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

September 6, 2018

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.

n here]

Signature:

Date: September 6, 2018

In memoriam of
Laurie White
1927–2018

Abstract

As Webber's classic 1929 text *English Composition and Literature* states: "A sentence is a group of words expressing a complete thought." People use natural language is used 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 an area which fundamentally 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 the novel contributions published as in conference proceedings or journals; and with the literature review having been published as part of a book.

When considering how to represent English input into a natural language processing system, a common response is to consider modelling it as a sequential modelling problem: 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.

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 out performed by sums of embeddings in tasks where word order is not highly significant. It is found that even in tasks were word order does matter to an extent, the improved training capacity of the simpler model still can mean that it performs better than more complex models. This limitation thus hurts surprisingly little.

Acknowledgements

Lorem Ipsum

Write this

Authorship declaration

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

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

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

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

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

Student contribution to work:
Devised problem. Designed and implemented algorithms. Conducted experiments. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.
Co-author signatures and dates:

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018b). "NovelPerspective: Identifying Point of View Characters". In: *Proceedings of ACL 2018, System Demonstrations.* Association for Computational Linguistics
Location in thesis: 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.
Co-author signatures and dates:

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

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

Details of the work:
Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2018a). "DataDeps.jl: Repeatable Data Setup for Reproducible Data Science". In: *for Journal of Open Research Software (Under Review)*
Location in thesis: Chapter 9
Student contribution to work:
Primary author of software. Created figures. Wrote publication. Supervisors reviewed and provided useful feedback for improvement.

Co-author signatures and dates:

Details of the work:
Lyndon White and Sebastin Santy (2018). "DataDepsGenerators.jl: making reusing data easy by automatically generating DataDeps.jl registration code". In: *Journal of Open Source Software (Under Review)*
Location in thesis: Chapter 10
Student contribution to work:
Original author of software. Provided direction, guidance, and code review for its enhancement. Wrote publication.
Co-author signatures and dates:

I, Roberto Togneri certify that the student statements
regarding their contribution to each of the works listed above are correct.
Coordinating supervisor signature: [insert signature]
Date: [insert date]

Dr Togner
sign

Contents

Part I

Introduction

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 the rich and highly structured nature of human language. However at a second glance, similar surface information has been used in information retrieval with great success since the inception of the field (Maron 1961). Linear combinations of embeddings can be considered as a dimensionality reduction of a bag of words, with a particular weighting scheme. Dimensionality reduction can be characterised as finding the best low dimensional representation of a high dimensional input according to some quality criterion. In the case of word embeddings, that quality criterion is generally relating 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*, or *embedding-BOW* or similar. The primary focus of this work has been on sums of word embeddings (SOWE), i.e. a linear combination with unit weight. Closely related to this is a mean of word embeddings (MOWE), which is a SOWE 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.

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

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

Cífka and Bojar (2018) found that taking a mean of word embeddings outperformed almost all 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 can not truly handle all the complexities of language. But rather that the occurrence of the complexities they can not handle is rarer in practice in many tasks than is often expected.

Conneau et al. (2018) constructed 10 probing tasks to isolate some of the information captured by sentence representations. They found the strong performance of the mean of word embeddings on sentence level tasks to be striking. They attribute it to the sentence level information being redundantly encoded in the word-forms: the surface level information is surprisingly useful for tasks which at first look very sophisticated. With the exception of their word-content task, they did find more sophisticated models 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 that even in tasks where it would seem that non-surface information incorporating word-order is required, in practice other issues cause the more powerful models that are (theoretically) able to handle these situations correctly to be never-the-less outperformed. This is particularly the case where the theoretical improvement from incorporating this information is small, relative to the practical complexity of the techniques required to leverage it. Such a case where word order matters but the error from ignoring it is small, is particular illustrated in **??**.

At a high-level the success of these techniques comes down to most human language being easy to understand and simple. This expectation of language being easily understood is highlighted by Grice (1975), which claims that the communication is conducted following a cooperative principle. The overall supermaxim for Grice's cooperative principle is the speakers are expected to "be perspicuous" or more perspicuously, to use speech that is clearly expressed and easily understood. The particular relevant maxims within the 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 effects that meaning; and where multiple possible word orders exist. For such an adversary to be significant, both word orders must be reasonably likely to occur. However; such cases are rarer than one might expect as is demonstrated in **??**. Particularly when punctuation is included, which it reasonably can be as a token embedding. 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.

This when applied in sentence or phrase representation contexts, such as discussed in Chapter 5, and **??** which gives support to the notion that word order is often not a very significant feature in determining meaning. While it seems clear that word order, and other factors of linguistic structure must contribute significantly to the meaning of the phrase. However, our result 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 greater linguistic structure may be the key to bridging the 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 discussed in Chapter 5, and **??** leads us to consider other uses of linear combinations for representation. **??** and Chapter 6 consider tasks well outside of phrase representation where the order clearly does not matter.

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

On the complexity of models. 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 and RvNNs, 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 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.

There are a few papers which show RNN are only as powerful a Regex, or t they don't have much memory

1.1 Thesis Overview

This thesis tackles a number of natural language understanding problems, and in the solutions draws conclusions on the capacity of linear combinations of embeddings.

