.

rons.[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 $y^\star$, and each corresponding prediction y.

$$\begin{aligned} loss = &\frac{1}{2}\left(\sin(y^\star_{hue}) - y_{shue}\right)^2 \\ &+ \frac{1}{2}\left(\cos(y^\star_{hue}) - y_{chue}\right)^2 \\ &+ \left(y^\star_{sat} - y_{sat}\right)^2 \\ &+ \left(y^\star_{val} - y_{val}\right)^2 \end{aligned} \tag{6.1}$$

This 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**).

6.4 Evaluation

6.4.1 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.

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

Perplexity is defined as:

$$PP(\tau) = \exp_2\left(\left(\frac{-1}{|\tau|}\sum_{\forall(t,(\tilde{x}))\in\tau}\log_2 p(\tilde{x}\mid t)\right)\right) \tag{6.2}$$

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 standardised 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.

6.4.2 Angularly Correct Calculations on HSV

We use the HSV color-space (**smith1978color**) through-out 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 modelling 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:

$$SE(h_1, h_2) = \min\left(\left(h_1 - h_2\right)^2, \left(h_1 - h_2 - 1\right)^2\right) \tag{6.3}$$

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:

$$\bar{h} = \text{atan2}\left(\frac{1}{N}\sum_{i=1}^{i=N}\sin(h_i),\ \frac{1}{N}\sum_{i=1}^{i=N}\cos(h_i)\right) \tag{6.4}$$

This gives the mean angle.

6.4.3 Operational Upper-bounds

To establish a rough upper-limit on the modelling results we evaluate a direct method which bypasses the language understanding component of the task. These direct 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. This operational upper bound bypass the compositional language understanding part of the process. It is as if the input module (as discussed in Section 6.3.4) would perfectly resolve the sequence of terms into a single item.

They represent an approximate upper bound, as they fully exploit all information in the training data for each input. There is no attempt to determine how each term affects the result. We say approximate upper-bound, as it is not strictly impossible that the term-based methods may happen to model the test data better than can be directly determined by training data as processed per exact color name. 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 6.4.4). Doing this in a way that generalizes to get better results than the direct exploitation of the training data, would require very well calibrated control of (over/under)fitting.

Operational Upper-bound for Distribution Estimation: KDE

To determine an operational upper-bound 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 6.3.3. 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 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 6.8.2. 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 6.6.2, this is because the neural networks are able to learn the smoothness relationship of adjacent bins.

Our KDE-based operational upper bound 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 modelling 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.

Operational Upper-bound 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 of all the training observations for a given exact color name. The mean is taken in the angularly correct way (as discussed in Section 6.4.2). 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 6.1, 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.

6.4.4 Evaluation Strategies

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.

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 operational upper bound models, which bypasses the term processing step. To evaluate this generalisation 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. Similar restriction is done to create a restricted development set, 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 combination, we can assess the capacity of the models 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.

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.

6.5 Experimental Setup

6.5.1 Implementation

The implementation of all the models was in the julia programming language (**Julia**). The full implementation can be downloaded from the GitHub repository.[4] The machine learning components make heavy use of the MLDataUtils.jl[5] and TensorFlow.jl,[6] packages. The latter of which we enhanced significantly to allow for this work to be carried out. The discretization and the KDE for the Operational Upper Bound is done using KernalDensityEstimation.jl.[7]

6.5.2 Common Network Features

Drop-out(**srivastava2014dropout**) is used on all ReLU layers and on the GRU in the RNN, 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.

6.6 Results

6.6.1 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. To illustrate the models trained on the full dataset, Figure 6.7 shows examples of distribution estimates, and Figure 6.8 shows similar examples for point estimates. It can be seen that the models' outputs using term based estimation are generally similar to the non-term-based operational upper-bound, as is intended. This shows that the models are correctly fitting to estimate the colors. This aligns with the strong results found in the quantitative evaluations discussed in Section 6.6.2.

On the effects of word-order

The different input modules have a different capacity to leverage word-order. This is reflected in Figures 6.7 and 6.8, when considering the pairs of outputs that differ only in word order, such as `purple-pink`

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

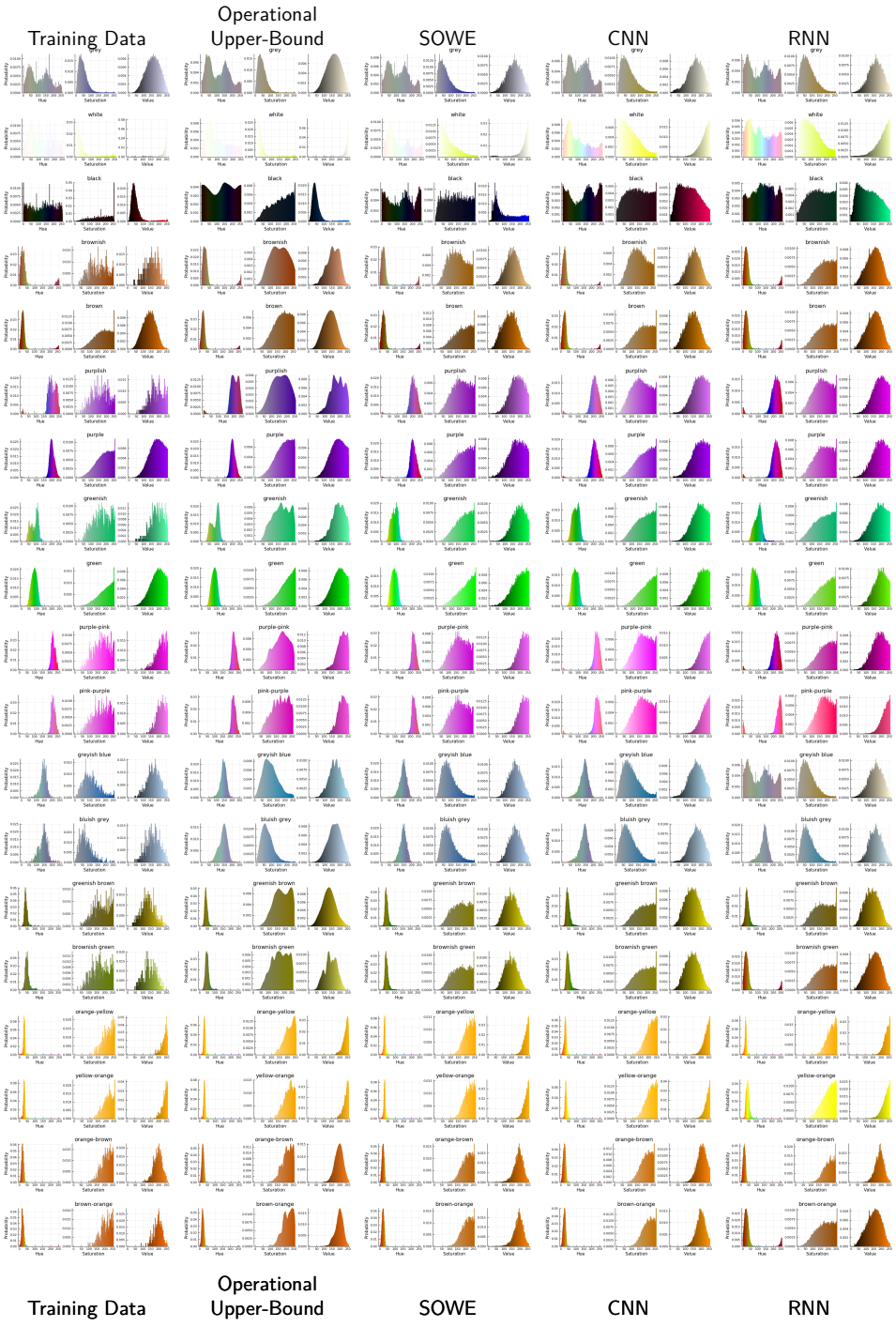


Figure 6.7: Some examples of the output distribution estimates from the models trained on the full dataset

Figure 6.8: Some examples of the output point estimates from the models trained on the full dataset

and `pink-purple`. The plots presented for the training data and for the Operational Upper-Bound 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. The RNN produces the most strikingly different results. It seems that the first term dominates the final output: `purple-pink` is more purple, and `pink-purple` is more pink. 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 RNN is dominated by the first term and not the latter terms[8]. This shows that the GRU is functioning to remember the earlier inputs. This is happening too strongly however, as it is causing incorrect outputs.

On the smoothness of the distribution estimates

In Figure 6.7 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 Operational Upper-Bound 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 with 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 6.8.2.

6.6.2 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 6.1 to 6.6. The CNN and SOWE models perform almost as well as the operational upper-bound. 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. The RNN did not perform as well at these tasks. While it is only marginally

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

behind the SOWE and CNN on the full point estimation task (Table 6.2), 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 shown Section 6.6.2 it did learn generally acceptable colors, but it is not as close a match to the population's expectation.

Ordered Task

The performance of SOWE on the order tasks (Tables 6.3 and 6.4) 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 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 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.

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 6.5 and 6.6 show the results of the model on the test set made up of rare combinations of color names (as described in Section 6.4.4) 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 Operation Upper Bound models are unable to produce estimates from the unseen combinations testing set as they do not process the color names term-wise. The RNN models continue to perform badly on the unseen combination of terms task for both point and distribution estimation.

On distribution estimation (Table 6.5) 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 6.6) the order is flipped with the CNN outperforming the SOWE model. In-fact the CNN actually performs better with the restricted training set. This may be due to the CNN not fitting to the training data as well as the SOWE, as on the full training set (for this test set) we see the SOWE outperforms the CNN. This suggests that the CNN point estimation model may be better at capturing the shared information about term usages, at the expense of

Method	$\frac{PP}{256^3}$
Operational Upper Bound	0.071
SOWE	**0.075**
CNN	0.078
RNN	0.089

Table 6.1: The results for the **full distribution estimation task**. Lower perplexity (PP) is better.

fitting to the final point. Unlike for distribution estimates, the unseen color point estimates are worse than the overall results from the full task (Table 6.2), though the errors are still small 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 6.3.5). However, it is also possible to derive the mean from the distribution estimation models. Those results are also presented in Tables 6.2, 6.4 and 6.6. In general these results perform marginally worse (using the MSE metric) than their corresponding modules using the point estimation output module. We note that for the operational upper-bound, the distributions mean is almost identical to the true mean of points, as expected.

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. 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. We suspect that such a method may have trouble learning the smoothness of the output space (as discussed in Section 6.6.1).

6.6.3 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

Method	MSE
Operational Upper Bound	0.066
SOWE	**0.067**
CNN	**0.067**
RNN	0.071
Distribution Mean Operational Upper Bound	0.066
Distribution Mean SOWE	0.068
Distribution Mean CNN	0.069
Distribution Mean RNN	0.077

Table 6.2: The results for the **full point estimation task**. Lower mean squared error (MSE) is better.

Method	$\frac{PP}{256^3}$
Operational Upper Bound	0.053
SOWE	**0.055**
CNN	0.057
RNN	0.124

Table 6.3: 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	MSE
Operational Upper Bound	0.065
SOWE	**0.066**
CNN	**0.066**
RNN	0.096
Distribution Mean Operational Upper Bound	0.065
Distribution Mean SOWE	**0.066**
Distribution Mean CNN	**0.066**
Distribution Mean RNN	0.095

Table 6.4: 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	Full Training Set $\frac{PP}{256^3}$	Restricted Training Set $\frac{PP}{256^3}$
Operational Upper Bound	0.050	–
SOWE	**0.050**	**0.055**
CNN	0.052	0.065
RNN	0.117	0.182

Table 6.5: The results for the **unseen combinations distribution estimation task**. 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.

Method	Full Training Set MSE	Restricted Training Set MSE
Operational Upper Bound	0.062	–
SOWE	**0.065**	0.079
CNN	0.072	**0.070**
RNN	0.138	0.142
Distribution Mean Operational Upper Bound	0.062	–
Distribution Mean SOWE	0.073	0.076
Distribution Mean CNN	0.073	0.084
Distribution Mean RNN	0.105	0.152

Table 6.6: The results for the **unseen combinations point estimation task**. Lower mean squared error (MSE) 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.

testing data at all. This is even more extreme than the unseen combination task considered in Tables 6.5 and 6.6 where the terms appeared in training, but not the combination of terms. In the examples shown in Figures 6.9 and 6.10, 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 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.

6.7 Conclusion

We have presented three input modules (SOWE, CNN, RNN), 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 an operational upper bound 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 very strong, approaching this upper bound.

A key take away from our results is that using a SOWE should be pre-

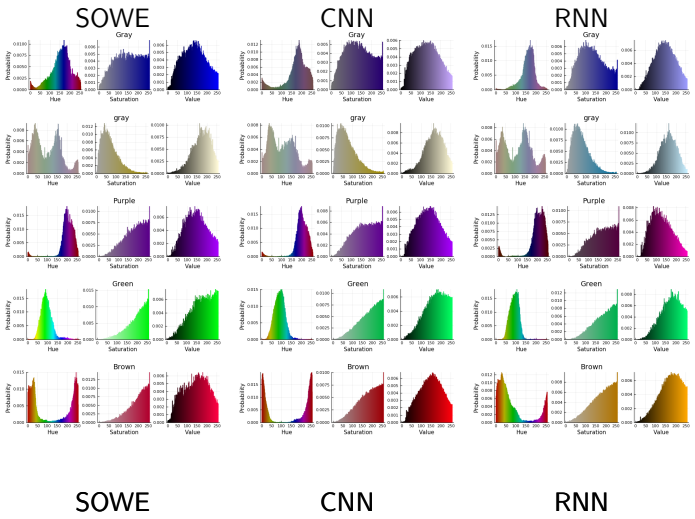


Figure 6.9: Some example distribution estimations for colors names which are completely outside the training data. The terms: `Brown`, `gray`, `Gray`, `Green`, and `Purple`, do not occur in any of the color data; however `brown`, `grey` `green`, and `purple` do occur.

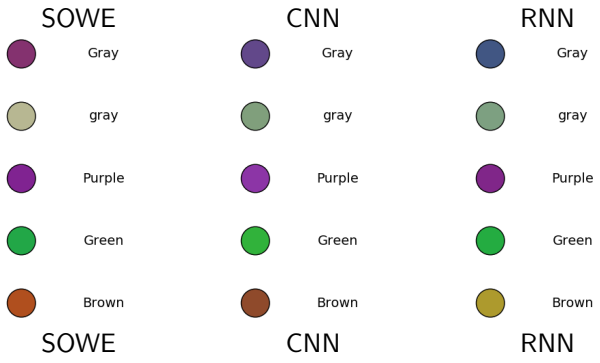


Figure 6.10: Some example point estimates for colors names which are completely outside the training data. The terms: `Brown`, `gray`, `Gray`, `Green`, and `Purple`, do not occur in any of the color data; however `brown`, `grey` `green`, and `purple` do occur.

ferred over an RNN for short phrase natural language understanding tasks when order is not a very significant factor. The RNN is 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 its performance to be significantly exceeded by the SOWE and CNN. The SOWE is an unordered model roughly corresponding to a bag of words. The 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 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. The RNN however must fully process the ordered nature of its inputs, as its output comes only from the final node. It would be interesting to further contrast a bidirectional RNN.

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, this can be used for better prediction in areas such as document classification and sentiment detection.

A further interesting 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 exploited by any known methods. It is expected that color-term usage may vary with subcultures.

6.8 Appendix

6.8.1 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 $S_{ab} = \left\{\left|\rho(V_{a|t}, V_{b|t})\right| \; t \in T_V\right\}$.

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

Table 6.7: The third quartile for the pairwise Spearman's correlation of the color channels given the color name.

We consider the third quartile of that correlation as the indicative statistic in Table 6.7. 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.

6.8.2 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 three term based models (SOWE, CNN, RNN) all perform very similarly whether or not the training data is smoothed. This is seen consistently in all the distribution estimation tasks. Contrast Tables 6.8 to 6.10 to the tables for the unsmoothed results Tables 6.1, 6.3 and 6.5.

If however, smoothing is not applied to the operational upper bound, it works far worse. In Tables 6.8 to 6.10 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 6.10. As these were

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

Table 6.8: 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 6.1.

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

Table 6.9: 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 6.3.

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.

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
RNN-smoothed	0.112	0.183

Table 6.10: 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 6.5.

Chapter 7

Finding Word Sense Embeddings Of Known Meaning

This paper was presented at the 19th Conference on Intelligent Text Processing and Computational Linguistics, in 2018.

Abstract

Word sense embeddings are vector representations of polysemous words – words with multiple meanings. These induced sense embeddings, however, do not necessarily correspond to any dictionary senses of the word. To overcome this, we propose a method to find new sense embeddings with known meaning. We term this method refitting, as the new embedding is fitted to model the meaning of a target word in an example sentence. The new lexically refitted embeddings are learnt using the probabilities of the existing induced sense embeddings, as well as their vector values. Our contributions are threefold: (1) The refitting method to find the new sense embeddings; (2) a novel smoothing technique, for use with the refitting method; and (3) a new similarity measure for words in context, defined by using the refitted sense embeddings. We show how our techniques improve the performance of the Adaptive Skip-Gram sense embeddings for word similarly evaluation; and how they allow the embeddings to be used for lexical word sense disambiguation.

7.1 Introduction

Popular word embedding vectors, such as Word2Vec, represent a word's semantic meaning and its syntactic role as a point in a vector space (**mikolov2013efficient**; **pennington2014glove**). As each word is only given one embedding, such methods are restricted to the representation of only a single combined sense, or meaning, of the word. *Word sense embeddings* generalise word embeddings to handle polysemous and homonymous words. Often these sense embeddings are learnt through unsupervised Word Sense Induction (WSI) (**Reisinger2010**; **Huang2012**; **tian2014probabilistic**; **AdaGrams**). The induced sense embeddings are unlikely to directly coincide with any set of human defined meaning at all, i.e. they will not match lexical senses such as those defined in a lexical dictionary, e.g. WordNet (**miller1995wordnet**). These induced

senses may be more specific, more broad, or include the meanings of jargon not in common use.

One may argue that WSI systems can capture better word senses than human lexicographers do manually. However, this does not mean that induced senses can replace standard lexical senses. It is important to appreciate the vast wealth of existing knowledge defined around lexical senses. Methods to link induced senses to lexical senses allow us to take advantage of both worlds.

We propose *a refitting method* to generate a sense embedding vector that matches with a labelled lexical sense. Given an example sentence with the labelled lexical sense of a particular word, the refitting method algorithmically combines the induced sense embeddings of the target word such that the likelihood of the example sentence is maximised. We find that in doing so, the sense of the word in that sentence is captured. With the refitting, the induced sense embeddings are now able to be used in more general situations where standard senses, or user defined senses are desired.

Refitting word sense vectors to match a lexicographical sense inventory, such as WordNet or a translator's dictionary, is possible if the sense inventory features at least one example of the target sense's use. Our method allows this to be done very rapidly, and from only the single example of use this has with possible applications in low-resource languages.

Refitting can also be used to fit to a user provided example, giving a specific sense vector for that use. This has strong applications in information retrieval. The user can provide an example of a use of the word they are interested in. For example, searching for documents about "`banks`" as in "the river banks were very muddy". By generating an embedding for that specific sense, and by comparing with the generated embeddings in the indexed documents, we can not only pick up on suitable uses of other-words for example "`beach`" and "`shore`", but also exclude different usages, for example of a financial bank. The method we propose, using our refitted embeddings, has lower time complexity than AvgSimC (**Reisinger2010**), the current standard method for evaluating the similarity of words in context. This is detailed in Section 7.5.1.

We noted during refitting, that a single induced sense would often dominate the refitted representation. It is rare in natural language for the meaning to be so unequivocal. Generally, a significant overlap exists between the meaning of different lexical senses, and there is often a high level of disagreement when humans are asked to annotate a corpus (**veronis1998study**). We would expect that during refitting there would likewise be contention over the most likely induced sense. Towards this end, we develop a smoothing method, which we call *geometric smoothing* that de-emphasises the sharp decisions made by the (unsmoothed) refitting method. We found that this significantly improves the results. This suggests that the sharpness of sense decisions is an issue with the language model, which smoothing can correct. The geometric smoothing method is presented in Section 7.3.2.

We demonstrate the refitting method on sense embedding vectors induced using Adaptive Skip-Grams (AdaGram) (**AdaGrams**), as well as our

own simple greedy word sense embeddings. The method is applicable to any skip-gram-like language model that can take a sense vector as its input, and can output the probability of a word appearing in that sense's context.

The rest of the paper is organised as follows: Section 7.2 briefly discusses two areas of related works. Section 7.3 presents our refitting method, as well as our proposed geometric smoothing method. Section 7.4 describes the WSI embedding models used in the evaluations. Section 7.5 defines the RefittedSim measure for word similarity in context, and presents its results. Section 7.6 shows how the refitted sense vectors can be used for lexical WSD. Finally, the paper concludes in Section 7.7.

7.2 Related Works

7.2.1 Directly Learning Lexical Sense Embeddings

In this area of research, the induction of word sense embeddings is treated as a supervised, or semi-supervised task, that requires sense labelled corpora for training.

Iacobacci et al. (**iacobacci2015sensembed**) use a Continuous Bag of Word language model (**mikolov2013efficient**), using word senses as the labels rather than words. This is a direct application of word embedding techniques. To overcome the lack of a large sense labelled corpus, Iacobacci et al. use a 3rd party WSD tool, BabelFly (**Moro2014**), to add sense annotations to a previously unlabelled corpus.

Chen et al. (**Chen2014**) use a supervised approach to train sense vectors, with an unsupervised WSD labelling step. They partially disambiguate their training corpus, using word sense vectors based on WordNet; and use these labels to train their embeddings. This relabelled data is then used as training data, for finding sense embeddings using skip-grams.

Our refitting method learns a new sense embedding as a weighted sum of existing induced sense embeddings of the target word. Refitting is a one-shot learning solution, as compared to the approaches used in the works discussed above. A notable advantage is the time taken to add a new sense. Adding a new sense is practically instantaneous, and replacing the entire sense inventory, of several hundred thousand senses, is only a matter of a few hours. Whereas for the existing approaches this would require repeating the training process, which will often take several days. Refitting is a process done to word sense embeddings, rather than a method for finding sense embeddings from a large corpus.

7.2.2 Mapping induced senses to lexical senses

By defining a stochastic map between the induced and lexical senses, Agirre et al. (**agirre2006**), propose a general method for allowing WSI systems to be used for WSD. Their work was used in SemEval-2007 Task 02 (**SemEval2007WSIandWSD**) to evaluate all entries. Agirre et al.

use a mapping corpus to find the probability of a lexical sense, given the induced sense according to the WSI system. This is more general than the approach we propose here, which only works for sense embedding based WSI. By exploiting the particular properties of sense embedding based WSI systems we propose a system that can better facilitate the use of this subset of WSI systems for WSD.

7.3 Proposed Refitting Framework

The key contribution of this work is to provide a way to synthesise a word sense embedding given only a single example sentence and a set of pretrained sense embedding vectors. We termed this *refitting* the sense vectors. By refitting the unsupervised vectors we define a new vector, that lines up with the specific meaning of the word from the example sentence.

This can be looked at as a one-shot learning problem, analogous to regression. The training of the induced sense, and of the language model, can be considered an unsupervised pre-training step. The new word sense embedding should give a high value for the likelihood of the example sentence, according to the language model. It should also generalise to give a high likelihood of other contexts where this word sense occurs.

We initially attempted to directly optimise the sense vector to predict the example. We applied the L-BFGS (**nocedal1980updating**) optimisation algorithm with the sense vector being the parameter being optimised over, and the objective being to maximise the probability of the example sentence according to the language model. This was found to generalise poorly, due to over-fitting, and to be very slow. Rather than a direct approach, we instead take inspiration from the locally linear relationship between meaning and vector position that has been demonstrated for word embeddings (**mikolov2013efficient**; **mikolovSkip**; **mikolov2013linguisticsubstructures**).

To refit the induced sense embeddings to a particular meaning of a word, we express that a new embedding as as a weighted combination of the induced sense vectors. The weight is determined by the probability of each induced sense given the context.

Given a collection of induced (unlabelled) embeddings $\mathbf{u} = u_1, ..., u_{n_u}$, and an example sentence $\mathbf{c} = w_1, ..., w_{n_c}$ we define a function $l(\mathbf{u} \mid \mathbf{c})$ which determines the refitted sense vector, from the unsupervised vectors and the context as:

$$l(\mathbf{u} \mid \mathbf{c}) = \sum_{\forall u_i \in \mathbf{u}} u_i P(u_i \mid \mathbf{c}) \tag{7.1}$$

Bayes' Theorem can be used to estimate the posterior predictive distribution $P(u_i \mid \mathbf{c})$.

Bengio et al. (**NPLM**) describe a similar method to Equation (7.1) for finding (single sense) word embeddings for words not found in their vocabulary. The formula they give is as per Equation (7.1), but summing over the entire vocabulary of words (rather than just $\mathbf{u}$).

7.3.1 A General WSD method

Using the language model and application of Bayes' theorem, we define a general word sense disambiguation method that can be used for refitting (Equation (7.1)), and for lexical word sense disambiguation (see Section 7.6). This is a standard approach of using Bayes' theorem (**tian2014probabilistic**; **AdaGrams**). We present it here for completeness.

The context is used to determine which sense is the most suitable for this use of the *target word* (the word being disambiguated). Let $\mathbf{s} = (s_1, ..., s_n)$, be the collection of senses for the target word[1].

Let $\mathbf{c} = (w_1, ..., w_{n_c})$ be a sequence of words making up the context of the target word. For example for the target word *kid*, the context could be $\mathbf{c} = ($ *wow the wool from the, is, so, soft, and, fluffy*$)$, where *kid* is the central word taken from between *the* and *fluffy*.

For any particular sense, s_i, the multiple sense skip-gram language model can be used to find the probability of a word w_j occurring in the context: $P(w_j \mid s_i)$. By assuming the conditional independence of each word w_j in the context, given the sense embedding s_i, the probability of the context can be calculated:

$$P(\mathbf{c} \mid s_i) = \prod_{\forall w_j \in \mathbf{c}} P(w_j \mid s_i) \tag{7.2}$$

The correctness of the conditional independence assumption depends on the quality of the representation – the ideal sense representation would fully capture all information about the contexts it can appear in – thus the other contexts elements would not present any additional information, and so $P(w_a \mid w_b, s_i) = P(w_a \mid s_i)$. Given this, we have an estimate of $P(\mathbf{c} \mid s_i)$ which can be used to find $P(s_i \mid \mathbf{c})$. However, a false assumption of independence contributes towards overly sharp estimates of the posterior distribution **rosenfeld2000two**, which we seek to address in Section 7.3.2 with geometric smoothing.

Bayes' Theorem is applied to this context likelihood function $P(\mathbf{c} \mid s_i)$ and a prior for the sense $P(s_i)$ to allow the posterior probability to be found:

$$P(s_i \mid \mathbf{c}) = \frac{P(\mathbf{c} \mid s_i)P(s_i)}{\sum_{s_j \in \mathbf{s}} P(\mathbf{c} \mid s_j)P(s_j)} \tag{7.3}$$

This is the probability of the sense given the context.

7.3.2 Geometric Smoothing for General WSD

During refitting, we note that often one induced sense would be calculated as having much higher probability of occurring than the others (according to Equation (7.3)). This level of certainty is not expected to occur in natural languages, ambiguity is almost always possible. To resolve such dominance problems, we propose a new *geometric smoothing* method. This is suitable for smoothing posterior probability estimates derived

[1] As this part of our method is used with both the unsupervised senses and the lexical senses, referred to as **u** and **l** respectively in other parts of the paper, here we use a general sense **s** to avoid confusion.

from products of conditionally independent likelihoods. It smooths the resulting distribution, by shifting all probabilities to be closer to the uniform distribution.

We hypothesize that the sharpness of probability estimates from Equation (7.3) is a result of data sparsity, and of a false independence assumption in Equation (7.2). This is well known to occur for n-gram language models **rosenfeld2000two**. Word-embeddings language models largely overcome the data sparsity problem due to weight sharing effects (**NPLM**). We suggest that the problem remains for word sense embeddings, where there are many more classes. Thus the training data must be split further between each sense than it was when split for each word. The power law distribution of word use (**zipf1949human**) is compounded by word senses within those used also following the a power law distribution (**Kilgarriff2004**). Rare senses are liable to over-fit to the few contexts they do occur in, and so give disproportionately high likelihoods to contexts that those are similar to. We propose to handle these issues through additional smoothing.

We consider replacing the unnormalised posterior with its n_c-th root, where n_c is the length of the context. We replace the likelihood of Equation (7.2) with $P_S(\mathbf{c} \mid s_i) = \prod_{\forall w_j \in \mathbf{c}} \sqrt[n_c]{P(w_j \mid s_i)}$. Similarly, we replace the prior with: $P_S(s_i) = \sqrt[n_c]{P(w_j \mid s_i)}$ When this is substituted into Equation (7.3), it becomes a smoothed version of $P(s_i \mid \mathbf{c})$.

$$P_S(s_i \mid \mathbf{c}) = \frac{\sqrt[n_c]{P(\mathbf{c} \mid s_i)P(s_i)}}{\sum_{s_j \in \mathbf{s}} \sqrt[n_c]{P(\mathbf{c} \mid s_j)P(s_j)}} \tag{7.4}$$

The motivation for taking the n_c-th root comes from considering the case of the uniform prior. In this case $P_S(\mathbf{c} \mid s_i)$ is the geometric mean of the individual word probabilities $P_S(w_j \mid s_i)$. Consider, if one has two context sentences, $\mathbf{c} = \{w_1, ..., w_{n_c}\}$ and $\mathbf{c}' = \{w'_1, ..., w'_{n_{c'}}\}$, such that $n'_c > n'_c$ then using Equation (7.2) to calculate $P(\mathbf{c} \mid s_i)$ and $P(\mathbf{c}' \mid s_i)$ will result in incomparable results as additional number of probability terms will dominate – often significantly more than the relative values of the probabilities themselves. The number of words that can occur in the context of any given sense is very large – a large portion of the vocabulary. We would expect, averaging across all words, that each addition word in the context would decrease the probability by a factor of $\frac{1}{V}$, where V is the vocabulary size. The expected probabilities for $P(\mathbf{c} \mid s_i)$ is $\frac{1}{V^{n_c}}$ and for $P(\mathbf{c}' \mid s_i)$ is $\frac{1}{V^{n_{c'}}}$. As $n_{c'} > n_c$, thus we expect $P(\mathbf{c}' \mid s_i) \ll P(\mathbf{c} \mid s_i)$. Taking the n_c-th and $n_{c'}$-th roots of $P(\mathbf{c} \mid s_i)$ and $P(\mathbf{c} \mid s_i)$ normalises these probabilities so that they have the same expected value; thus making a context-length independent comparison possible. When this normalisation is applied to Equation (7.3), we get the smoothing effect.

7.4 Experimental Sense Embedding Models

We trained two sense embedding models, AdaGram (**AdaGrams**) and our own Greedy Sense Embedding method. During training we use the

Wikipedia dataset as used by Huang et al. (**Huang2012**). However, we do not perform the extensive preprocessing used in that work.

Most of our evaluations are carried out on Adaptive SkipGrams (AdaGram) (**AdaGrams**). AdaGram is a non-parametric Bayesian extension of Skip-gram. It learns a number of different word senses, as are required to properly model the language.

We use the implementation[2] provided by the authors with minor adjustments for Julia (**Julia**) v0.5 compatibility.

The AdaGram model was configured to have up to 30 senses per word, where each sense is represented by a 100 dimension vector. The sense threshold was set to 10^{-10} to encourage many senses. Only words with at least 20 occurrences are kept, this gives a total vocabulary size of 497,537 words.

To confirm that our techniques are not merely a quirk of the AdaGram method or its implementation, we implemented a new simple baseline word sense embedding method. This method starts with a fixed number of randomly initialised embeddings, then greedily assigns each training case to the sense which predicts it with the highest probability (using Equation (7.3)). The task remains the same: using skip-grams with hierarchical softmax to predict the context words for the input word sense. This is similar to **neelakantan2015efficient**, however it is using collocation probability, rather than distance in vector-space as the sense assignment measure. Our implementation is based on a heavily modified version of Word2Vec.jl[3].

This method is intrinsically worse than AdaGram. Nothing in the model encourages diversification and specialisation of the embeddings. Manual inspection reveals that a variety of senses are captured, though with significant repetition of common senses, and with rare senses being missed. Regardless of its low quality, it is a fully independent method from AdaGram, and so is suitable for our use in checking the generalisation of the refitting techniques.

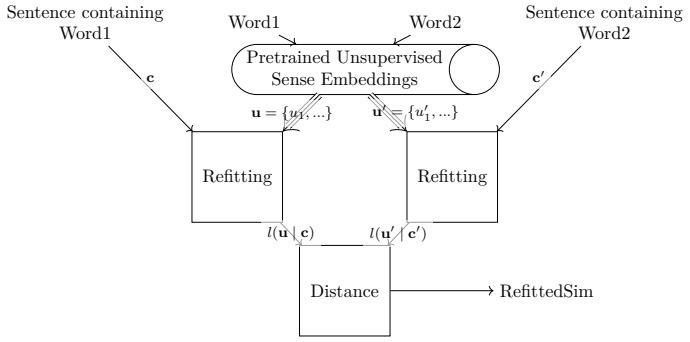
The vocabulary used is smaller than for the AdaGram model. Words with at least 20,000 occurrences are allocated 20 senses. Words with at least 250 occurrences are restricted to a single sense. The remaining rare words are discarded. This results in a vocabulary size of 88,262, with 2,796 words having multiple senses. We always use a uniform prior, as the model does not facilitate easy calculation of the prior.

7.5 Similarity of Words in Context

Estimating word similarity with context is the task of determining how similar words are, when presented with the context they occur in. The goal of this task is to match human judgements of word similarity. For each of the target words and contexts; we use refitting on the target word to create a word sense embedding specialised for the meaning in the context provided. Then the similarity of the refitted vectors can be

[2] `https://github.com/sbos/AdaGram.jl`
[3] `https://github.com/tanmaykm/Word2Vec.jl/`

Figure 7.1: Block diagram for RefittedSim similarity measure



measured using cosine distance (or similar). By measuring similarity this way, we are defining a new similarity measure.

Reisinger and Mooney (**Reisinger2010**) define a number of measures for word similarity suitable for use with sense embeddings. The most successful was AvgSimC, which has become the gold standard method for use on similarity tasks. It has been used with great success in many works **Huang2012**; **Chen2014**; **tian2014probabilistic**.

AvgSimC is defined using distance metric d (normally cosine distance) as:

$$\text{AvgSimC}((\mathbf{u},\mathbf{c}),(\mathbf{u}',\mathbf{c}')) = \frac{1}{n \times n'} \sum_{u_i \in \mathbf{u}} \sum_{u'_j \in \mathbf{u}'} P(u_i \mid \mathbf{c})\, P(u'_j \mid \mathbf{c}')\, d(u_i, u'_j) \tag{7.5}$$

for contexts $\mathbf{c}$ and $\mathbf{c}'$, the contexts of the two words to be compared, and for $\mathbf{u} = \{u_1, ..., u_n\}$ and $\mathbf{u}' = \{u'_1, ..., u\prime_{n'}\}$ the respective sets of induced senses of the two words.