1.1.1 Overview of Literature

This dissertation begins with a detailed discussion of the established methods for input representation in natural language understanding tasks. This literature review does not focus on linear combinations of embeddings, which we develop upon throughout the rest of this dissertation Rather it focuses upon the techniques we build upon, and the alternatives to our methods. Part II was originally published as the main content of White et al. (2018a). It excludes the introductory chapters on machine learning and recurrent neural networks which was present in that book. Further to the literature review section of this dissertation, each chapter in Part III includes a background or related works section with particularly relevant works to that paper discussed, as is usual for a thesis by publication.

1.1.2 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 the basis 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.

1.1.3 Chapter 3 Word Sense Representations

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

Chapter	**Structure**	Task	**Embeddings**
Chapter 5	Sentences	Paraphrase grouping	Word2Vec (Mikolov et al. 2013a)
??	Short Phrases	Color understanding	FastText (Bojanowski et al. 2017)
??	Word Senses	Similarity with context & Word sense disambiguation	AdaGram (Bartunov et al. 2015) & Bespoke greedy sense embeddings
Chapter 6	Adj. Contexts	POV character detection	FastText (Bojanowski et al. 2017)
??	Sentences	Recovering bags of words	GLoVE (Pennington, Socher, and Manning 2014)
??	Sentences	Recovering sentences	GLoVE (Pennington, Socher, and Manning 2014)

Table 1.1: Summary of the investigations published within this dissertation.

1.1.4 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 shorted phrases. This chapter contains some discussion of the sums of word embeddings that are the focus of this work, but contains primarily discussion of the alternatives which we contrast with.

1.1.5 Overview of Novel Contributions

An overview of tasked 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 **??** through a color understanding (estimation) task. Given the observed properties found by sums of word embeddings, this leads to the investigation of if weighted sums of word sense embeddings might better resplendent a particular usage of a word in **??**. The capacity also lends to the investigation of using a sum of word embeddings to represent the contexts of all usages of a named entity, for the point of view character detection task investigated in Chapter 6. We conclude with a complementary pair of works in **????**, which investigate the ability to recover bags of words and sentences, from sums of word embeddings representing sentences. These final works illustrate some of the reasons why the linear combinations work so well.

Chapter 5: (White et al. 2015)
"How Well Sentence Embeddings Capture Meaning"

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 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 (such as a deep neural network) 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 is advantaged by being much higher dimensionality than other models. The LCOWEs outperform the bag of words as they also capture synonyms and other features of lexical relatedness. Slightly worse than the bag of words was the bag of words with PCA dimensionality reduction to 300 dimensions. This confirms our expectation that LCOWEs are a better form of dimensionality reduction for preserving meaning from a bag of words than PCA.

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

A limitation of our investigation is that it does not include the examination of any encoder-decoder based methods, such as Skip-Thought (Kiros et al. 2015), or machine translation models. Another limitation of the work is that the unfolding recursive autoencoder (Socher et al. 2011a) use 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 task has its own important value, it is harder to judge the generalisation ability for example of a sentence representation that works well for sentiment analysis. This idea of using an abstract probing task to evaluate sentence representations has been significantly advanced and generalised to a battery of such tasks in later works such as Adi et al. (2017) and Conneau et al. (2018). The interesting finding in our work, which significantly contributed to the direction of this dissertation, was that the LCOWEs (SOWE/MOWE) were notably the best performing on this 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 color. They are thus useful for evaluating the performance of LCOWE on short phrases.

??: (White et al. 2018)
“Learning of Colors from Color Names: Distribution and Point Estimation”

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

The solving problem of color estimation from natural language color name, has pragmatic uses. Color estimation from description has utility as a tool for improving human-computer interaction. For example allowing free(-er) text for specifying colors in plotting software, using point estimation. It also has utility as an education tool: people from different cultures, especially non-native English speakers, may not know exactly what color range is described by `dark salmon`, and our model allows for tools to be created to answer such queries using distribution estimation.

A limitation of this study is the metrics used. For distribution estimation, the perplexity of the discretized distributions in color space is reported. It would be preferable to uses 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, using 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.

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 the a neural network based method for color distribution estimation, which itself has practical use as a teaching tool; 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 representation.

??: (White et al. 2018b)
"Finding Word Sense Embeddings Of Known Meaning"

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. Further more, if one knows the probably of each of the coarse induced senses being the correct sense for this use, then it makes sense that the location of the fine grained sense embedding would be closer to the more likely coarse sense, and further from the less likely coarse sense. As such we propose a method to define these specific case word senses based on a probability weighted sum of coarser word sense embeddings. We say that we *refit* the original sense embeddings, using the single example sentence to induce the fine grained sense embedding.

Using this we define a similarity measure which we call RefittedSim, which we find to work better than AvgSimC (Reisinger and Mooney 2010). AvgSimC is a probability-weighted average of all the pairwise similarity scores for each sense embedding. In contrast RefittedSim is a single similarity score as measured between the two refitted vectors – which are the probability weighted averages of the coarser sense vectors. On the embeddings used in our evaluations this gave a solid improvement over AvgSimC. It is also asymptotically faster to evaluate.

We also evaluated using refitting for word sense disambiguation (WSD). Normally, induced senses can not 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 this we can use the skip-gram formulation for probability of the context given the refitted sense, and so apply Baye's theorem to find the most-likely sense. However, we found that the results were only marginally better than the baseline. Though it was notably better than the results of the method presented by Agirre et al. (2006); which, to the best of our knowledge, is the only prior method for leveraging induced senses for WSD with only a limited number of examples. Nearly unsupervised WSD is a very difficult problem; with a strong baseline of simply reporting the most-common sense. Our results do suggest that our refitting method does not learn features that are antithetical to 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 was that it did not perform the evaluation on state-of-the-art word-sense embeddings. As such, while it's comparisons between these embeddings are valid they can not be readily compared to the current state-of-the-art on the tasks. It is thus not entirely clear that improvements when our method is applied to better performing models would be proportionate.