7.5.1 A New Similarity Measure: RefittedSim

We define a new similarity measure, RefittedSim, as the distance between the refitted sense embeddings. As shown in Figure 7.1 the example contexts are used to refit the induced sense embeddings of each word. This is a direct application of Equation (7.1).

Using the same definitions as in Equation (7.5), RefittedSim is defined as:

$$\text{RefittedSim}((\mathbf{u},\mathbf{c}),(\mathbf{u}',\mathbf{c}')) \;=\; d(l(\mathbf{u} \mid \mathbf{c}), l(\mathbf{u}' \mid \mathbf{c}') = d\left(\textstyle\sum_{u_i \in \mathbf{u}} u_i P(u_i \mid \mathbf{c}),\, \sum_{u'_j \in \mathbf{u}'} u_i P(u'_j \mid \mathbf{c}')\right) \tag{7.6}$$

AvgSimC is a probability weighted average of pairwise computed distances for each sense vector. Whereas RefittedSim is a single distance measured between the two refitted vectors – which are the probability weighted averages of the original unsupervised sense vectors.

There is a notable difference in time complexity between AvgSimC and RefittedSim. AvgSimC has time complexity $O(n \left\|\mathbf{c}\right\| + n' \left\|\mathbf{c}'\right\| + n \times n')$, while RefittedSim has $O(n \left\|\mathbf{c}\right\| + n' \left\|\mathbf{c}'\right\|)$. The product of the number of senses of each word $n \times n'$, may be small for dictionary senses, but it is often large for induced senses. Dictionaries tend to define only a few

Table 7.1: Spearman rank correlation $\rho \times 100$ when evaluated on the SCWS task, for varying hyper-parameters.

Method	Geometric Smoothing	Use Prior	AvgSimC	RefittedSim
AdaGram	T	T	**53.8**	64.8
AdaGram	T	F	36.1	**65.0**
AdaGram	F	T	43.8	47.8
AdaGram	F	F	20.7	24.1
Greedy	T	F	23.6	49.7
Greedy	F	F	22.2	40.7

Table 7.2: Spearman rank correlation $\rho \times 100$ when evaluated on the SCWS task, compared to other methods . RefittedSim-S is with smoothing, and RefittedSim-SU is with uniform prior

Paper	Embedding	Similarity	$\rho \times 100$
This paper	AdaGram	AvgSimC	43.8
This paper	AdaGram	RefittedSim-S	64.8
This paper	AdaGram	RefittedSim-SU	65.0
Huang2012	Huang et al.	AvgSimC	65.7
Huang2012	Pruned tf-idf	AvgSimC	60.5
Chen2014	Chen et al.	AvgSimC	68.9
tian2014probabilistic	Tian et al.	AvgSimC	65.4
tian2014probabilistic	Tian et al.	MaxSim	**65.6**
iacobacci2015sensembed	SenseEmbed	Min Tanimoto	58.9
iacobacci2015sensembed	SenseEmbed	Weighted Tanimoto	62.4

senses per word – the average[4] number of senses per word in WordNet is less than three (**miller1995wordnet**). For induced senses, however, it is often desirable to train many more senses, to get better results using the more fine-grained information. Reisinger and Mooney (**Reisinger2010**) found optimal results in several evaluations near 50 senses. In such cases the $O(n \times n')$ is significant, avoiding it with RefittedSim makes the similarity measure more useful for information retrieval.

7.5.2 Experimental Setup

We evaluate our refitting method using Stanford's Contextual Word Similarities (SCWS) dataset (**Huang2012**). During evaluation, each context paragraph is limited to 5 words to either side of the target word, as in the training.

7.5.3 Results

Table 7.1 shows the results of our evaluations on the SCWS similarity task. A significant improvement can be seen by applying our techniques.

The RefittedSim method consistently outperforms AvgSimC across all configurations. Similarly geometric smoothing consistently improves performance both for AvgSimC and for RefittedSim. The improvement is significantly more for RefittedSim than for AvgSimC results. In general using the unsupervised sense prior estimate from the AdaGram model, improves performance – particularly for AvgSimC. The exception to this

[4]It should be noted, though, that the number of meanings is not normally distributed (**zipf1945meaning**).

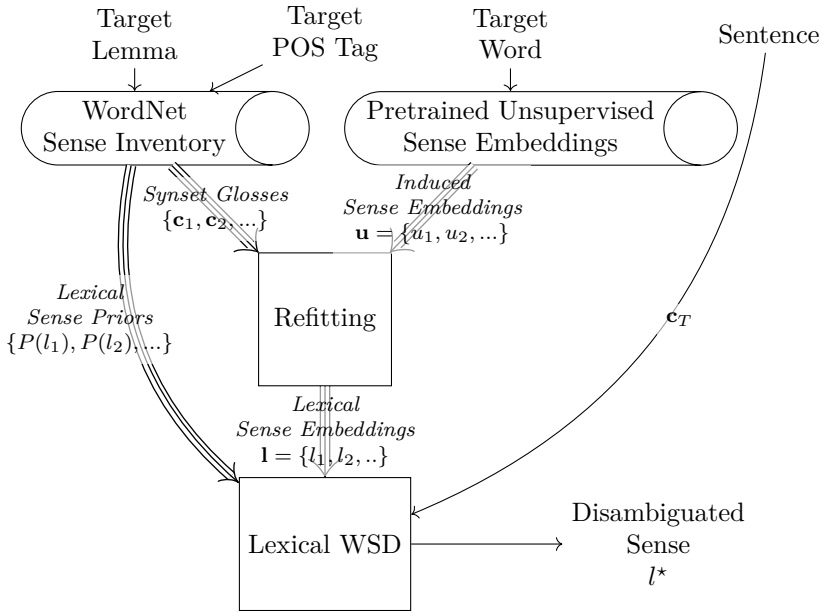


Figure 7.2: Block diagram for performing WSD using refitting.

is with RefittedSim with smoothing, where it makes very little difference. Unsurprisingly, given its low quality, the Greedy embeddings are always outperformed by AdaGram. It is not clear if these improvements will transfer to clustering based methods due to the differences in how the sense probability is estimated, compared to the language model based method evaluated on in Table 7.1.

Table 7.2 compares our results with those reported in the literature using other methods. These results are not directly comparable, as each method uses a different training corpus, with different preprocessing steps, which can have significant effects on performance. It can been seen that by applying our techniques we bring the results of our AdaGram model from very poor ($\rho \times 100 = 43.8$) when using normal AvgSimC without smoothing, up to being competitive with other models, when using RefittedSim with smoothing. The method of Chen et al. (**Chen2014**), has a significant lead on the other results presented. This can be attributed to its very effective semi-supervised fine-tuning method. This suggests a possible avenue for future development in using refitted sense vectors to relabel a corpus, and then performing fine-tuning similar to that done by Chen et al.

7.6 Word Sense Disambiguation

7.6.1 Refitting for Word Sense Disambiguation

Once refitting has been used to create sense vectors for lexical word senses, an obvious used of them is to perform word sense disambiguation. In this section we refer to the lexical word sense disambiguation problem, i.e. to take a word and find its dictionary sense; whereas the methods discussed in Equations (7.3) and (7.4) consider the more general problem, as applicable to disambiguating lexical or induced word senses depending on the inputs. Our overall process shown in Figure 7.2 uses both: first

disambiguating the induced senses as part of refitting, then using the refitted sense vectors to find the most likely dictionary sense.

First, refitting is used to transform the induced sense vectors into lexical sense vectors. We use the targeted word's lemma (i.e. base form), and part of speech (POS) tag to retrieve all possible definitions of the word (Glosses) from WordNet; there is one gloss per sense. These glosses are used as the example sentence to perform refitting (see Section 7.3). We find embeddings, $\mathbf{l} = \{l_1, ..., l_{n_l}\}$ for each of the lexical word senses using Equation (7.1). These lexical word senses are still supported by the language model, which means one can apply the general WSD method to determine the posterior probability of a word sense, given an observed context.

When given a sentence $\mathbf{c}_T$, containing a target word to be disambiguated, the probability of each lexical word sense $P(l_i \mid \mathbf{c}_T)$, can be found using Equation (7.3) (or the smoothed version Equation (7.4)), over the lexically refitted sense embeddings. Then, selecting the correct sense is simply selecting the most likely sense:

$$l^\star(\mathbf{l}, \mathbf{c}_T) = \underset{\forall l_i \in \mathbf{l}}{\operatorname{argmax}} P(l_i|\mathbf{c}_T) = \underset{\forall l_i \in \mathbf{l}}{\operatorname{argmax}} \frac{P(\mathbf{c}_T \mid l_i)P(l_i)}{\sum_{\forall l_j \in \mathbf{l}} P(\mathbf{c}_T \mid l_j)P(l_j)} \tag{7.7}$$

7.6.2 Lexical Sense Prior

WordNet includes frequency counts for each word sense based on Semcor (**tengi1998design**). These form a prior for $P(l_i)$. The comparatively small size of Semcor means that many word senses do not occur at all. We apply add-one smoothing to remove any zero counts. This is in addition to using our proposed geometric smoothing as an optional part of the general WSD. Geometric smoothing serves a different (but related) purpose, of decreasing the sharpness of the likelihood function – not of removing impossibilities from the prior.

7.6.3 Experimental Setup

The WSD performance is evaluated on the SemEval 2007 Task 7.

We use the weighted mapping method of Agirre et al. (**agirre2006**), (see Section 7.2.2) as a baseline alternative method for using WSI senses for WSD. We use Semcor as the mapping corpus, to derive the mapping weights.

The second baseline we use is the Most Frequent Sense (MFS). This method always disambiguates any word as having its most common meaning. Due to the power law distribution of word senses, this is a very effective heuristic (**Kilgarriff2004**). We also consider the results when using a backoff to MSF when a method is unable to determine the word sense the method can report the MFS instead of returning no result (a non-attempt).

Method	Attempted	Precision	Recall	F1
Refitted-S AdaGram	99.91%	0.799	0.799	**0.799**
Refitted AdaGram	99.91%	0.774	0.773	0.774
Refitted-S Greedy	79.95%	0.797	0.637	0.708
Refitted-S Greedy $*$	100.00%	0.793	0.793	0.793
Refitted Greedy	79.95%	0.725	0.580	0.645
Refitted Greedy $*$	100.00%	0.793	0.793	0.793
Mapped AdaGram	84.31%	0.776	0.654	0.710
Mapped AdaGram $*$	100.00%	0.736	0.736	0.736
MFS baseline	100.00%	0.789	0.789	0.789

Table 7.3: Results on SemEval 2007 Task 7 – course-all-words disambiguation. The *-S* marks results using geometric smoothing. The $*$ marks results with MSF backoff.

7.6.4 Word Sense Disambiguation Results

The results of employing our method for WSD , are shown in Table 7.3. Our results using smoothed refitting, both with AdaGram and Greed Embeddings with backoff, outperform the MSF baseline **Navigli:2007:STC:1621474.1621480** – noted as a surprisingly hard baseline to beat (**Chen2014**).

The mapping method (**agirre2006**) was not up to the task of mapping unsupervised senses to supervised senses, on this large scale task. The Refitting method works better. Though refitting is only usable for language-model embedding WSI, the mapping method is suitable for all WSI systems.

While not directly comparable due to the difference in training data, we note that our Refitted results, are similar in performance, as measured by F1 score, to the results reported by Chen et al. (**Chen2014**). AdaGram with smoothing, and Greedy embeddings with backoff have close to the same result as reported for L2R with backoff – with the AdaGram slightly better and the Greedy embeddings slightly worse. They are exceeded by the best method reported in that paper: S2C method with backoff. Comparison to non-embedding based methods is not discussed here for brevity. Historically state of the art systems have functioned very differently; normally by approaching the WSD task by more direct means.

Our results are not strong enough for Refitted AdaGram to be used as a WSD method on its own, but do demonstrate that the senses found by refitting are capturing the information from lexical senses. It is now evident that the refitted sense embeddings are able to perform WSD, which was not possible with the unsupervised senses.

7.7 Conclusion

A new method is proposed for taking unsupervised word embeddings, and adapting them to align to particular given lexical senses, or user provided usage examples. This refitting method thus allows us to find word sense embeddings with known meaning. This method can be seen as a one-shot learning task, where only a single labelled example of each class is available for training. We show how our method can be used to create embeddings to evaluate the similarity of words, given their contexts.

This allows us to propose a new similarity measuring method, RefittedSim. The performance of RefittedSim on AdaGram is comparable to the results reported by the researchers of other sense embeddings techniques using AvgSimC, but its time complexity is significantly lower. We also demonstrate how similar refitting principles can be used to create a set of vectors that are aligned to the meanings in a sense inventory, such as WordNet.

We show how this can be used for word sense disambiguation. On this difficult task, it performs marginally better than the hard to beat MFS baseline, and significantly better than a general mapping method used for working with WSI senses on lexical WSD tasks. As part of our method for refitting, we present a geometric smoothing to overcome the issues of overly dominant senses probability estimates. We show that this significantly improves the performance. Our refitting method provides effective bridging between the vector space representation of meaning, and the traditional discrete lexical representation. More generally it allows a sense embedding to be created to model the meaning of a word in any given sentence. Significant applications of sense embeddings in tasks such as more accurate information retrieval thus become possible.

Chapter 8

NovelPerspective: Identifying Point of View Characters

This paper was presented at 56th Annual Meeting of the Association for Computational Linguistics (ACL) in 2018, in the System Demonstrations track.

Abstract

We present NovelPerspective: a tool to allow consumers to subset their digital literature, based on point of view (POV) character. Many novels have multiple main characters each with their own storyline running in parallel. A well-known example is George R. R. Martin's novel: "A Game of Thrones", and others from that series. Our tool detects the main character that each section is from the POV of, and allows the user to generate a new ebook with only those sections. This gives consumers new options in how they consume their media; allowing them to pursue the storylines sequentially, or skip chapters about characters they find boring. We present two heuristic-based baselines, and two machine learning based methods for the detection of the main character.

8.1 Introduction

Often each section of a novel is written from the perspective of a different main character. The characters each take turns in the spot-light, with their own parallel storylines being unfolded by the author. As readers, we have often desired to read just one storyline at a time, particularly when reading the book a second-time. In this paper, we present a tool, NovelPerspective, to give the consumer this choice.

Our tool allows the consumer to select which characters of the book they are interested in, and to generate a new ebook file containing just the sections from that character's point of view (POV). The critical part of this system is the detection of the POV character. This is not an insurmountable task, building upon the well established field of named entity recognition. However to our knowledge there is no software to do this. Such a tool would have been useless, in decades past when booked were distributed only on paper. But today, the surge in popularity of ebooks has opened a new niche for consumer narrative processing. Methods are being created to extract social relationships between characters (**elson2010socialnetworks**; **wohlgenannt2016extracting**); to

align scenes in movies with those from books (**moviebook**); and to otherwise augment the literature consumption experience. Tools such as the one presented here, give the reader new freedoms in controlling how they consume their media.

Having a large cast of characters, in particular POV characters, is a hallmark of the epic fantasy genre. Well known examples include: George R.R. Martin's "A Song of Ice and Fire", Robert Jordan's "Wheel of Time", Brandon Sanderson's "Cosmere" universe, and Steven Erikson's "Malazan Book of the Fallen", amongst thousands of others. Generally, these books are written in *limited* third-person POV; that is to say the reader has little or no more knowledge of the situation described than the main character does.

We focus here on novels written in the *limited* third-person POV. In these stories, the main character is, for our purposes, the POV character. Limited third-person POV is written in the third-person, that is to say the character is referred to by name, but with the observations limited to being from the perspective of that character. This is in-contrast to the *omniscient* third-person POV, where events are described by an external narrator. Limited third-person POV is extremely popular in modern fiction. It preserves the advantages of first-person, in allowing the reader to observe inside the head of the character, while also allowing the flexibility to the perspective of another character (**booth2010rhetoric**). This allows for multiple concurrent storylines around different characters. Our tool helps users un-entwine such storylines, giving the option to process them sequentially.

The utility of dividing a book in this way varies with the book in question. Some books will cease to make sense when the core storyline crosses over different characters. Other novels, particularly in epic fantasy genre, have parallel storylines which only rarely intersect. While we are unable to find a formal study on this, anecdotally many readers speak of:

- "Skipping the chapters about the boring characters."
- "Only reading the *real* main character's sections."
- "Reading ahead, past the side-stories, to get on with the *main* plot."

Particularly if they have read the story before, and thus do not risk confusion. Such opinions are a matter of the consumer's personal taste. The NovelPerspective tool gives the reader the option to customise the book in this way, according to their personal preference.

We note that sub-setting the novel once does not prevent the reader from going back and reading the intervening chapters if it ceases to make sense, or from sub-setting again to get the chapters for another character whose path intersects with the storyline they are currently reading. We can personally attest for some books reading the chapters one character at a time is indeed possible, and pleasant: the first author of this paper read George R.R. Martin's "A Song of Ice and Fire" series in exactly this fashion.

The primary difficulty in segmenting ebooks this way is attributing each section to its POV character. That is to say detecting who is the point of view character. Very few books indicate this clearly, and the reader

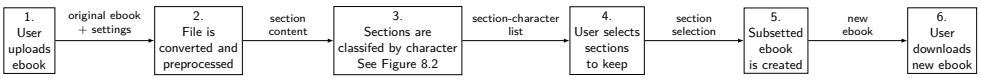


Figure 8.1: The full NovelPerspective pipeline. Note that step 5 uses the original ebook to subset.