The **key contribution** of this work was to define a method for specializing word sense embeddings for a single use case. In doing 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 representation according to how likely they are to be correct. Important secondary contributions include the method for smoothing the probability of correctness; and RefittedSim, a new similarity measure using this refitting to evaluated the similarity of word in context.

Chapter 6: (White et al. 2018b)
"NovelPerspective: Identifying Point of View Characters"

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 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 they are from the perspective of. 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 can not 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, 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-fit quiet easily. It's performance dropped sharply on the test set, compared to it's oracle performance if trained on the testset, 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 for 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 significant. With logistic regression, SVM, and decision trees giving similarly high results for both feature sets.

The **key contribution** of this work was 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 mean of word embeddings 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`.

??: (White et al. 2016a)
"Generating Bags of Words from the Sums of their Word Embeddings"

Given the consideration of a sum of word embeddings as dimensionality reduced form of a bag of words a important question is to how recoverable the bag of words is from the sum. A practical way to lower bound the loss of information is to demonstrate a deterministic method that can recover a portion of the bag of words.

We propose as method to extract the original bag of words from a sum of word embeddings. Thus placing a bound on the information lost in 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 it's 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 the higher dimensional the embeddings are, and with fewer words. Even with relatively low dimensions it works quiet well. This shows that embeddings are not for example constantly cancelling each other in the sum.

Can I comment that
”This meth
would not
work as we
on a mean
of word embeddings”,
s there son
thing inter
esting to sa
about that

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

The **key contribution** of this work was produce a system which attempts to convert 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.

??: (White et al. 2016b)
“Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem”

Given that it was demonstrated that the bag of word can be recovered, the obvious follow up question is if we can recover the the sentence. This **??** is a small supplement to **??**.

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

We find that surprisingly often they can. The majority of sentences of length up to 18 can be successfully recovered from a SOWE. With, as expected, the longer the sentence the more difficult the recovery. This suggests that the number of likely possible orderings for the words in a sentence is much lower than it may at first seem. Particularly since this method is based on a simple trigram language does so well. There is no doubt that a more sophisticated language model perform significantly better.

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

The **key contribution** of this work was produce a system which attempts order bags of words recovered from sums of word embeddings into the sentences from which they came. The capacity to do this places a lower bound on how well sentences can be represented. If a correctly sentence can be fully recovered from a sum of word embeddings using just a language model; then a SOWE is effectively sending sentences to unique areas of the representation space. The use of the methods of **??** and **??**, together with a system trained to output a approximation to a SOWE, is an interesting, though not really practical, method for natural language generation.

1.2 Some Math

Should I
just delete
this sectior
What is it
adding?

1.2.1 Word Embeddings

Given a word represented by an integer w, from a vocabulary $\mathbb{V} \subset \mathbb{Z}$, and a matrix of embeddings, represented as C, its embedding can be found by slicing out the wth column: $C_{:,w}$.

For $\tilde{e}_w$ the elementary unit vector, i.e. the one-hot vector representation of w, it can be seen that the word embedding can be represented as the product of the embedding matrix with the one-hot vector.

$$C_{:,w} = C\,\tilde{e}_w$$

1.2.2 Sum of Word Embeddings

For some sequence of (not necessarily unique words) words $\mathcal{W} \in \mathbb{V}^{\mathbb{Z}_0}$ represented $\mathcal{W} = \left(w^1, w^2, \ldots, w^n\right)$, where w^i is an integer representing the identity of ith word.

The sum of word embeddings (SOWE) representation is written as:

$$\sum_{i=1}^{i=n} C_{:,w^i}$$

The bag of word (BOW) representation is written as a vector from $\mathbb{Z}^{|\mathbb{V}|}$.

$$\tilde{x} = \sum_{i=1}^{i=n} \tilde{e}_{w^i}$$

,

Using this the sum of word embeddings can be seen to be the product of the embedding matrix with a BOW vector.

$$\sum_{i=1}^{i=n} C_{:,w^i} = \sum_{i=1}^{i=n} C\,\tilde{e}_{w^i} = C\,\sum_{i=1}^{i=n} \tilde{e}_{w^i} = C\,\tilde{x}$$

1.2.3 Mean of Word Embeddings

The mean of word embeddings representation is written as:

$$\frac{1}{n}\sum_{i=1}^{i=n} C_{:,w^i}$$

Note that n is equal to the element-wise sum of the BOW vector (x), i.e. to its l1-norm:

$$n = \left\|\tilde{x}\right\|_1 = \sum_{\forall j \in \mathbb{V}} \tilde{x}_j$$

Thus the mean of word embeddings can be seen as the product of the embedding matrix with the l1-normalized BOW vector.

$$\frac{1}{n}\sum_{i=1}^{i=n} C_{:,w^i} = \frac{1}{n}\sum_{i=1}^{i=n} C\,\tilde{e}_{w^i} = \frac{1}{n}C\,\sum_{i=1}^{i=n} x = C\,\frac{\tilde{x}}{\left\|\tilde{x}\right\|_1}$$

1.2.4 Linear Combination of Embeddings

The full generalisation of this is that any linear combination of embeddings can be seen as product of the embedding matrix, the a weighted bag of words.

A weighting function can be defined for any linear combination scheme. That is to say mapping from a given bag of words, and a word, to the weighting of that word. $\alpha : \mathbb{Z}^{|\mathbb{V}|} \times \mathbb{V} \to \mathbb{R}$.

(For example for the mean of word embeddings $\alpha(\tilde{x}, w) = \frac{1}{\|\tilde{x}\|_1}$.)

From the weighting function, we can evaluated it for a given BOW, for each word in the vocabulary to define a weighting vector $\tilde{\alpha}^{\tilde{\mathrm{x}}}$:

$$\tilde{\alpha}^{\tilde{\mathrm{x}}} = \left[\alpha(\tilde{x}, w)\right]_{w \in \mathbb{V}}$$

Using $\odot$ as the Hadamard (i.e. element-wise) product, we can thus write:

$$\sum_{i=1}^{i=n} \alpha(\tilde{x}, w^i) C_{:,w^i} = C \sum_{i=1}^{i=n} \alpha(\tilde{x}, w^i) \tilde{e}_{w^i} = C \left(\tilde{\alpha}^{\tilde{\mathrm{x}}} \odot \tilde{x}\right)$$

Part II

Literature Review

Chapter 2

Word Representations

This chapter originally appears as Chapter 3 of the book Neural Representations of Natural Language, published by by Springer.

You shall know a word by the company it keeps.

– J.R. Firth, 1957

Abstract

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

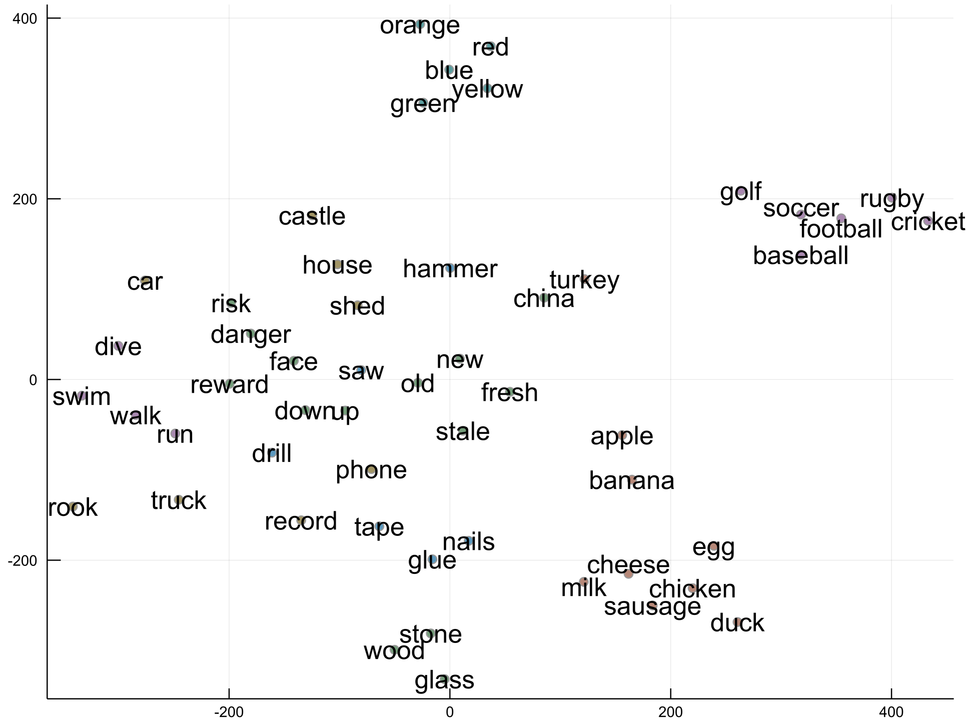
We begin the consideration of the representation of words using neural networks with the work on language modeling. This is not the only place one could begin the consideration: the information retrieval models, such as LSI (Dumais et al. 1988) and LDA (Blei, Ng, and Jordan 2003), based on word co-location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

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

2.1 Representations for Language Modeling

The language modeling task is to predict the next word given the prior words (Rosenfeld 2000). For example, if a sentence begins `For lunch I will have a hot`, then

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



there is a high probability that the next word will be `dog` or `meal`, and lower probabilities of words such as `day` or `are`. Mathematically it is formulated as:

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

or to use the compact notation

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

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

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

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

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

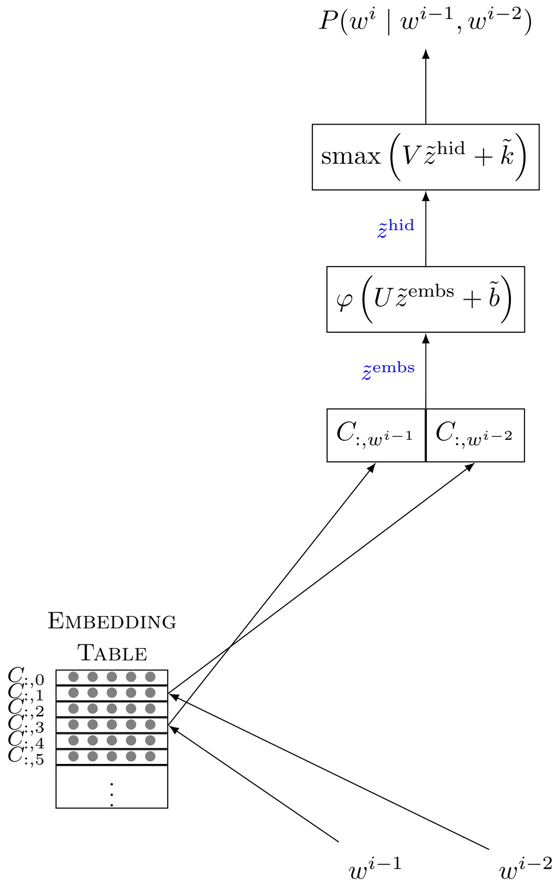
$$P(w^i \mid w^{i-1}, \ldots, w^1) = P(w^i \mid w^{i-1}, w^{i-2}) \tag{2.3}$$