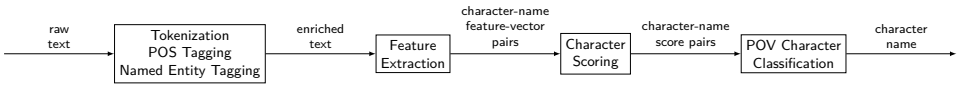


Figure 8.2: The general structure of the character classification systems. This repeated for each section of the book during step 3 of the full pipeline shown in Figure 8.1.

is expected to infer it during reading. This is easy for most humans, but automating it is a challenge. To solve this, the core of our tool is its character classification system. We investigated several options which the main text of this paper will discuss.

8.2 Character Classification Systems

The full NovelPerspective pipeline is shown in Figure 8.1. The core character classification step (step 3), is detailed in Figure 8.2. In this step the raw text is first enriched with parts of speech, and named entity tags. We do not perform co-reference resolution, working only with direct entity mentions. From this, features are extracted for each named entity. These feature vectors are used to score the entities for the most-likely POV character. The highest scoring character is returned by the system. The different systems presented modify the **Feature Extraction** and **Character Scoring** steps. A broadly similar idea, for detecting the focus location of news articles, was presented by **2017focus**.

8.2.1 Baseline systems

To the best of our knowledge no systems have been developed for this task before. As such, we have developed two deterministic baseline character classifiers. These are both potentially useful to the end-user in our deployed system (Section 8.5), and used to gauge the performance of the more complicated systems in the evaluations presented in Section 8.4.

It should be noted that the baseline systems, while not using machine learning for the character classification steps, do make extensive use of machine learning-based systems during the preprocessing stages.

"First Mentioned" Entity

An obvious way to determine the main character of the section is to select the first named entity. We use this to define the "First Mentioned" baseline In this system, the **Feature Extraction** step is simply retrieving the position of the first use of each name; and the **Character Scoring** step assigns each a score such that earlier is higher. This works for many examples: *"One dark and stormy night, Bill heard a knock at the door."*;

however it fails for many others: *" 'Is that Tom?' called out Bill, after hearing a knock."*. Sometimes a section may go several paragraphs describing events before it even mentions the character who is perceiving them. This is a varying element of style.

"Most Mentioned" Entity

A more robust method to determine the main character, is to use the occurrence counts. We call this the "Most Mentioned" baseline. The **Feature Extraction** step is to count how often the name is used. The **Character Scoring** step assigns each a score based what proportional of all names used were for this entity. This works well for many books. The more important a character is, the more often their name occurs. However, it is fooled, for example, by book chapters that are about the POV character's relationship with a secondary character. In such cases the secondary character may be mentioned more often.

8.2.2 Machine learning systems

One can see the determination of the main character as a multi-class classification problem. From the set of all named entities in the section, classify that section as to which one is the main character. Unlike typical multi-class classification problems the set of possible classes varies per section being classified. Further, even the total set of possible named characters, i.e. classes, varies from book to book. An information extraction approach is required which can handle these varying classes. As such, a machine learning model for this task can not incorporate direct knowledge of the classes (i.e. character names).

We reconsider the problem as a series of binary predictions. The task is to predict if a given named entity is the point of view character. For each possible character (i.e. each named-entity that occurs), a feature vector is extracted (see Section 8.2.2). This feature vector is the input to a binary classifier, which determines the probability that it represents the main character. The **Character Scoring** step is thus the running of the binary classifier: the score is the output probability normalised over all the named entities.

Feature Extraction for ML

We investigated two feature sets as inputs for our machine learning-based solution. They correspond to different **Feature Extraction** steps in Figure 8.2. A hand-engineered feature set, that we call the "Classical" feature set; and a more modern "Word Embedding" feature set. Both feature sets give information about how the each named entity token was used in the text.

The "Classical" feature set uses features that are well established in NLP related tasks. The features can be described as *positional features*, like in the First Mentioned baseline; *occurrence count features*, like in the *Most Mentioned* baseline and *adjacent POS counts*, to give usage context. The *positional features* are the index (in the token counts) of the first and last

occurrence of the named entity. The *occurrence count features* are simply the number of occurrences of the named entity, supplemented with its rank on that count compared to the others. The *adjacent POS counts* are the occurrence counts of each of the 46 POS tags on the word prior to the named entity, and on the word after. We theorised that this POS information would be informative, as it seemed reasonable that the POV character would be described as doing more things, so co-occurring with more verbs. This gives 100 base features. To allow for text length invariance we also provide each of the base features expressed as a portion of its maximum possible value (e.g. for a given POS tag occurring before a named entity, the potion of times this tag occurred). This gives a total of 200 features.

The "Word Embedding" feature set uses FastText word vectors (**bojanowski2016en**). We use the pretrained 300 dimensional embeddings trained on English Wikipedia [1]. We concatenate the 300 dimensional word embedding for the word immediately prior to, and immediately after each occurrence of a named entity; and take the element-wise mean of this concatenated vector over all occurrences of the entity. Such averages of word embeddings have been shown to be a useful feature in many tasks (**White2015SentVecMeaning**; **mikolovSkip**). This has a total of 600 features.

Classifier

The binary classifier, that predicts if a named entity is the main character, is the key part of the **Character Scoring** step for the machine learning systems. From each text in the training dataset we generated a training example for every named entity that occurred. All but one of these was a negative example. We then trained it as per normal for a binary classifier. The score for a character is the classifier's predicted probability of its feature vector being for the main character.

Our approach of using a binary classifier to rate each possible class, may seem similar to the one-vs-rest approach for multi-class classification. However, there is an important difference. Our system only uses a single binary classifier; not one classifier per class, as the classes in our case vary with every item to be classified. The fundamental problem is information extraction, and the classifier is a tool for the scoring which is the correct information to report.

With the classical feature set we use logistic regression, with the features being preprocessed with 0-1 scaling. During preliminary testing we found that many classifiers had similar high degree of success, and so chose the simplest. With the word embedding feature set we used a radial bias support vector machine, with standardisation during preprocessing, as has been commonly used with word embeddings on other tasks.

[1] `https://fasttext.cc/docs/en/pretrained-vectors.html`

Dataset	Chapters	POV Characters
ASOIAF	256	15
SOC	91	9
WOT	432	52
combined	779	76

Table 8.1: The number of chapters and point of view characters for each dataset.

8.3 Experimental Setup

8.3.1 Datasets

We make use of three series of books selected from our own personal collections. The first four books of George R. R. Martin's "A Song of Ice and Fire" series (hereafter referred to as ASOIAF); The two books of Leigh Bardugo's "Six of Crows" duology (hereafter referred to as SOC); and the first 9 volumes of Robert Jordan's "Wheel of Time" series (hereafter referred to as WOT). In Section 8.4 we consider the use of each as a training and testing dataset. In the online demonstration (Section 8.5), we deploy models trained on the combined total of all the datasets.

To use a book for the training and evaluation of our system, we require a ground truth for each section's POV character. ASOIAF and SOC provide ground truth for the main character in the chapter names. Every chapter only uses the POV of that named character. WOT's ground truth comes from an index created by readers.[2] We do not have any datasets with labelled sub-chapter sections, though the tool does support such works.

The total counts of chapters and characters in the datasets, after preprocessing, is shown in Table 8.1. Preprocessing consisted of discarding chapters for which the POV character was not identified (e.g. prologues); and of removing the character names from the chapter titles as required.

8.3.2 Evaluation Details

In the evaluation, the systems are given the body text and asked to predict the character names. During evaluation, we sum the scores of the characters alternative aliases/nick-names used in the books. For example merging `Ned` into `Eddard` in ASOIAF. This roughly corresponds to the case that a normal user can enter multiple aliases into our application when selecting sections to keep. We do not use these aliases during training, though that is an option that could be investigated in a future work.

8.3.3 Implementation

The full source code is available on GitHub. [3] Scikit-Learn (**scikit-learn**) is used for the machine learning and evaluations, and NLTK (**NLTK**) is used for textual preprocessing. The text is tokenised, and tagged with

[2] `http://wot.wikia.com/wiki/List_of_Point_of_View_Characters`
[3] `https://github.com/oxinabox/NovelPerspective/`

Test Set	Method	Train Set	Acc
ASOIAF	First Mentioned	—	0.250
ASOIAF	Most Mentioned	—	0.914
ASOIAF	ML Classical Features	SOC	0.953
ASOIAF	ML Classical Features	WOT	**0.984**
ASOIAF	ML Classical Features	WOT+SOC	0.977
ASOIAF	ML Word Emb. Features	SOC	0.863
ASOIAF	ML Word Emb. Features	WOT	0.977
ASOIAF	ML Word Emb. Features	WOT+SOC	0.973
SOC	First Mentioned	—	0.429
SOC	Most Mentioned	—	0.791
SOC	ML Classical Features	WOT	0.923
SOC	ML Classical Features	ASOIAF	0.923
SOC	ML Classical Features	WOT+ASOIAF	0.934
SOC	ML Word Emb. Features	WOT	0.934
SOC	ML Word Emb. Features	ASOIAF	**0.945**
SOC	ML Word Emb. Features	WOT+ASOIAF	**0.945**
WOT	First Mentioned	—	0.044
WOT	Most Mentioned	—	0.660
WOT	ML Classical Features	SOC	0.701
WOT	ML Classical Features	ASOIAF	**0.745**
WOT	ML Classical Features	ASOIAF+SOC	0.736
WOT	ML Word Emb. Features	SOC	0.551
WOT	ML Word Emb. Features	ASOIAF	0.699
WOT	ML Word Emb. Features	ASOIAF+SOC	0.681

Table 8.2: The results of the character classifier systems. The best results are **bolded**.

POS and named entities using NLTK's default methods. Specifically, these are the Punkt sentence tokenizer, the regex-based improved Tree-Bank word tokenizer, greedy averaged perceptron POS tagger, and the max-entropy binary named entity chunker. The use of a binary, rather than a multi-class, named entity chunker is significant. Fantasy novels often use "exotic" names for characters, we found that this often resulted in character named entities being misclassified as organisations or places. Note that this is particularly disadvantageous to the First Mentioned baseline, as any kind of named entity will steal the place. Nevertheless, it is required to ensure that all character names are a possibility to be selected.

8.4 Results and Discussion

Our evaluation results are shown in Table 8.2 for all methods. This includes the two baseline methods, and the machine learning methods with the different feature sets. We evaluate the machine learning methods using each dataset as a test set, and using each of the other two and their combination as the training set.

The First Mentioned baseline is very weak. The Most Mentioned baseline is much stronger. In almost all cases machine learning methods outperform both baselines. The results of the machine learning method on the ASOIAF and SOC are very strong. The results for WOT are weaker, though they are still accurate enough to be useful when combined with manual checking.

Test Set	Method	Train Set	Acc
ASOIAF	ML Classical Features	ASOIAF	0.980
ASOIAF	ML Word Emb. Features	ASOIAF	0.988
SOC	ML Classical Features	SOC	0.945
SOC	ML Word Emb. Features	SOC	0.956
WOT	ML Classical Features	WOT	0.785
WOT	ML Word Emb. Features	WOT	0.794

Table 8.3: The training set accuracy of the machine learning character classifier systems.

It is surprising that using the combination of two training sets does not always out-perform each on their own. For many methods training on just one dataset resulted in better results. We believe that the difference between the top result for a method and the result using the combined training sets is too small to be meaningful. It can, perhaps, be attributed to a coincidental small similarity in writing style of one of the training books to the testing book. To maximise the generalisability of the NovelPerspective prototype (see Section 8.5), we deploy models trained on all three datasets combined.

Almost all the machine learning models resulted in similarly high accuracy. The exception to this is word embedding features based model trained on SOC, which for both ASOIAF and WOT test sets performed much worse. We attribute the poor performance of these models to the small amount of training data. SOC has only 91 chapters to generate its training cases from, and the word embedding feature set has 600 dimensions. It is thus very easily to over-fit which causes these poor results.

Table 8.3 shows the training set accuracy of each machine learning model. This is a rough upper bound for the possible performance of these models on each test set, as imposed by the classifier and the feature set. The WOT bound is much lower than the other two texts. This likely relates to WOT being written in a style that closer to the line between third-person *omniscient*, than the more clear third-person *limited* POV of the other texts. We believe longer range features are required to improve the results for WOT. However, as this achieves such high accuracy for the other texts, further features would not improve accuracy significantly, without additional more difficult training data (and may cause over-fitting).

The results do not show a clear advantage to either machine learning feature set. Both the classical features and the word embeddings work well. Though, it seems that the classical feature are more robust; both with smaller training sets (like SOC), and with more difficult test sets (like WOT).

8.5 Demonstration System

The demonstration system is deployed online at `https://white.ucc.asn.au/tools/np`. A video demonstrating its use can be found at `https://youtu.be/iu41pUF4wTY`. This web-app, made using the CherryPy framework,[4] al-

[4] `http://cherrypy.org/`

lows the user to apply any of the model discussed to their own novels.

The web-app functions as shown in Figure 8.1. The user uploads an ebook, and selects one of the character classification systems that we have discussed above. They are then presented with a page displaying a list of sections, with the predicted main character(/s) paired with an excerpt from the beginning of the section. The user can adjust to show the top-k most-likely characters on this screen, to allow for additional recall.

The user can select sections to retain. They can use a regular expression to match the character names(/s) they are interested in. The sections with matching predicted character names will be selected. As none of the models is perfect, some mistakes are likely. The user can manually correct the selection before downloading the book.

8.6 Conclusion

We have presented a tool to allow consumers to restructure their ebooks around the characters they find most interesting. The system must discover the named entities that are present in each section of the book, and then classify each section as to which character's point of view the section is narrated from. For named entity detection we make use of standard tools. However, the classification is non-trivial. In this design we implemented several systems. Simply selecting the most commonly named character proved successful as a baseline approach. To improve upon this, we developed several machine learning based approaches which perform very well. While none of the classifiers are perfect, they achieve high enough accuracy to be useful.

A future version of our application will allow the users to submit corrections, giving us more training data. However, storing this information poses copyright issues that are yet to be resolved.

Acknowledgements We would like to thank Dr Gerhard Wohlgenannt (ITMO University, Saint Petersburg) for his feedback on this work just prior to submission. This research was partially funded by Australian Research Council grants DP150102405 and LP110100050.

Chapter 9

Generating Bags of Words from the Sums of their Word Embeddings

This paper was presented at the 17th Conference on Intelligent Text Processing and Computational Linguistics, in 2016. Where it received the award for best student publication.

Abstract

Many methods have been proposed to generate sentence vector representations, such as recursive neural networks, latent distributed memory models, and the simple sum of word embeddings (SOWE). However, very few methods demonstrate the ability to reverse the process – recovering sentences from sentence embeddings. Amongst the many sentence embeddings, SOWE has been shown to maintain semantic meaning, so in this paper we introduce a method for moving from the SOWE representations back to the bag of words (BOW) for the original sentences. This is a part way step towards recovering the whole sentence and has useful theoretical and practical applications of its own. This is done using a greedy algorithm to convert the vector to a bag of words. To our knowledge this is the first such work. It demonstrates qualitatively the ability to recreate the words from a large corpus based on its sentence embeddings.

As well as practical applications for allowing classical information retrieval methods to be combined with more recent methods using the sums of word embeddings, the success of this method has theoretical implications on the degree of information maintained by the sum of embeddings representation. This lends some credence to the consideration of the SOWE as a dimensionality reduced, and meaning enhanced, data manifold for the bag of words.

9.1 Introduction

The task being tackled here is the *resynthesis* of bags of words (BOW) from sentence embedding representations. In particular the generation of BOW from vectors based on the sum of the sentence's constituent words' embeddings (SOWE). To the knowledge of the authors, this task has not been attempted before.

The motivations for this task are the same as in the related area of sentence generation. **Dinu2014CompositionalGeneration** observe that given a sentence has a given meaning, and the vector encodes the same meaning, then it must be possible to translate in both directions between the natural language and the vector representation. A sub-step of this

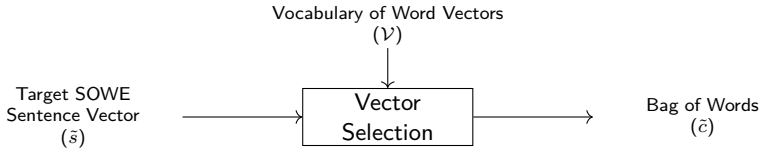


Figure 9.1: The process for the regenerating BOW from SOWE sentence embeddings.

task is the unordered case (BOW), rather than true sentences, which we tackle in this paper. The success of the implementation does indicates the validity of this dual space theory, for the representations considered (where order is neglected). There are also some potential practical applications of such an implementation, often ranging around common vector space representations.

Given suitable bidirectional methods for converting between sentence embeddings and bags of words, the sentence embedding space can be employed as a *lingua franca* for translation between various forms of information – though with loss of word order information. The most obvious of which is literal translation between different natural languages; however the use extends beyond this.

Several approaches have been developed for representing images and sentences in a common vector space. This is then used to select a suitable caption from a list of candidates (**farhadi2010every**; **socherDTRNN**). Similar methods, creating a common space between images and SOWE of the keywords describing them, could be used to generate keyword descriptions using BOW resynthesis – without any need for a list. This would allows classical word-based information retrieval and indexing techniques to be applied to images.

A similar use is the replacement of vector based extractive summarisation (**KaagebExtractiveSummaristation**; **yogatamaextractive**), with keyword based abstractive summarisation, which is the generation of a keyword summary from a document. The promising use of SOWE generation for all these applications is to have a separate model trained to take the source information (e.g. a picture for image description, or a cluster of sentences for abstract summarisation) as its input and train it to output a vector which is close to a target SOWE vector. This output can then be used to generate the sentence.