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

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

A key issue with n-gram language models is that there exists a data-sparsity problem which causes issues in training them. Particularly for larger values of n. Most combinations of words occur very rarely (Ha et al. 2009). It is thus hard to estimate their occurrence probability. Combinations of words that do not occur in the corpus are naturally given a probability of zero. This is unlikely to be true though – it is simply a matter of rare phrases never occurring in a finite corpus. Several approaches have been taken to handle this. The simplest is add-one smoothing which adds an extra "fake" observation to every combination of terms. In common use are various back-off methods (Katz 1987; Kneser and Ney 1995) which use the bigram probabilities to estimate the probabilities of unseen trigrams (and so forth for other n-grams.). However, these methods are merely clever statistical tricks – ways to reassign probability mass to leave some left-over for unseen cases. Back-off is smarter than add-one smoothing, as it portions the probability fairly based on the $(n{-}1)$-gram probability. Better still would be a method which can learn to see the common-role of words (Brown et al. 1992). By looking at the fragment: `For lunch I want a hot`, any reader knows that the next word is most likely going to be a food. We know this for the same reason we know the next word in `For elevenses I had a cold …` is also going to be a food. Even though `elevenses` is a vary rare word, we know from the context that it is a meal (more on this later), and we know it shares other traits with meals, and similarly `have` / `had`, and `hot` / `cold`. These traits influence the words that can occur after them. Hard-clustering words into groups is nontrivial, particularly given words having multiple meanings, and subtle differences in use. Thus the motivation is for a language modeling method which makes use of these shared properties of the words, but considers them in a flexible soft way. This motivates the need for representations which hold such linguistic information. Such representations must be discoverable from the corpus, as it is beyond reasonable to effectively hard-code suitable feature extractors. This is exactly the kind of task which a neural network achieves implicitly in its internal representations.

Figure 2.2: The Neural Trigram Language Model


$P(w^i \mid w^{i-1}, w^{i-2})$
$\text{smax}\left(V\tilde{z}^{\text{hid}} + \tilde{k}\right)$
$\tilde{z}^{\text{hid}}$
$\varphi\left(U\tilde{z}^{\text{embs}} + \tilde{b}\right)$
$\tilde{z}^{\text{embs}}$
$C_{:,w^{i-1}}$
$C_{:,w^{i-2}}$
Embedding Table
$C_{:,0}$
$C_{:,1}$
$C_{:,2}$
$C_{:,3}$
$C_{:,4}$
$C_{:,5}$
w^{i-1}
w^{i-2}

2.1.1 The Neural Probabilistic Language Model

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

Simplified Model considered with Input Embeddings

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

The neural trigram probabilistic network is defined by:

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

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

In the neural trigram language model, each of the previous two words is used to look-up a vector from the embedding matrix. These are then concatenated to give a dense, continuous-space input to the above hidden layer. The output layer is a softmax layer, it gives the probabilities for each word in the vocabulary, such that $\hat{y}_{w^i} = P(w^i \mid w^{i-1}, w^{i-2})$. Thus producing a useful language model.

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

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):

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

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

Bayes-like Reformulation

When we consider the model with output embeddings, it is natural to also consider it under the light of the Bayes-like reformulation from Chapter 1 of White et al. (2018a):

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

which in this case is:

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

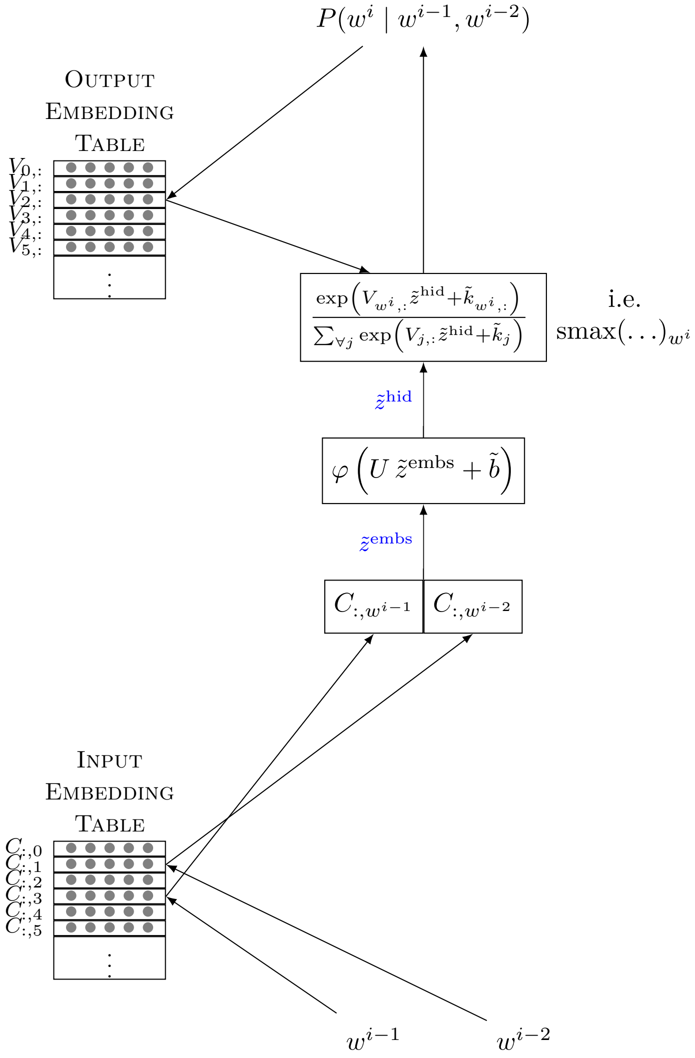
where $\sum_{\forall v\in\mathbb{V}}$ is summing over every possible word v from the vocabulary $\mathbb{V}$, which does include the case $v = w^i$.

Notice the term:

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

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

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


$P(w^i \mid w^{i-1}, w^{i-2})$
Output Embedding Table
$V_{0,:}$
$V_{1,:}$
$V_{2,:}$
$V_{3,:}$
$V_{4,:}$
$V_{5,:}$
$\vdots$
$\frac{\exp\left(V_{w^i,:}\tilde{z}^{\text{hid}}+\tilde{k}_{w^i,:}\right)}{\sum_{\forall j}\exp\left(V_{j,:}\tilde{z}^{\text{hid}}+\tilde{k}_j\right)}$
i.e. $\text{smax}(\ldots)_{w^i}$
$\tilde{z}^{\text{hid}}$
$\varphi\left(U\,\tilde{z}^{\text{embs}}+\tilde{b}\right)$
$\tilde{z}^{\text{embs}}$
$C_{:,w^{i-1}}$
$C_{:,w^{i-2}}$
Input Embedding Table
$C_{:,0}$
$C_{:,1}$
$C_{:,2}$
$C_{:,3}$
$C_{:,4}$
$C_{:,5}$
$\vdots$
w^{i-1}
w^{i-2}

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

The Neural Probabilistic Language Model

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

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

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 Schwenk (2004). Though in general it is not an unheard of technique in neural network machine learning.

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

2.1.2 RNN Language Models

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

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

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

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

[1] no pun intended

Figure 2.4: Neural Probabilistic Language Model

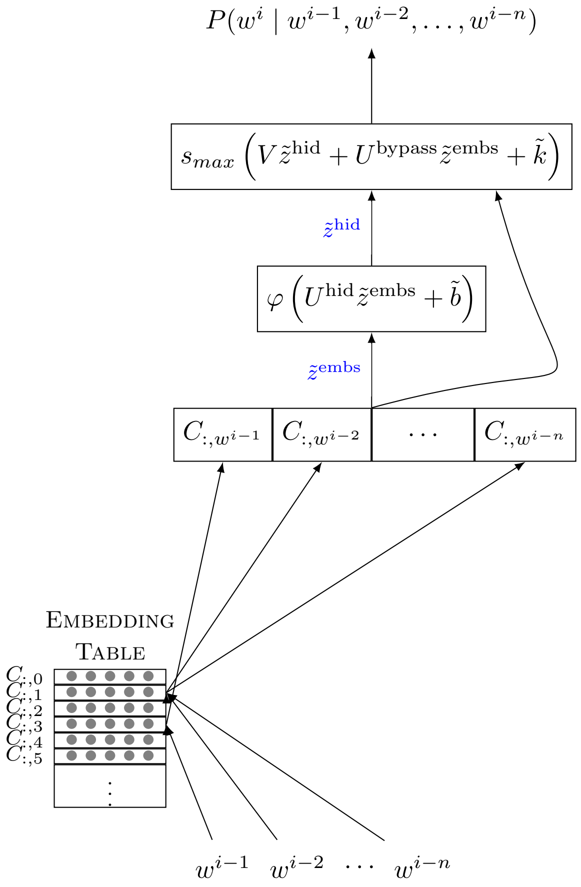
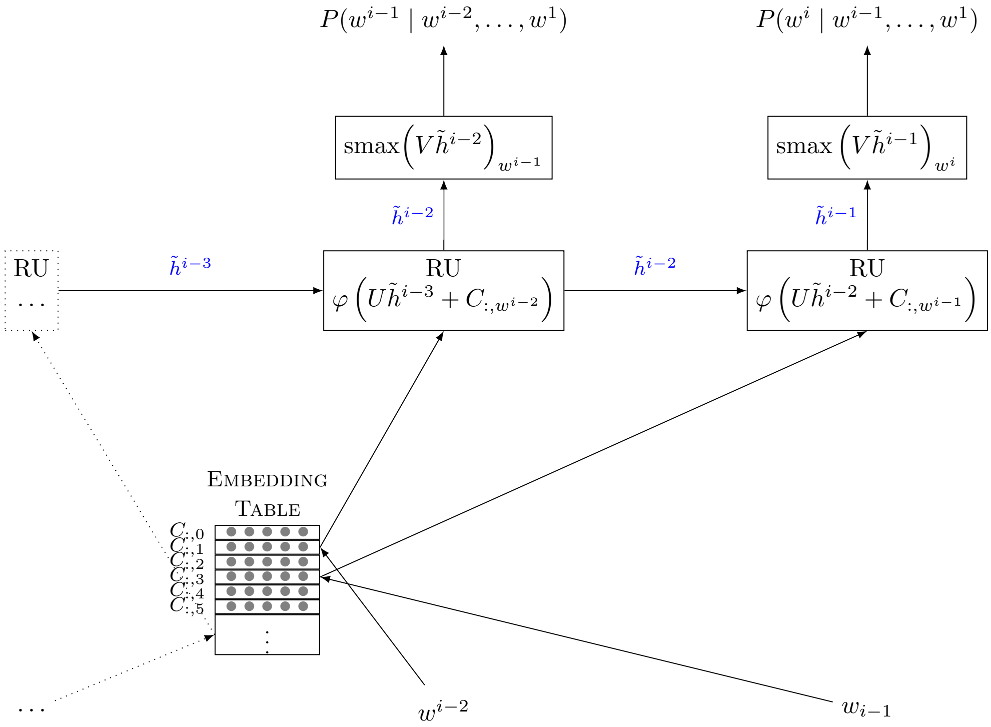