The method proposed in this paper has an input of a sum of word embeddings (SOWE) as the sentence embedding, and outputs the bag of word (BOW) which it corresponds to. The input is a vector for example $\tilde{s} = [-0.79, 1.27, 0.28, ..., -1.29]$, which approximates a SOWE vector, and outputs a BOW for example `{,: 1, best:1, it:2, of:2, the:2, times:2, was:2, worst:1}` – the BOW for the opening line of Dickens' *Tale of Two Cities*. Our method for BOW generation is shown in Figure 9.1, note that it takes as input only a word embedding vocabulary ($\mathcal{V}$) and the vector ($\tilde{s}$) to generate the BOW ($\tilde{c}$).

The rest of the paper is organized into the following sections. Section 9.2 introduces the area, discussing in general sentence models, and prior work on generation. Section 9.3 explains the problem in detail and our algorithm for solving it. Section 9.4 described the settings used for evaluation. Section 9.5 discusses the results of this evaluation. The paper

presents its conclusions in Section 9.6, including a discussion of future work.

9.2 Background

The current state of the art for full sentence generation from sentence embeddings are the works of **iyyer2014generating** and **Bowman2015SmoothGene** Both these advance beyond the earlier work of **Dinu2014CompositionalGenerati** which is only theorised to extend beyond short phrases. Iyyer et al. and Bowman et al. produce full sentences. These sentences are shown by examples to be loosely similar in meaning and structure to the original sentences. Neither works has produced quantitative evaluations, making it hard to determine between them. However, when applied to the various quantitative examples shown in both works neither is able to consistently reproduce exact matches. This motivates investigation on a simpler unordered task, converting a sum of word embeddings to bag of words, as investigated in this paper.

Bag of words is a classical natural language processing method for representing a text, sentence or document, commonly used in information retrieval. The text is represented as a multiset (or bag), this is an unordered count of how often each word occurs.

Word embeddings are vector representations of words. They have been shown to encode important syntactic and semantic properties. There are many different types of word embeddings (**Yin2015**). Two of the more notable are the SkipGrams of **mikolov2013efficient**; **mikolov2013linguisticsubs** and the Global Vector word representations (GloVe) of **pennington2014glove**. Beyond word representations are sentence embeddings.

Sentence embeddings represent sentences, which are often derived from word embeddings. Like word embeddings they can capture semantic and syntactic features. Sentence vector creation methods include the works of **le2014distributed** and **socher2014recursive**. Far simpler than those methods, is the sum of word embeddings (SOWE). SOWE, like BOW, draws significant criticism for not only disregarding sentence structure, but disregarding word order entirely when producing the sentence embedding. However, this weaknesses, may be offset by the improved discrimination allowed through words directly affecting the sentence embedding. It avoids the potential information loss through the indirection of more complex methods. Recent results suggest that this may allow it to be comparable overall to the more linguistically consistent embeddings when it comes to representing meaning.

White2015SentVecMeaning found that when classifying real-world sentences into groups of semantically equivalent paraphrases, that using SOWE as the input resulted in very accurate classifications. In that work White et al. partitioned the sentences into groups of paraphrases, then evaluated how well a linear SVM could classify unseen sentences into the class given by its meaning. They used this to evaluate a variety of different sentence embeddings techniques. They found that the classification accuracy when using SOWE as the input performed very similarly to the best performing methods – less than 0.6% worse on the harder task.

From this they concluded that the mapping from the space of sentence meaning to the vector space of the SOWE, resulted in sentences with the same meaning going to distinct areas of the vector space.

RitterPosition presented a similar task on spacial-positional meaning, which used carefully constructed artificial data, for which the meanings of the words interacted non-simply – thus theoretically favouring the more complex sentence embeddings. In their evaluation the task was classification with a Naïve Bayes classifier into one of five categories of different spatial relationships. The best of the SOWE models they evaluated, outperformed the next best model by over 5%. These results suggest this simple method is still worth consideration for many sentence embedding representation based tasks. SOWE is therefore the basis of the work presented in this paper.

9.3 The Vector Selection Problem

At the core of this problem is what we call the Vector Selection Problem, to select word embedding vectors which sum to be closest to the target SOWE (the input). The word embeddings come from a known vector vocabulary, and are to be selected with potential repetition. Selecting the vectors equates to selecting the words, because there is a one to one correspondence between the word embedding vectors and their words. This relies on no two words having exactly the same embeddings – which is true for all current word embedding techniques.

The Vector Selection Problem is defined on $(\mathcal{V}, \tilde{s}, d)$
for a finite vocabulary of vectors $\mathcal{V}$, $\mathcal{V} \subset \mathbb{R}^n$, a target sentence embedding $\tilde{s}$, $\tilde{s} \in \mathbb{R}^n$, and any distance metric d, by:

$$\underset{\left\{\forall \tilde{c} \in \mathbb{N}_0^{|\mathcal{V}|}\right\}}{\operatorname{argmin}} \; d(\tilde{s}, \sum_{\tilde{x}_j \in \mathcal{V}} \tilde{x}_j \, c_j)$$

$\tilde{x}_j$ is the vector embedding for the jth word in the vocabulary $\tilde{x}_j \in \mathcal{V}$ and c_j is the jth element of the count vector $\tilde{c}$ being optimised – it is the count of how many times the x_j occurs in approximation to the sum being assessed; and correspondingly it is the count of how many times the jth word from the vocabulary occurs in the bag of words. The selection problem is thus finding the right words with the right multiplicity, such that the sum of their vectors is as close to the input target vector, $\tilde{s}$, as possible.

9.3.1 NP-Hard Proof

The vector selection problem is NP-Hard. It is possible to reduce from any given instance of a *subset sum problem* to a vector selection problem. The *subset sum problem* is NP-complete (**karp1972reducibility**). It is defined: for some set of integers ($\mathcal{S} \subset \mathbb{Z}$), does there exist a subset ($\mathcal{L} \subseteq \mathcal{S}$) which sums to zero ($0 = \sum_{l_i \in \mathcal{L}} l_i$). A suitable metric, target vector and vocabulary of vectors corresponding to the elements $\mathcal{S}$ can be defined by a bijection; such that solving the vector selection problem will give the subset of vectors corresponding to a subset of $\mathcal{S}$ with the

smallest sum; which if zero indicates that the subset sum does exists, and if nonzero indicates that no such subset ($\mathcal{L}$) exists. A fully detailed proof of the reduction from subset sum to the vector selection problem can be found on the first author's website. [1]

9.3.2 Selection Algorithm

The algorithm proposed here to solve the selection problem is a greedy iterative process. It is a fully deterministic method, requiring no training, beyond having the word embedding mapping provided. In each iteration, first a greedy search (Greedy Addition) for a path to the targeted sum point $\tilde{s}$ is done, followed by correction through substitution (n-Substitution). This process is repeated until no change is made to the path. The majority of the selection is done in the Greedy Addition step, while the n-substitution handles fine tuning.

Greedy Addition

The greedy addition phase is characterised by adding the best vector to the bag at each step (see the pseudo-code in Algorithm 1). At each step, all the vectors in the current bag are summed, and then each vector in the vocabulary is added in turn to evaluate the new distance the new bag would have from the target, the bag which sums to be closest to the target becomes the current solution. This continues until there is no option to add any of the vectors without moving the sum away from the target. There is no bound on the size of the bag of vector (i.e. the length of the sentence) in this process, other than the greedy restriction against adding more vectors that do not get closer to the solution.

Greedy Addition works surprisingly well on its own, but it is enhanced with a fine tuning step, n-substitution, to decrease its greediness.

Data: the metric d
the target sum $\tilde{s}$
the vocabulary of vectors $\mathcal{V}$
the current best bag of vectors bag_c: initially $\emptyset$
Result: the modified bag_c which sum to be as close as greedy search can get to the target $\tilde{s}$, under the metric d

begin
  $\tilde{t} \longleftarrow \sum_{x_i \in bag_c} x_i$
  while *true* **do**
    $\tilde{x}^* \longleftarrow \operatorname{argmin}_{x_j \in \mathcal{V}} d(\tilde{s}, \tilde{t} + \tilde{x}_j)$ `/* exhaustive search of` $\mathcal{V}$ `*/`
    if $d(\tilde{s}, \tilde{t} + \tilde{x}^*) < d(\tilde{s}, \tilde{t})$ **then**
      $\tilde{t} \longleftarrow \tilde{t} + \tilde{x}^*$ $bag_c \longleftarrow bag_c \cup \{\tilde{x}^*\}$
    else
      return bag_c `/* No further improving step found */`
    end
  end
end

Algorithm 1: Greedy Addition. In practical implementation, the bag of vectors can be represented as list of indices into columns of the embedding vocabulary matrix, and efficient matrix summation methods can be used.

[1] `http://white.ucc.asn.au/publications/White2015BOWgen/`

n-Substitution

We define a new substitution based method for fine tuning solutions called n-substitution. It can be described as considering all subbags containing up to n elements, consider replacing them with a new subbag of up that size n from the vocabulary, including none at all, if that would result in the overall bag getting closer to the target $\tilde{s}$.

The reasoning behind performing the n-substitution is to correct for greedy mistakes. Consider the 1 dimensional case where $\mathcal{V} = 24, 25, 100$ and $\tilde{s} = 148$, $d(x, y) = |x - y|$. Greedy addition would give $bag_c = [100, 25, 24]$ for a distance of 1, but a perfect solution is $bag_c = [100, 24, 24]$ which is found using 1-substitution. This substitution method can be considered as re-evaluating past decisions in light of the future decisions. In this way it lessens the greed of the addition step.

The n-substitution phase has time complexity of $O(\binom{C}{n}V^n)$, for $C = \sum \tilde{c}$ i.e. current cardinality of bag_c. With large vocabularies it is only practical to consider 1-substitution. With the Brown Corpus, where $|\mathcal{V}| \approxeq 40,000$, it was found that 1-substitution provides a significant improvement over greedy addition alone. On a smaller trial corpora, where $|\mathcal{V}| \approxeq 1,000$, 2-substitution was used and found to give further improvement. In general it is possible to initially use 1-substitution, and if the overall algorithm converges to a poor solution (given the distance to the target is always known), then the selection algorithm can be retried from the converged solution, using 2-substitution and so forth. As n increases the greed overall decreases; at the limit the selection is not greedy at all, but is rather an exhaustive search.

9.4 Experimental Setup and Evaluations

9.4.1 Word Embeddings

GloVe representations of words (**pennington2014glove**) are used in our evaluations. There are many varieties of word embeddings which work with our algorithm. GloVe was chosen simply because of the availability of a large pre-trained vocabulary of vectors. The representations used for evaluation were pretrained on 2014 Wikipedia and Gigaword 5[2]. Preliminary results with SkipGrams from **mikolov2013efficient** suggested similar performance.

9.4.2 Corpora

The evaluation was performed on the Brown Corpus (**francis1979brown**) and on a subset of the Books Corpus (**moviebook**). The Brown Corpus was sourced with samples from a 500 fictional and non-fictional works from 1961. The Books Corpus was sourced from 11,038 unpublished novels. The Books Corpus is extremely large, containing roughly 74 million sentences. After preprocessing we randomly selected 0.1% of these for evaluation.

[2]Kindly made available online at `http://nlp.stanford.edu/projects/glove/`

For simplicity of evaluation, sentences containing words not found in the pretrained vector vocabulary are excluded. These were generally rare mis-spellings and unique numbers (such as serial numbers). Similarly, words which are not used in the corpus are excluded from the vector vocabulary.

After the preprocessing the final corpora can be described as follows. The Brown Corpus has 42,004 sentences and a vocabulary of 40,485 words. Where-as, the Books Corpus has 66,464 sentences, and a vocabulary of 178,694 words. The vocabulary sizes are beyond what is suggested as necessary for most uses (**nation2006large**). These corpora remain sufficiently large and complex to quantitatively evaluate the algorithm.

9.4.3 Vector Selection

The Euclidean metric was used to measure how close potential solutions were to the target vector. The choice of distance metric controls the ranking of each vector by how close (or not) it brings the the partial sum to the target SOWE during the greedy selection process. Preliminary results on one-tenth of the Books Corpus used in the main evaluation found the Manhattan distance performed marginally worse than the Euclidean metric and took significantly longer to converge.

The commonly used cosine similarity, or the linked angular distance, have an issue of zero distances between distinct points – making them not true distance metrics. For example the SOWE of *"a can can can a can"* has a zero distance under those measures to the SOWE for *"a can can"*.[3] That example is a pathological, though valid sentence fragment. True metrics such as the Euclidean metric do not have this problem. Further investigation may find other better distance metrics for this step.

The Julia programming language (**Julia**), was used to create the implementation of the method, and the evaluation scripts for the results presented in the next section. This implementation, evaluation scripts, and the raw results are available online.[4]. Evaluation was carried out in parallel on a 12 core virtual machine, with 45Gb of RAM. Sufficient RAM is required to load the entire vector vocabulary in memory.

9.5 Results and Discussion

Table 9.1 shows examples of the output. Eight sentences which were used for demonstration of sentence generation in **iyyer2014generating**; **Bowman2015SmoothGeneration** have the BOW generation results shown. All examples except *(a)* and *(f)* are perfect. Example *(f)* is interesting as it seems that the contraction token *'re* was substituted for *are*, and *do* for *doing*. Inspections of the execution logs for running on the examples show that this was a greedy mistake that would be corrected using 2-substitution. Example *a* has many more mistakes.

[3] The same is true for any number of repetitions of the word *buffalo* – each of which forms a valid sentence as noted in **tymoczko1995sweet**

[4] `http://www.cicling.org/2016/data/97`

Table 9.1: Examples of the BOW Produced by our method using the Books Corpus vocabulary, compared to the Correct BOW from the reference sentences. The P and C columns show the the number of occurrences of each word in the Produced and Correct bags of words, respectively. **Bolded** lines highlight mistakes. Examples a-e were sourced from **iyyer2014generating**, Examples f-h from **Bowman2015SmoothGeneration**. Note that in example a, the *"___..._(n)"* represents n repeated underscores (without spaces).

(a) ralph waldo emerson dismissed this poet as the jingle man and james russell lowell called him three-fifths genius and two-fifths sheer fudge

Word	P	C
2008	**1**	**0**
___..._*(13)*	**1**	**0**
___..._*(34)*	**1**	**0**
___..._*(44)*	**1**	**0**
“	**1**	**0**
aldrick	**1**	**0**
and	2	2
as	**0**	**1**
both	**1**	**0**
called	**0**	**1**
dismissed	1	1
emerson	1	1
fudge	1	1
genius	1	1
hapless	**1**	**0**
him	1	1
hirsute	**1**	**0**
james	1	1
jingle	1	1
known	**1**	**0**
lowell	1	1
man	**0**	**1**
poet	1	1
ralph	1	1
russell	1	1
sheer	1	1
the	1	1
this	1	1
three-fifths	1	1
two-fifths	1	1
waldo	1	1
was	**1**	**0**

(b) thus she leaves her husband and child for aleksei vronsky but all ends sadly when she leaps in front of a train

Word	P	C
a	1	1
aleksei	1	1
all	1	1
and	1	1
but	1	1
child	1	1
ends	1	1
for	1	1
front	1	1
her	1	1
husband	1	1
in	1	1
leaps	1	1
leaves	1	1
of	1	1
sadly	1	1
she	2	2
thus	1	1
train	1	1
vronsky	1	1
when	1	1

(c) name this 1922 novel about leopold bloom written by james joyce

Word	P	C
1922	1	1
about	1	1
bloom	1	1
by	1	1
james	1	1
joyce	1	1
leopold	1	1
name	1	1
novel	1	1
this	1	1
written	1	1

(d) this is the basis of a comedy of manners first performed in 1892

Word	P	C
1892	1	1
a	1	1
basis	1	1
comedy	1	1
first	1	1
in	1	1
is	1	1
manners	1	1
of	2	2
performed	1	1
the	1	1
this	1	1

(e) in a third novel a sailor abandons the patna and meets marlow who in another novel meets kurtz in the congo

Word	P	C
a	2	2
abandons	1	1
and	1	1
another	1	1
congo	1	1
in	3	3
kurtz	1	1
marlow	1	1
meets	2	2
novel	2	2
patna	1	1
sailor	1	1
the	2	2
third	1	1
who	1	1

(f) how are you doing ?

Word	P	C
’re	**1**	**0**
?	1	1
are	**0**	**1**
do	**1**	**0**
doing	**0**	**1**
how	1	1
well	**1**	**0**
you	**0**	**1**

(g) we looked out at the setting sun .

Word	P	C
.	1	1
at	1	1
looked	1	1
out	1	1
setting	1	1
sun	1	1
the	1	1
we	1	1

(h) i went to the kitchen .

Word	P	C
.	1	1
i	1	1
kitchen	1	1
the	1	1
to	1	1
went	1	1

Table 9.2: The performance of the BOW generation method. Note the final line is for the Books Corpus, where-as the preceding are or the Brown Corpus.

Corpus	Embedding Dimensions	Portion Perfect	Mean Jaccard Score	Mean Precision	Mean Recall	Mean F1 Score
Brown	50	6.3%	0.175	0.242	0.274	0.265
Brown	100	19.4%	0.374	0.440	0.530	0.477
Brown	200	44.7%	0.639	0.695	0.753	0.720
Brown	300	70.4%	0.831	0.864	0.891	0.876
Books	300	75.6%	0.891	0.912	0.937	0.923

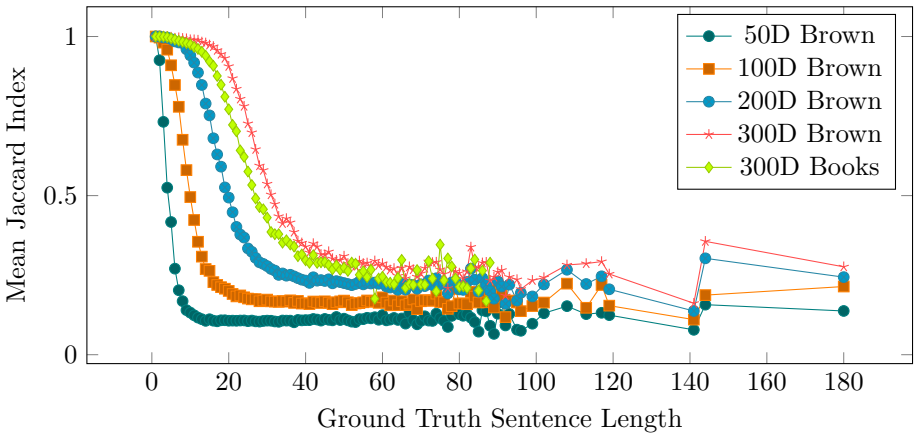


Figure 9.2: The mean Jaccard index achieved during the word selection step, shown against the ground truth length of the sentence. Note that the vast majority of sentences are in the far left end of the plot. The diminishing samples are also the cause of the roughness, as the sentence length increases.

The mistakes in Example *(a)* seem to be related to unusual nonword tokens, such as the three tokens with 13, 34, and 44 repetitions of the underscore character. These tokens appear in the very large Books corpus, and in the Wikipedia/Gigaword pretraining data used for word embeddings, but are generally devoid of meaning and are used as structural elements for formatting. We theorise that because of their rarity in the pre-training data they are assigned an unusual word-embedding by GloVE. There occurrence in this example suggests that better results may be obtained by pruning the vocabulary. Either manually, or via a minimum uni-gram frequency requirement. The examples overall highlight the generally high performance of the method, and evaluations on the full corpora confirm this.

Table 9.2 shows the quantitative performance of our method across both corpora. Five measures are reported. The most clear is the portion of exact matches – this is how often out of all the trials the method produced the exact correct bag of words. The remaining measures are all means across all the values of the measures in each trial. The Jaccard index is the portion of overlap between the reference BOW, and the output BOW – it is the cardinality of the intersection divided by that of the union. The precision is the portion of the output words that were correct; and the recall is the portion of all correct words which were output. For precision and recall word repetitions were treated as distinct. The F_1 score is the harmonic mean of precision and recall. The recall is higher than the precision, indicating that the method is more prone to producing additional incorrect words (lowering the precision), than to missing words out (which would lower the recall).

Initial investigation focused on the relationship between the number of dimensions in the word embedding and the performance. This was carried out on the smaller Brown corpus. Results confirmed the expectation that higher dimensional embeddings allow for better generation of words. The best performing embedding size (i.e. the largest) was then used to evaluate success on the Books Corpus. The increased accuracy when using higher dimensionality embeddings remains true at all sentence lengths.

As can be seen in Figure 9.2 sentence length is a very significant factor in the performance of our method. As the sentences increase in length, the number of mistakes increases. However, at higher embedding dimensionality the accuracy for most sentences is high. This is because most sentences are short. The third quartile on sentence length is 25 words for Brown, and 17 for the Books Corpus. This distribution difference is also responsible for the apparent better results on the Books Corpus, than on the Brown corpus.

While the results shown in Table 9.2 suggest that on the Books corpus the algorithm performs better, this is due to its much shorter average sentence length. When taken as a function of the sentence length, as shown in Figure 9.2, performance on the Books Corpus is worse than on the Brown Corpus. It can be concluded from this observation that increasing the size of the vocabulary does decrease success in BOW regeneration. Books Corpus vocabulary being over four times larger, while the other factors remained the same, resulted in lower performance. However, when taking all three factors into account, we note that increasing the *vocabulary size* has significantly less impact than increasing the *sentence length* or the *embedding dimensionality* on the performance.

9.6 Conclusion

A method was presented for how to regenerate a bag of words, from the sum of a sentence's word embeddings. This problem is NP-Hard. A greedy algorithm was found to perform well at the task, particularly for shorter sentences when high dimensional embeddings are used.

Resynthesis degraded as sentence length increased, but remained strong with higher dimensional models up to reasonable length. It also decreased as the vocabulary size increased, but significantly less so. The BOW generation method is functional with usefully large sentences and vocabulary.

From a theoretical basis the resolvability of the selection problem shows that adding up the word embeddings does preserve the information on which words were used; particularly for higher dimensional embeddings. This shows that collisions do not occur (at least not frequently) such that two unrelated sentences do not end up with the same SOWE representation.

This work did not investigate the performance under noisy input SOWEs – which occur in many potential applications. Noise may cause the input to better align with an unusual sum of word embeddings, than with its true value. For example it may be shifted to be very close a sentence embedding that is the sum of several hundred word embeddings. Investigating, and solving this may be required for applied uses of any technique that solves the vector selection problem.

More generally, future work in this area would be to use a stochastic language model to suggest suitable orderings for the bags of words. While this would not guarantee correct ordering every-time, we speculate that it could be used to find reasonable approximations often. Thus allowing this

bag of words generation method to be used for full sentence generation, opening up a much wider range of applications.

Acknowledgements This research is supported by the Australian Postgraduate Award, and partially funded by Australian Research Council grants DP150102405 and LP110100050. Computational resources were provided by the National eResearch Collaboration Tools and Resources project (Nectar).

Chapter 10

Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem

This paper was presented at the High Dimensional Data Mining Workshop at the IEEE International Conference on Data Mining, in 2016.

Abstract

Converting a sentence to a meaningful vector representation has uses in many NLP tasks, however very few methods allow that representation to be restored to a human readable sentence. Being able to generate sentences from the vector representations demonstrates the level of information maintained by the embedding representation – in this case a simple sum of word embeddings. We introduce such a method for moving from this vector representation back to the original sentences. This is done using a two stage process; first a greedy algorithm is utilised to convert the vector to a bag of words, and second a simple probabilistic language model is used to order the words to get back the sentence. To the best of our knowledge this is the first work to demonstrate quantitatively the ability to reproduce text from a large corpus based directly on its sentence embeddings.

10.1 Introduction

Generally sentence generation is the main task of the more broad natural language generation field; here we use the term only in the context of sentence generation from sentence vector representation. For our purposes, a sentence generation method has as its input a sentence embedding, and outputs the sentence which it corresponds to. The input is a vector, for example $\tilde{s} = [0.11, 0.57, -0.21, ..., 1.29]$, and the output is a sentence, for example "The boy was happy.".

Dinu2014CompositionalGeneration motivates this work from a theoretical perspective given that a sentence encodes its meaning, and the vector encodes the same meaning, then it must be possible to translate

in both directions between the natural language and the vector representation. In this paper, we present an implementation that indicates to some extent the equivalence between the natural language space and the sum of word embeddings (SOWE) vector representation space. This equivalence is shown by demonstrating a lower bound on the capacity of the vector representation to be used for sentence generation.

The current state of the art methods for sentence generation produce human readable sentences which are rough approximations of the intended sentence. These existing works are those of **iyyer2014generating** and **Bowman2015SmoothGeneration**. Both these have been demonstrated to produce full sentences. These sentences are qualitatively shown to be loosely similar in meaning to the original sentences. Neither work has produced quantitative evaluations, making it hard to compare their performance. Both are detailed further in Section 10.2. Both these methods use encoder/decoder models trained through machine learning; we present here a more deterministic algorithmic approach, but restrict the input sentence vector to be the non-compositional sum of word embeddings representation.

RitterPosition and **White2015SentVecMeaning** found that when classifying sentences into categories according to meaning, simple SOWE outperformed more complex sentence vector models. Both works used sentence embeddings as the input to classifiers. **RitterPosition** classified challenging artificial sentences into categories based on the positional relationship described using Naïve Bayes. **White2015SentVecMeaning** classified real-world sentences into groups of semantically equivalent paraphrases. In the case of **RitterPosition** this outperformed the next best representation by over 5%. In the case of **White2015SentVecMeaning** it was within a margin of 1% from the very best performing method. These results suggest that there is high consistency in the relationship between a point in the SOWE space, and the meaning of the sentence.

wieting2015towards presented a sentence embedding based on the related average of word-embedding, showing excellent performance across several competitive tasks. They compared their method's performance against several models, including recurrent neural networks, and long short term memory (LSTM) architectures. It was found that their averaging method outperformed the more complex LSTM system, on most sentence similarity and entailment task. Thus these simple methods are worth further consideration. SOWE is the basis of the work presented in this paper.

Our method performs the sentence generation in two steps, as shown in Figure 10.1. It combines the work of **White2015BOWgen** on generating bags of words (BOW) from sums of word embeddings (SOWE); with the work of **Horvat2014** on ordering BOW into sentences. The overall approach, of word selection followed by word ordering, can be used to generate proper sentences from SOWE vectors.

The rest of the paper is organized into the following sections. Section 10.2 discusses the prior work on sentence generation. Section 10.3 explains the problem in detail and how our method is used to solve it. Section 10.4 describes the settings used for evaluation. Section 10.5 presents the results of this evaluation. The paper concludes with Section 10.6 and a

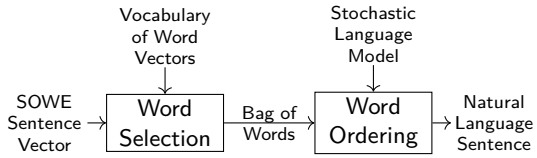


Figure 10.1: The Sel. BOW+Ord. process for the regenerating sentences from SOWE-type sentence vectors.

discussion of future work on this problem.

10.2 Related Works

To the best of our knowledge only three prior works exist in the area of sentence generation from embeddings. The first two (**Dinu2014CompositionalGe** **iyyer2014generating**) are based on the recursive structures in language, while **Bowman2015SmoothGeneration**, uses the sequential structure.

Dinu2014CompositionalGeneration extends the models described by **zanzotto2010estimating** and **Guevara2010** for generation. The composition is described as a linear transformation of the input word embeddings to get an output vector, and another linear transformation to reverse the composition reconstructing the input. The linear transformation matrices are solved using least squares regression. This method of composing, can be applied recursively from words to phrases to clauses and so forth. It theoretically generalises to whole sentences, by recursively applying the composition or decomposition functions. However, Dinu and Baroni's work is quantitatively assessed only on direct reconstruction for decomposing Preposition-Noun and Adjective-Noun word phrases. In these cases where the decomposition function was trained directly on vectors generated using the dual composition function they were able to get perfect reconstruction on the word embedding based inputs.

iyyer2014generating extends the work of **SocherEtAl2011:PoolRAE** defining an unfolding recursive dependency-tree recursive autoencoder (DT-RAE). Recursive neural networks are jointly trained for both composing the sentence's words into a vector, and for decomposing that vector into words. This composition and decomposition is done by reusing a composition neural network at each vertex of the dependency tree structure, with different weight matrices for each dependency relation. The total network is trained based on the accuracy of reproducing its input word embeddings. It can be used to generate sentences, if a dependency tree structure for the output is provided. This method was demonstrated quantitatively on five examples; the generated sentences were shown to be loosely semantically similar to the originals.

Bowman2015SmoothGeneration uses a a modification of the variational autoencoder (VAE) (**2014VAE**) with natural language inputs and outputs, to learn the sentence representations. These input and output stages are performed using long short-term memory recurrent neural networks (**hochreiter1997long**). They demonstrate a number of uses of this technique, one of which is sentence generation, in the sense of

this paper. While Bowman et al. do define a generative model, they do not seek to recreate a sentence purely from its vector input, but rather to produce a series of probability distributions on the words in the sentence. These distributions can be evaluated greedily, which the authors used to give three short examples of resynthesis. They found the sentence embeddings created captured largely syntactic and loose topical information.

We note that none of the aforementioned works present any quantitative evaluations on a corpus of full sentences. We suggest that that is due to difficulties in evaluation. As noted in **iyyer2014generating** and **Bowman2015SmoothGeneration**, they tend to output lose paraphrases, or roughly similar sentences. This itself is a separately useful achievement to pure exact sentence generation; but it is not one that allows ready interpretation of how much information is maintained by the embeddings. Demonstration of our method at generating the example sentences used in those work is available as supplementary material[1]. As our method often can exactly recreate the original sentence from its vector representation evaluation is simpler.

Unlike current sentence generation methods, the non-compositional BOW generation method of **White2015BOWgen** generally outputs a BOW very close to the reference for that sentence – albeit at the cost of losing all word order information. It is because of this accuracy that we base our proposed sentence generation method on it (as detailed in Section 10.3.1). The word selection step we used is directly based on their greedy BOW generation method. We improve it for sentence generation by composing with a word ordering step to create the sentence generation process.

10.3 General Framework

As discussed in Section 10.1, and shown in Figure 10.1, the approach taken to generate the sentences from the vectors comes in two steps. First selecting the words used – this is done deterministically, based on a search of the embedding space. Second is to order them, which we solve by finding the most likely sequence according to a stochastic language model. Unlike the existing methods, this is a deterministic approach, rather than a machine learn method. The two subproblems which result from this split resemble more classical NP-Hard computer science problems; thus variations on known techniques can be used to solve them.

10.3.1 Word Selection

White2015BOWgen approaches the BOW generation problem, as task of selecting the vectors that sum to be closest to a given vector. This is related to the knapsack and subset sum problems. They formally define

[1] `http://white.ucc.asn.au/publications/White2016SOWE2Sent/`

the vector selection problem as:

$$(\tilde{s}, \mathcal{V}, d) \mapsto \underset{\left\{\forall \tilde{c} \in \mathbb{N}_0^{|\mathcal{V}|}\right\}}{\operatorname{argmin}} d(\tilde{s}, \sum_{\tilde{x}_j \in \mathcal{V}} \tilde{x}_j c_j)$$

to find the bag of vectors selected from the vocabulary set $\mathcal{V}$ which when summed is closest to the target vector $\tilde{s}$. Closeness is assessed with distance metric d. $\tilde{c}$ is the indicator function for that multi-set of vectors. As there is a one to one correspondence between word embeddings and their words, finding the vectors results in finding the words. **White2015BOWgen** propose a greedy solution to the problem[2].

The key algorithm proposed by **White2015BOWgen** is greedy addition. The idea is to greedily add vectors to a partial solution building towards a complete bag. This starts with an empty bag of word embeddings, and at each step the embedding space is searched for the vector which when added to the current partial solution results in the minimal distance to the target – when compared to other vectors from the vocabulary. This step is repeated until there are no vectors in the vocabulary that can be added without moving away from the solution. Then a fine-tuning step, n-substitution, is used to remove some simpler greedy mistakes.

The n-substitution step examines partial solutions (bags of vectors) and evaluates if it is possible to find a better solution by removing n elements and replacing them with up-to n different elements. The replacement search is exhaustive over the n-ary Cartesian product of the vocabulary. Only for $n = 1$ is it currently feasible for practical implementation outside of highly restricted vocabularies. Never-the-less even 1-substitution can be seen as lessening the greed of the algorithm, through allowing early decisions to be reconsidered in the full context of the partial solution. The algorithm does remain greedy, but many simple mistakes are avoided by n-substitution. The greedy addition and n-substitution processes are repeated until the solution converges.

10.3.2 The Ordering Problem

After the bag of words has been generated by the previous step, it must be ordered (sometimes called linearized). For example "are how , today hello ? you", is to be ordered into the sentence: "hello , how are you today ?". This problem cannot always be solved to a single correct solution. **Mitchell2008** gives the example of "It was not the sales manager who hit the bottle that day, but the office worker with the serious drinking problem." which has the same word content (though not punctuation) as "That day the office manager, who was drinking, hit the problem sales worker with a bottle, but it was not serious.". However, while a unique ordering cannot be guaranteed, finding the most likely word ordering is possible. There are several current methods for word ordering

To order the words we use a method based on the work of **Horvat2014**, which uses simple trigrams. More recent works, such as beam-search and

[2]We also investigated beam search as a possible improvement over the greedy addition and n-substitution used by **White2015BOWgen**, but did not find significant improvement. The additional points considered by the beam tended to be words that would be chosen by the greedy addition in the later steps – thus few alternatives where found.

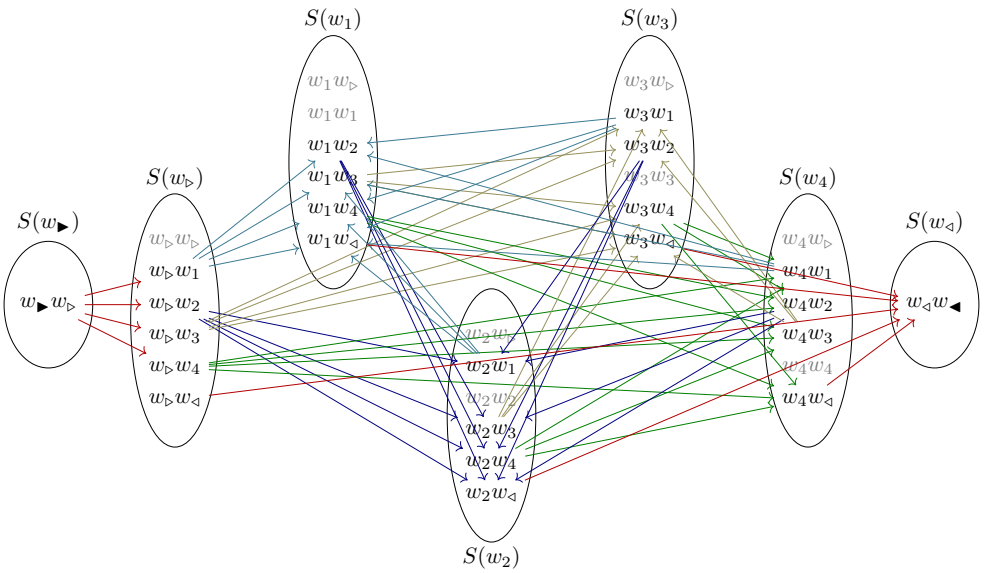


Figure 10.2: A graph showing the legal transitions between states, when the word-ordering problem is expressed similar to a GA-TSP. Each edge $\langle w_a, w_b\rangle \rightarrow \langle w_c, w_d\rangle$ has cost $-\log(P(w_c \,|\, w_a w_b)$. The nodes are grouped into districts (words). Nodes for invalid states are greyed out.

LSTM language model and proposed by **2016arXiv160408633S**; or a syntactic rules based method such as presented in **ZhangWordOrderSyntax**, could be used. These more powerful ordering methods internalise significant information about the language. The classical trigram language model we present is a clearer baseline for the capacity to regenerate the sentences; which then be improved by using such systems.

Horvat2014 formulated the word ordering problem as a generalised asymmetrical travelling salesman problem (GA-TSP). Figure 10.2 shows an example of the connected graph for ordering five words. We extend beyond the approach of **Horvat2014** by reformulating the problem as a linear mixed integer programming problem (MIP). This allows us to take advantage of existing efficient solvers for this problem. Beyond the GA-TSP approach, a direct MIP formulation allows for increased descriptive flexibility and opens the way for further enhancement. Some of the constraints of a GA-TSP can be removed, or simplified in the direct MIP formulation for word ordering. For example, word ordering does have distinct and known start and end nodes (as shall be detailed in the next section). To formulate it as a GA-TSP it must be a tour without beginning or end. **Horvat2014** solve this by simply connecting the start to the end with a zero cost link. This is not needed if formulating this as a MIP problem, the start and end nodes can be treated as special cases. Being able to special case them as nodes known always to occur allows some simplification in the subtour elimination step. The formulation to mixed integer programming is otherwise reasonably standard.

Notation

We will write w_i to represent a word from the bag $\mathcal{W}$ ($w_i \in \mathcal{W}$), with arbitrarily assigned unique subscripts. Where a word occurs with multiplicity greater than 1, it is assigned multiple subscripts, and is henceforth

treated as a distinct word.

Each vertex is a sequence of two words, $\langle w_i, w_j \rangle \in \mathcal{W}^2$. This is a Markov state, consisting of a word w_j and its predecessor word w_i – a bigram.

Each edge between two vertices represents a transition from one state to another which forms a trigram. The start vertex is given by $\langle w_{\blacktriangleright}, w_{\triangleright} \rangle$, and the end by $\langle w_{\triangleleft}, w_{\blacktriangleleft} \rangle$. The pseudowords $w_{\blacktriangleright}, w_{\triangleright}, w_{\triangleleft}, w_{\blacktriangleleft}$ are added during the trigram models' training allowing knowledge about the beginning and ending of sentences to be incorporated.

The GA-TSP districts are given by the sets of all states that have a given word in the first position. The district for word w_i is given by $S(w_i) \subseteq \mathcal{W}^2$, defined as $S(w_i) = \{\langle w_i, w_j \rangle \mid \forall w_j \in \mathcal{W}\}$. It is required to visit every district, thus it is required to use every word. With this description, the problem can be formulated as a MIP optimisation problem.

Optimization Model

Every MIP problem has a set of variables to optimise, and a cost function that assesses how optimal a given choice of values for that variable is. The cost function for the word ordering problem must represent how unlikely a particular order is. The variables must represent the order taken. The variables are considered as a table (τ) which indicates if a particular transition between states is taken. Note that for any pair of Markov states $\langle w_a, w_b \rangle, \langle w_c, w_d \rangle$ is legal if and only if $b = c$, so we denote legal transitions as $\langle w_i, w_j \rangle \to \langle w_j, w_k \rangle$. Such a transition has cost:

$$C[\langle w_i, w_j \rangle, \langle w_j, w_k \rangle] = -\log\left(P(w_k | w_i, w_j \rangle\right)$$

The table of transitions to be optimized is:

$$\tau[\langle w_i, w_j \rangle, \langle w_j, w_k \rangle] = \begin{cases} 1 & \text{if transition from} \\ & \langle w_i, w_j \rangle \to \langle w_j, w_k \rangle \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

The total cost to be minimized, is given by

$$C_{total}(\tau) = \sum_{\forall w_i, w_j, w_k \in \mathcal{W}^3} \tau[\langle w_i, w_j \rangle, \langle w_j, w_k \rangle] \cdot C[\langle w_i, w_j \rangle, \langle w_j, w_k \rangle]$$

The probability of a particular path (i.e. of a particular ordering) is thus given by $P(\tau) = e^{-C_{total}(\tau)}$

The word order can be found by following the links. The function $f_\tau(n)$ gives the word that, according to τ occurs in the nth position.

$$f_\tau(1) = \{w_a \mid w_a \in \mathcal{W} \wedge \tau[\langle w_{\blacktriangleright}, w_{\triangleright} \rangle, \langle w_{\triangleright}, w_a \rangle] = 1\}_1$$

$$f_\tau(2) = \{w_b \mid w_b \in \mathcal{W} \wedge \tau[\langle w_{\triangleright}, f_\tau(1) \rangle, \langle f_\tau(1), w_b \rangle] = 1\}_1$$

$$f_\tau(n) = \underset{\text{when } n \geq 3}{\{w_c \mid w_c \in \mathcal{W} \wedge \tau[\langle f_\tau(n-2), f_\tau(n-1) \rangle, \langle f_\tau(n-1), w_c \rangle] = 1\}_1}$$

The notation $\{\cdot\}_1$ indicates taking a singleton set's only element. The constraints on τ ensure that each set is a singleton.

Constraints

The requirements of the problem, place various constraints on to τ: The Markov state must be maintained: $\forall\langle w_a, w_b\rangle, \langle w_c, w_d\rangle \in \mathcal{W}^2$:

$$w_b \neq w_c \implies \tau[\langle w_a, w_b\rangle, \langle w_c, w_d\rangle] = 0$$

Every node entered must also be exited – except those at the beginning and end.
$\forall\langle w_i, w_j\rangle \in \mathcal{W}^2 \backslash \{\langle w_\blacktriangleright, w_\triangleright\rangle, \langle w_\triangleleft, w_\blacktriangleleft\rangle\}$:

$$\sum_{\forall\langle w_a, w_b\rangle \in \mathcal{W}^2} \tau[\langle w_a, w_b\rangle, \langle w_i, w_j\rangle] = \sum_{\forall\langle w_c, w_d\rangle \in \mathcal{W}^2} \tau[\langle w_i, w_j\rangle, \langle w_c, w_d\rangle]$$

Every district must be entered exactly once. i.e. every word must be placed in a single position in the sequence. $\forall w_i \in \mathcal{W} \backslash \{w_\blacktriangleright, w_\blacktriangleleft\}$:

$$\sum_{\forall\langle w_i, w_j\rangle \in S(w_i)} \sum_{\forall\langle w_a, w_b\rangle \in \mathcal{W}^2} \tau[\langle w_a, w_b\rangle, \langle w_i, w_j\rangle] = 1$$

To allow the feasibility checker to detect if ordering the words is impossible, transitions of zero probability are also forbidden. i.e. if $P(w_n | w_{n-2}, w_{n-1}) = 0$ then $\tau[\langle w_{n-2}, w_{n-1}\rangle, \langle w_{n-1}, w_n\rangle] = 0$. These transitions, if not expressly forbidden, would never occur in an optimal solution in any case, as they have infinitely high cost.

Lazy Subtour Elimination Constraints The problem as formulated above can be input into a MIPS solver. However, like similar formulations of the travelling salesman problem, some solutions will have subtours. As is usual callbacks are used to impose lazy constraints to forbid such solutions at run-time. However, the actual formulation of those constraints are different from a typical GA-TSP.

Given a potential solution τ meeting all other constraints, we proceed as follows.

The core path – which starts at $\langle w_\blacktriangleright, w_\triangleright\rangle$ and ends at $\langle w_\triangleleft, w_\blacktriangleleft\rangle$ can be found. This is done by practically following the links from the start node, and accumulating them into a set $T \subseteq \mathcal{W}^2$

From the core path, the set of words covered is given by $\mathcal{W}_T = \{w_i \mid \forall\langle w_i, w_j\rangle \in T\} \cup \{w_\blacktriangleleft\}$. If $\mathcal{W}_\mathcal{T} = \mathcal{W}$ then there are no subtours and the core path is the complete path. Otherwise, there is a subtour to be eliminated.

If there is a subtour, then a constraint must be added to eliminate it. The constraint we define is that there must be a connection from at least one of the nodes in the district covered by the core path to one of the nodes in the districts not covered.

The districts covered by the tour are given by $S_T = \bigcup_{w_t \in \mathcal{W}_T} S(w_t)$. The subtour elimination constraint is given by

$$\sum_{\forall\langle w_{t1}, w_{t2}\rangle \in S_T} \sum_{\forall\langle w_a, w_b\rangle \in \mathcal{W}^2 \backslash S_T} \tau[\langle w_{t1}, w_{t2}\rangle, \langle w_a, w_b\rangle] \geq 1$$

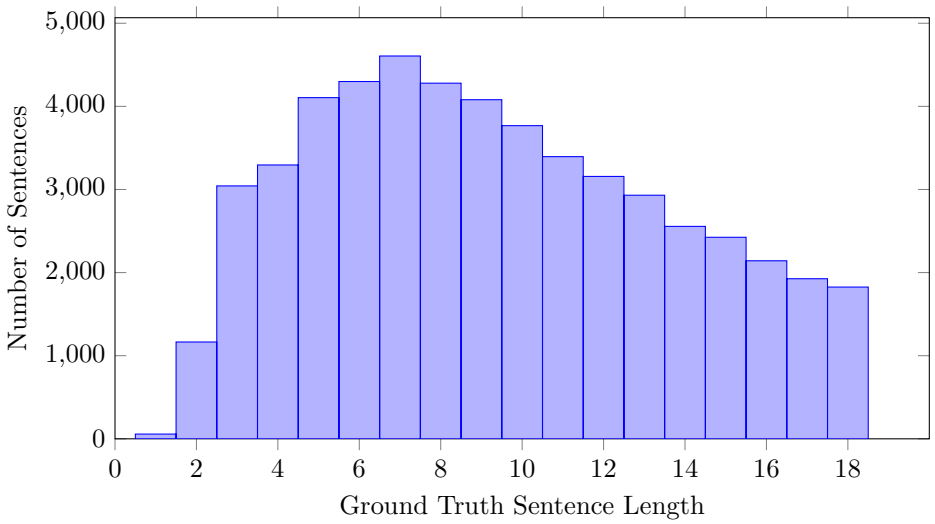


Figure 10.3: The distribution of the evaluation corpus after preprocessing.

i.e. there must be a transition from one of the states featuring a word that is in the core path, to one of the states featuring a word not covered by the core path.

This formulation around the notion of a core path that makes this different from typical subtour elimination in a GA-TSP. GA-TSP problems are not generally guaranteed to have any nodes which must occur. However, every word ordering problem is guaranteed to have such a node – the start and end nodes. Being able to identify the core path allows for reasonably simple subtour elimination constraint definition. Other subtour elimination constraints, however, also do exist.

10.4 Experimental Setup and Evaluations

This experimental data used in this evaluation was obtained from the data released with **White2015BOWgen**.[3]

10.4.1 Word Embeddings

GloVe representations of words are used in our evaluations (**pennington2014glove**). GloVe was chosen because of the availability of a large pre-trained vocabulary of vectors.[4] The representations used for evaluation were pretrained on the 2014 Wikipedia and Gigaword 5. Other vector representations are presumed to function similarly. **White2015BOWgen** showed that their word selection method significantly improves with higher dimensional embeddings. Due to their findings, we only evaluated 300 dimensional embeddings.

Process	Perfect Sentences	BLEU Score	Portion Feasible
Ref. BOW+Ord.	66.6%	0.806	99.6%
Sel. BOW+Ord.	62.2%	0.745	93.7%

Table 10.1: The overall performance of the Sel. BOW+Ord. sentence generation process when evaluated on the Books corpus.

10.4.2 Corpus and Language Modelling

The evaluation was performed on a subset of the Books Corpus (**moviebook**). The corpus was preprocessed as in the work of **White2015BOWgen**. This meant removing any sentences which used words not found in the embedding vocabulary.

After preprocessing, the base corpus, was split 90:10. 90% (59,694,016 sentences) of the corpus was used to fit a trigram model. This trigram language model was smoothed using the Knesler-Ney back-off method (**kneser1995improved**). The remaining 10% of the corpus was kept in reserve. From the 10%, 1% (66,464 sentences) were taken for testing. From this any sentences with length over 18 words were discarded – the time taken to evaluate longer sentences increases exponentially and becomes infeasible. This left a final test set of 53,055 sentences. Figure 10.3 shows the distribution of the evaluation corpus in terms of sentence length.

Note that the Books corpus contains many duplicate common sentences, as well as many duplicate books: according to the distribution site[5] only 7,087 out of 11,038 original books in the corpus are unique. We did not remove any further duplicates, which means there is a strong chance of a small overlap between the test set, and the set used to fit the trigrams.

10.4.3 Mixed Integer Programming

Gurobi version 6.5.0 was used to solve the MIP problems, invoked though the JuMP library (**jump**). During preliminary testing we found Gurobi to be significantly faster than the open source GLTK. Particularly for longer sentences, we found two orders of magnitude difference in speed for sentences of length 18. This is inline with the more extensive evaluations of **meindl2012analysis**. Gurobi was run under default settings, other than being restricted to a single thread. Restricting the solver to a single thread allowed for parallel processing.

Implementation was in the Julia programming language (**Julia**). The implementation, and non-summarised results are available for download.[6]

Process	Perfect BOWs	Mean Precision	Mean Jaccard Index
Sel. BOW (only)	75.6%	0.912	0.891

Table 10.2: The performance of the word selection step, on the Books corpus. This table shows a subset of the results reported by **White2015BOWgen**.

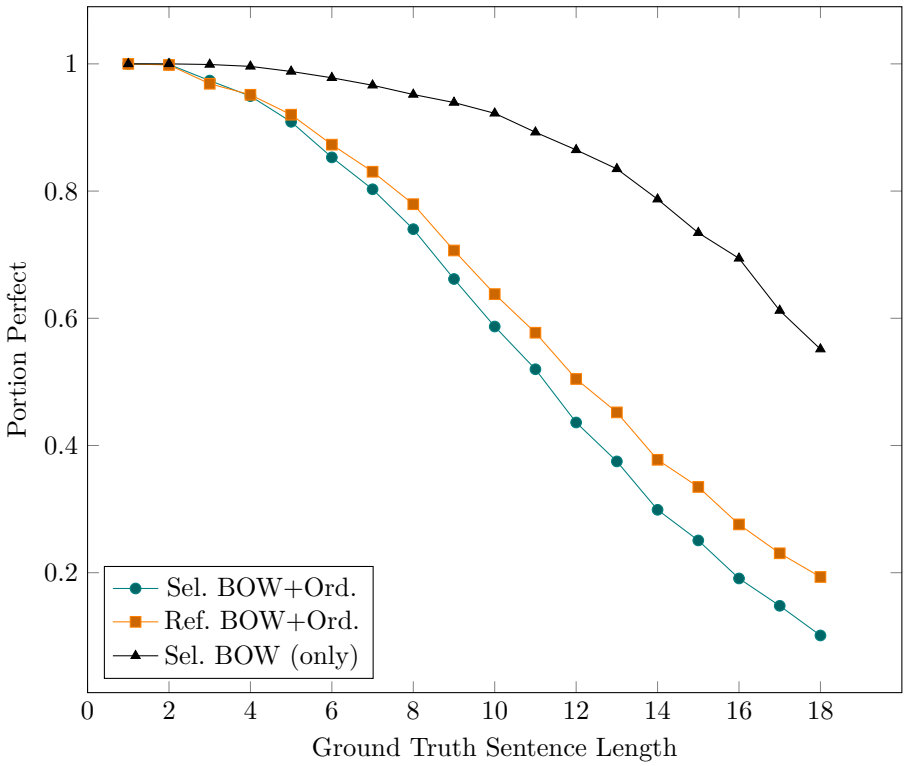


Figure 10.4: The portion of sentences reconstructed perfectly by the Sel. BOW+Ord. process. Shown also is the results on ordering only (Ref. BOW+Ord.), which orders the reference BOWS; and the portion of BOWs perfect from the word selection step only (Sel. BOW (only)) i.e. the input to the ordering step.

10.5 Results and Discussion

The overall results for our method (Sel. BOW+Ord.) sentence generation are shown in Table 10.1. Also shown are the results for just the ordering step, when the reference bag of words provided as the input (Ref. BOW+Ord.). The Perfect Sentences column shows the portion of the output sentences which exactly reproduce the input. The more forgiving BLEU Score **Papineni2002** is shown to measure how close the generated sentence is to the original. The portion of cases for which there does exist a solution within the constraints of the MIP ordering problem is showin in Portion Feasible. In the other cases, where the MIP problem is unsolvable, for calculating the BLEU score, we order the BOW based on the order resulting from the word selection step, or in the reference case randomly.

Table 10.2 shows the results reported by **White2015BOWgen** for the Word Selection step only (Sel. BOW (only)). The Perfect BOWs column reports the portion of the generated BOWs which perfectly match the reference BOWs. We also show the Mean Precision, averaged across all cases, this being the number of correct words generated, out of the total number of words generated. Similarly, the Mean Jaccard Index is shown, which is a measure of the similarities of the BOWs, being the size of the intersection of the generated BOW with the reference BOW, divided by the size of their union. We present these results to show how each step's performance impacts the overall system.

Both the Ref. BOW+Ord. and Sel. BOW (only) results place an upper bound on the performance of the overall approach (Sel. BOW+Ord.). The ordering only results (Ref. BOW+Ord.) show the best performance that can be obtained in ordering with this language model, when no mistakes are made in selection. Similarly, the selection only results (Sel. BOW (only)) are bounding as no matter how good the word ordering method is, it cannot recreate perfectly accurate sentences using incorrect words.

It can be noted that Ref. BOW+Ord. and Sel. BOW+Ord. were significantly more accurate than the best results reported by **Horvat2014**. We attribute this to Horvat and Byrne preprocessing the evaluation corpora to remove the easier sentences with 4 or less words. We did not remove short sentences from the corpus. The performance on these sentences was particularly high, thus improving the overall results on ordering.

The overall resynthesis (Sel. BOW+Ord.) degrades as the sentence length increases as shown in Figure 10.4. It can be seen from the figure that sentence length is a critical factor in the performance. The performance drop is largely from the complexity in the ordering step when faced with long sentences. This is evident in Figure 9.2, as performance degrades at almost the same rate even when using the perfect BOW (compare Ref. BOW+Ord. vs Sel. BOW+Ord.); rather than being degraded by the failures in the word selection step (Sel. BOW (only)). We can conclude that sentences with word selection failures (Sel. BOW (only))

[3] Available online at `http://white.ucc.asn.au/publications/White2016BOWgen/`
[4] Available online at `http://nlp.stanford.edu/projects/glove/`
[5] `http://www.cs.toronto.edu/~mbweb/`
[6] `http://white.ucc.asn.au/publications/White2016SOWE2Sent/`

are also generally sentences which would have word ordering failures even with perfect BOW (Ref. BOW+Ord.). Thus improving word selection, without also improving ordering, would not have improved the overall results significantly.

From observing examples of the output of method we note that normally mistakes made in the word selection step result in an unorderable sentence. Failures in selection are likely to result in a BOW that cannot be grammatically combined e.g. missing conjunctions. This results in no feasible solutions to the word ordering problem.

Our method considers the word selection and word ordering as separate steps. This means that unorderable words can be selected if there is an error in the first step. This is not a problem for the existing methods of **iyyer2014generating** and of **Bowman2015SmoothGeneration**. **iyyer2014generating** guarantees grammatical correctness, as the syntax tree must be provided as an input for resynthesis – thus key ordering information is indirectly provided and it is generated into. **Bowman2015SmoothG** on the other hand integrates the language model with the sentence embedding so that every point in the vector space includes information about word order. In general, it seems clear that incorporating knowledge about order, or at least co-occurrence probabilities, should be certain to improve the selection step. Even so the current simple approach has a strong capacity to get back the input, without such enhancement.

10.6 Conclusion

A method was presented for regenerating sentences, from the sum of a sentence's word embeddings. It uses sums of existing word embeddings, which are machine learnt to represent the sentences, and then generates natural language output, using only the embeddings and a simple trigram language model. Unlike existing methods, the generation method itself is deterministic rather than being based on machine-learnt encoder/decoder models. The method involved two steps, word selection and word ordering.

The first part is the word selection problem, of going from the sum of embeddings to a bag of words. To solve this we utilised the method presented in **White2015BOWgen**. Their greedy algorithm was found to perform well at regenerating a BOW. The second part was word ordering. This was done through a MIP bases reformulation of the work of the graph-based work of **Horvat2014**. It was demonstrated that a probabilistic language model can be used to order the bag of words output to regenerate the original sentences. While it is certainly impossible to do this perfectly in every case, for many sentences the most likely ordering is correct.

From a theoretical basis the resolvability of the selection problem, presented by **White2015BOWgen**, shows that adding up the word embeddings does preserve the information on which words were used; particularly for higher dimensional embeddings. This shows clearly that collisions do not occur (at least with frequency) such that two unrelated sentences do not end up with the same SOWE representation. This

work extends that by considering if the order can be recovered based on simple corpus statistics. Its recoverability is dependent, in part, on how frequent sentences with the same words in different order are in the corpus language – if they were very frequent then non-order preserving, non-compositional representations like SOWE would be poor at capturing meaning, and the ordering task would generally fail. As the method we presented generally does succeed, we can conclude that word order ambiguity is not a dominating problem. This supports the use of simple approaches like SOWE as a meaning representation for sentences – at least for sufficiently short sentences.

The technique was only evaluated on sentences with up to 18 words (inclusive), due to computational time limitations. Both accuracy and running time worsens exponentially as sentence length increases. With that said, short sentences are sufficient for many practical uses. For longer sentences, it is questionable as to the extent the information used is preserved by the SOWE representation – given they tend to have large substructures (like this one) compositional models are expected to be more useful. In evaluating such future representations, the method we present here is a useful baseline.

10.6.1 Acknowledgements

This research is supported by the Australian Postgraduate Award, and partially funded by Australian Research Council grants DP150102405 and LP110100050. Computational resources were provided by the National eResearch Collaboration Tools and Resources project (Nectar).

10.7 Supplementary Materials to Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem

These supplementary materials show additional examples of the performance of our method against the works of **iyyer2014generating**; **Bowman2015SmoothGeneration**, as of our well as on sentences with ambiguous order. Bare in mind, exact reproduction is not the goal of either prior work; nor truly is it a goal of out work. Our goal being the regeneration of sentences while preserving meaning – exact reproduction does of course meet that goal. The examples that follow should highlight the differences in the performance of the methods.

Tables 10.3 to 10.5 show quantitative examples; including comparison to the existing works. In these tables ✗ and ✓ are used to show correctness of the output in the selection (Sel.) and in the ordering (Ord.) steps.

The sentences shown in Table 10.3, are difficult. The table features long complex sentences containing many proper nouns. These examples are sourced from **iyyer2014generating**. The output from their DT-RAE method is also shown for contrast. Only 3C is completed perfectly by our method. Of the remainder the MIP word ordering problem has no solutions, except in 3D, where it is wrong, but does produce an ordered

sentence. In the others the language model constraints does not return any feasible ($P(\tau) > 0$) ordering solutions. This failure may be attributed in a large part to the proper nouns. Proper nouns are very sparse in any training corpus for language modelling. The Kneser-Ney smoothed trigrams back-off only down to bigrams, so if the words of the bigrams from the training corpus never appear adjacently in the training corpus, ordering fails. This is largely the case for very rare words. The other significant factor is the sentence length.

The sentences in Table 10.4, are short and use common words – they are easy to resynthesis. These examples come from **Bowman2015SmoothGeneration** The output of their VAE based approach can be compared to that from our approach. Of the three there were two exact match's, and one failure.

Normally mistakes made in the word selection step result in an unorderable sentence. Failures in selection are likely to result in a BOW that cannot be grammatically combined e.g. missing conjunctions. This results in no feasible solutions to the word ordering problem.

The examples shown in Table 10.5 highlight sentences where the order is ambiguous – where there are multiple reasonable solutions to the word ordering problem. In both cases the word selection performs perfectly, but the ordering is varied. In 5A, the Ref. BOW+Ord. sentence and the overall Sel. BOW+Ord. sentence in word order but not in word content. This is because under the trigram language model both sentences have exactly identical probabilities, so it comes to which solution is found first, which varies on the state of the MIP solver. In 5B the word order is switched – "from Paris to London" vs "to London from Paris", which has the same meaning. But, it could also have switched the place names. In cases like this where two orderings are reasonable, the ordering method is certain to fail consistently for one of the orderings. Though it is possible to output the second (and third etc.) most probable ordering, which does ameliorate the failure somewhat. This is the key limitation which prevents this method from direct practical applications.

		Sel.	Ord.
3A Reference	name this 1922 novel about leopold bloom written by james joyce .		
Ref. BOW+Ord.	written by name this . novel about 1922 bloom leopold james joyce	–	✖
Sel. BOW+Ord.	written novel by name james about leopold this bloom 1922 joyce .	✓	✖
DT-RAE Ref.	name this 1906 novel about gottlieb_fecknoe inspired by james_joyce		
DT-RAE Para.	what is this william golding novel by its written writer		
3B Reference	ralph waldo emerson dismissed this poet as the jingle man and james russell lowell called him three-fifths genius and two-fifths sheer fudge .	**Sel.**	**Ord.**
Ref. BOW+Ord.	sheer this as james two-fifths emerson fudge lowell poet genius waldo called russell the and ralph and him . dismissed jingle three-fifths man	–	✖
Sel. BOW+Ord.	him " james great as emerson genius ralph the lowell and sheer waldo three-fifths man fudge dismissed jingle russell two-fifths and gwalchmai 2009 vice-versa prominent called 21.25 explained	✗	✖
DT-RAE Ref.	henry_david_thoreau rejected this author like the tsar boat and imbalance created known good writing and his own death		
DT-RAE Para.	henry_david_thoreau rejected him through their stories to go money well inspired stories to write as her writing		
3C Reference	this is the basis of a comedy of manners first performed in 1892 .	**Sel.**	**Ord.**
Ref. BOW+Ord.	this is the basis of a comedy of manners first performed in 1892 .	–	✓
Sel. BOW+Ord.	this is the basis of a comedy of manners first performed in 1892 .	✓	✓
DT-RAE Ref.	another is the subject of this trilogy of romance most performed in 1874		
DT-RAE Para.	subject of drama from him about romance		
3D Reference	in a third novel a sailor abandons the patna and meets marlow who in another novel meets kurtz in the congo .	**Sel.**	**Ord.**
Ref. BOW+Ord.	kurtz and another meets sailor meets the marlow who abandons a third novel in a novel in the congo in patna .	–	✗
Sel. BOW+Ord.	kurtz and another meets sailor meets the marlow who abandons a third novel in a novel in the congo in patna .	✓	✗
DT-RAE Ref.	during the short book the lady seduces the family and meets cousin he in a novel dies sister from the mr.		
DT-RAE Para.	during book of its author young lady seduces the family to marry old suicide while i marries himself in marriage		
3E Reference	thus she leaves her husband and child for aleksei vronsky but all ends sadly when she leaps in front of a train .	**Sel.**	**Ord.**
Ref. BOW+Ord.	train front of child vronsky but and for leaps thus sadly all her she she in when aleksei husband ends a . leaves	–	✖
Sel. BOW+Ord.	she her all when child for leaves front but and train ends husband aleksei leaps of vronsky in a sadly micro-history thus , she the	✗	✖
DT-RAE Ref.	however she leaves her sister and daughter from former fiancé and she ends unfortunately when narrator drives into life of a house		
DT-RAE Para.	leaves the sister of man in this novel		

Table 10.3: A comparison our method, to the example sentences generated by the DT-RAE method of **iyyer2014generating**. Ref. BOW+Ord. shows the word ordering step on the reference BOW. the Sel. and Ord. columns indicate if the output had the correct words selected, and ordered respectively. With ✓ indicating correct and ✗ indicating incorrect. ✖ indicates not only that ordering was not correct, but that the MIP problem had no feasible solutions at all. DT-RAE Ref. shows the result of the method of **iyyer2014generating**, when the dependency tree of the output is provided to the generating process, whereas in DT-RAE Para. an arbitrary dependency tree is provided to the generating process. Note that the reference used as input to Sel. BOW+Ord. and Ref. BOW+Ord. sentence was varied slightly from that used in **iyyer2014generating** and **White2015BOWgen**, in that terminating punctuation was not removed, and nor were multiword entity references grouped into single tokens.

		Sel.	Ord.
4A Reference	we looked out at the setting sun .		
Ref. BOW+Ord.	we looked out at the setting sun .	–	✓
Sel. BOW+Ord.	we looked out at the setting sun .	✓	✓
VAE Mean	they were laughing at the same time .		
VAE Sample1	ill see you in the early morning .		
VAE Sample2	i looked up at the blue sky .		
VAE Sample3	it was down on the dance floor .		
4B Reference	i went to the kitchen .	**Sel.**	**Ord.**
Ref. BOW+Ord.	i went to the kitchen .	–	✓
Sel. BOW+Ord.	i went to the kitchen .	✓	✓
VAE Mean	i went to the kitchen .		
VAE Sample1	i went to my apartment .		
VAE Sample2	i looked around the room .		
VAE Sample3	i turned back to the table .		
4C Reference	how are you doing ?	**Sel.**	**Ord.**
Ref. BOW+Ord.	how are you doing ?	–	✓
Sel. BOW+Ord.	how 're do well ?	✗	✗
VAE Mean	what are you doing ?		
VAE Sample1	" are you sure ?		
VAE Sample2	what are you doing, ?		
VAE Sample3	what are you doing ?		

Table 10.4: A comparison of the output of the Two Step process proposed in this paper, to the example sentences generated by the VAE method of **Bowman2015SmoothGeneration**.

		Sel.	Ord.
5A Reference	it was the worst of times , it was the best of times .		
Ref. BOW+Ord.	it was the worst of times , it was the best of times .	–	✓
Sel. BOW+Ord.	it was the best of times , it was the worst of times .	✓	✗
5B Reference	please give me directions from Paris to London .	**Sel.**	**Ord.**
Ref. BOW+Ord.	please give me directions to London from Paris .	–	✗
Sel. BOW+Ord.	please give me directions to London from Paris .	✓	✗

Table 10.5: A pair of example sentences, where the correct order is particularly ambiguous.

Chapter 11

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. Chapter 5 investigated classifying paraphrases as a means to investigate the quality of SOWE as a sentence embedding method. Chapter 6 defined models for color estimation from short phrases. Chapter 7 considered if we could use weighted combinations of sense embeddings to better capture the sense used in a particular example. Chapter 8 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. Chapter 9 demonstrated a method that could partially recover bags of words from a given SOWE. Chapter 10 extended this work by attempting to order those bags of words into sentences. This demonstrated that a surpris-

ing 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.

11.1 Future work

11.1.1 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 6, 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.

11.1.2 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-EtAl:2014:EMNLP2014**), do not capture word content as well as LCOWE (**ac2018probingsentencevectors**). They do, however, have state of the art order representation. An interesting combination of the two, would be an encoder-decoder model, where the intermediate state between the encoder and the decoder, is augmented by concatenating the final RNN output, with a sum of word embeddings, for all the input words to the encoder. This corresponds to a bypass of the RNN units. A similar bypass of intermediate layers has been used in feed-forward networks including the notable neural probabilistic language model (**NPLM**).

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 (**lookafterdata**). 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 (**Julia**). 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 (**drummond2009replicability**), we maintain that being able to run the original code is important for checking, for understanding, for extension, and for future comparisons.

VabdewakkeReproduceableResearch 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" (**10.1371/journal.pbio.1001745**).

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 (**wren2008url**). 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] provide more convenient accesses with suitable preprocessing for commonly

[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

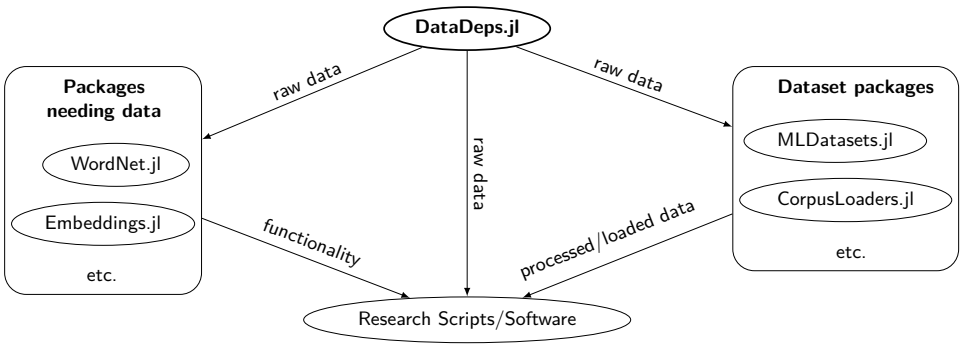


Figure A.1: The current package ecosystem depending on DataDeps.jl.

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 researcher can upload their data to an archival repository and update the registration.

[6] `https://github.com/JuliaText/WordNet.jl`

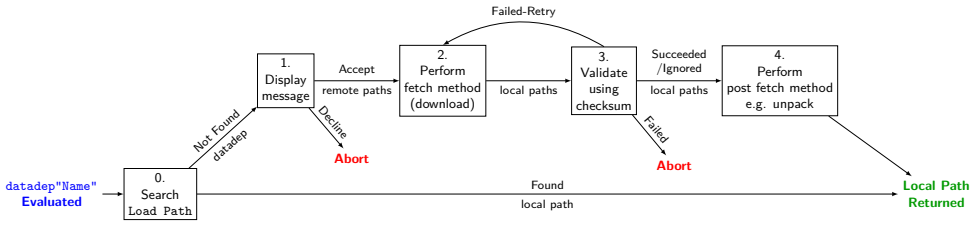


Figure A.2: The process that is executed when a data dependency is accessed by name.

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.

A.3 Availability

A.3.1 Operating system

DataDeps.jl is verified to work on Windows 7+, Linux, Mac OSX.

[7] `https://github.com/quiltdata/quilt`

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: White2015BOWgen 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 (**miller1995wordnet**). 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 (**Julia**), 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 (**2018arXiv180801091W**). 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 (**Julia**) 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 (**flux**). 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 (**2018arXiv180801091W**), 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 (**mikolov2013efficient**) for English
- multiple sets of GLoVE (**pennington2014glove**) embeddings for English
- multiple sets of FastText embeddings (**bojanowski2016enriching**; **fasttext157lang**) 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 (**Julia**) client library for the TensorFlow deep-learning framework (**tensorflow2015-whitepaper**; **tensorflow2016**). 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) (**MLandPL**). 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 (**mocha2014**), Knet (**knet2016**) and Flux (**flux**), as well as bindings to other frameworks such as MxNet (**mxnet2015**). 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.