$P(w^i \mid w^{i-1}, w^{i-2}, \ldots, w^{i-n})$
$s_{max}\left(V\tilde{z}^{\text{hid}} + U^{\text{bypass}}\tilde{z}^{\text{embs}} + \tilde{k}\right)$
$\tilde{z}^{\text{hid}}$
$\varphi\left(U^{\text{hid}}\tilde{z}^{\text{embs}} + \tilde{b}\right)$
$\tilde{z}^{\text{embs}}$
$C_{:,w^{i-1}}$
$C_{:,w^{i-2}}$
$\cdots$
$C_{:,w^{i-n}}$
Embedding Table
$C_{:,0}$
$C_{:,1}$
$C_{:,2}$
$C_{:,3}$
$C_{:,4}$
$C_{:,5}$
$\vdots$
w^{i-1}
w^{i-2}
$\cdots$
w^{i-n}

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


$P(w^{i-1} \mid w^{i-2}, \ldots, w^1)$
$P(w^i \mid w^{i-1}, \ldots, w^1)$
$\text{smax}\left(V\tilde{h}^{i-2}\right)_{w^{i-1}}$
$\text{smax}\left(V\tilde{h}^{i-1}\right)_{w^i}$
$\tilde{h}^{i-2}$
$\tilde{h}^{i-1}$
RU
$\ldots$
$\tilde{h}^{i-3}$
RU
$\varphi\left(U\tilde{h}^{i-3} + C_{:,w^{i-2}}\right)$
$\tilde{h}^{i-2}$
RU
$\varphi\left(U\tilde{h}^{i-2} + C_{:,w^{i-1}}\right)$
Embedding Table
$C_{:,0}$
$C_{:,1}$
$C_{:,2}$
$C_{:,3}$
$C_{:,4}$
$C_{:,5}$
$\vdots$
$\ldots$
w^{i-2}
w_{i-1}

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

2.2 Acausal Language Modeling

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

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

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

2.2.1 Continuous Bag of Words

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

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

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

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 Mikolov et al. (2013b). 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 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:

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

Figure 2.6: CBOW Language Model

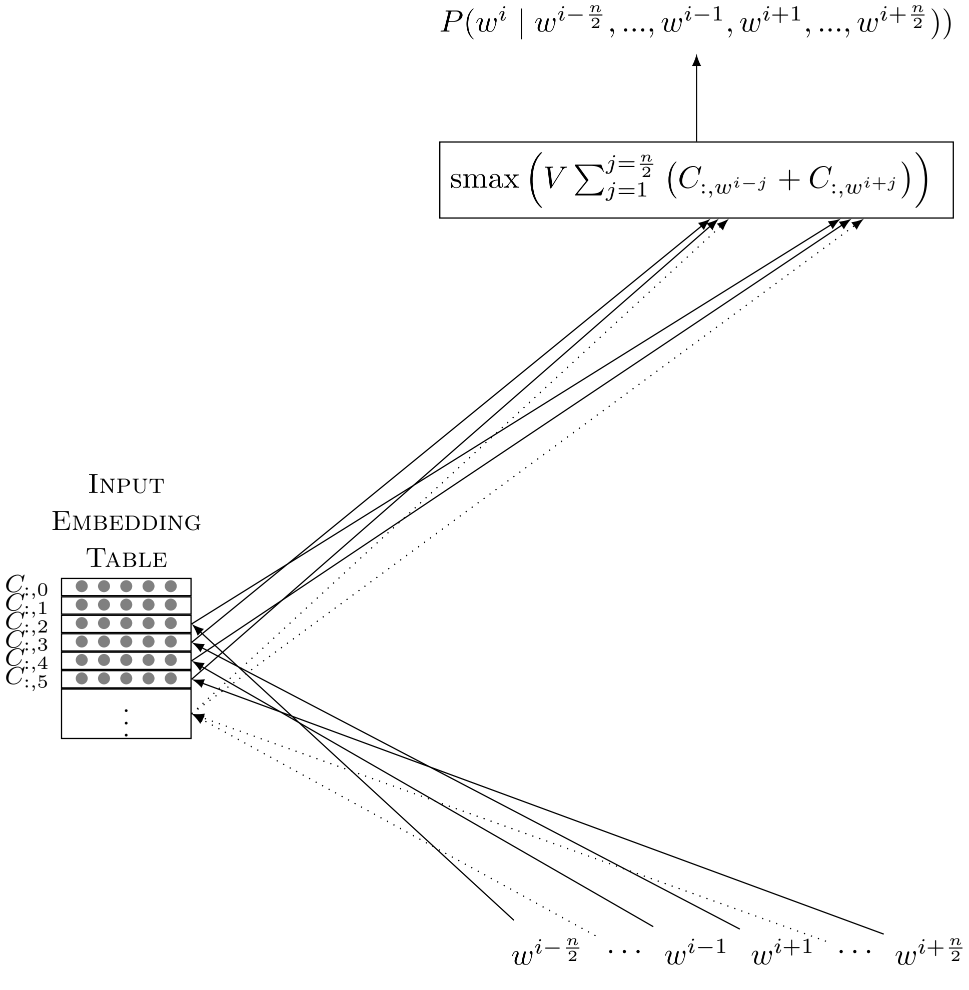

$P(w^i \mid w^{i-\frac{n}{2}}, ..., w^{i-1}, w^{i+1}, ..., w^{i+\frac{n}{2}}))$
$\text{smax}\left(V \sum_{j=1}^{j=\frac{n}{2}} \left(C_{:,w^{i-j}} + C_{:,w^{i+j}}\right)\right)$
Input Embedding Table
$C_{:,0}$
$C_{:,1}$
$C_{:,2}$
$C_{:,3}$
$C_{:,4}$
$C_{:,5}$
$w^{i-\frac{n}{2}}$
$\cdots$
w^{i-1}
w^{i+1}
$\cdots$
$w^{i+\frac{n}{2}}$

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.

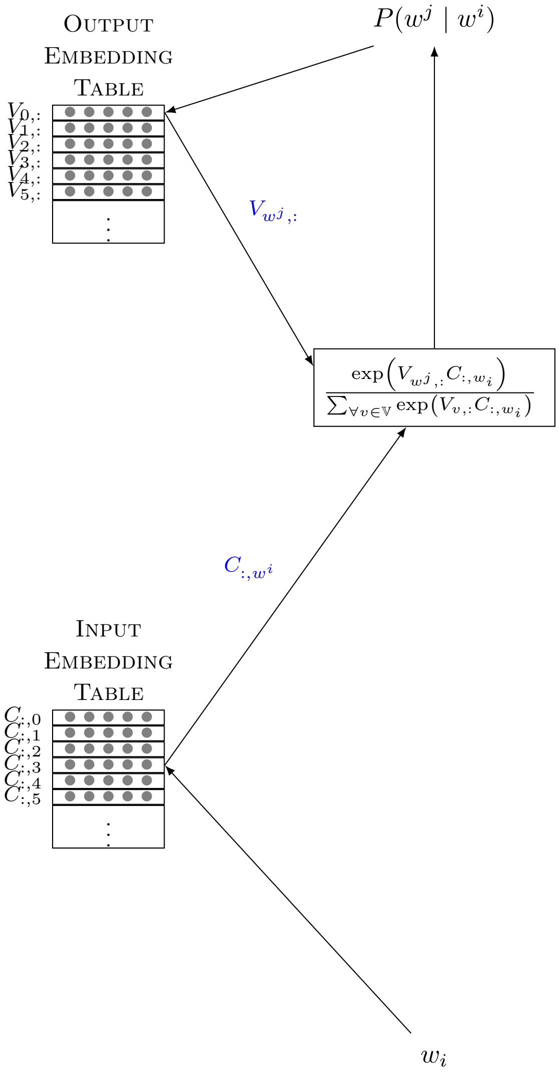


Figure 2.8: Example of analogy algebra

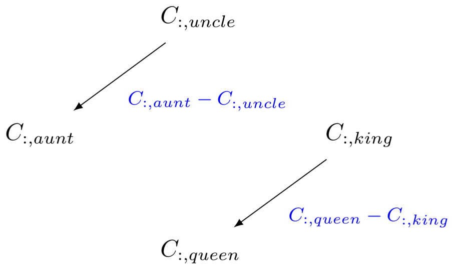
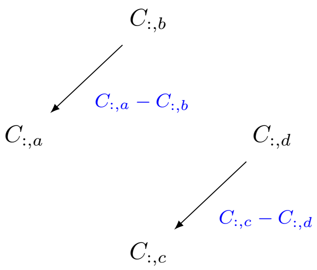


Figure 2.9: Vectors involved in analogy ranking tasks, this may help to understand the math in Equation (2.19)



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:

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

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

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

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

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

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 $\mathbb{V}$:

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

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

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

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

2.3 Co-location Factorisation

2.3.1 GloVe

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

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

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

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

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

$$\begin{aligned} \text{optimise} \quad & s(w^i, w^j) && = V_{w^j,:}\, C_{:,w^i} + \tilde{b}_{w^i} + \tilde{k}_{w^j} && (2.21) \\ \text{such that} \quad & s(w^i, w^j) && \approx \log(X_{w^i,w^j}) && (2.22) \end{aligned}$$

This is done via the minimisation of

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

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

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

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

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

2.3.2 Further equivalence of Co-location Prediction to Factorisation

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

2.3.3 Conclusion

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

2.4 Hierarchical Softmax and Negative Sampling

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

2.4.1 Hierarchical Softmax

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

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

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

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

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

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

An incredibly gentle introduction to hierarchical softmax

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

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

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 $\tilde{z}$.

Figure 2.10: Tree for 2 words

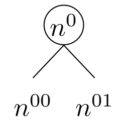
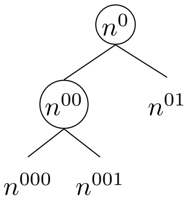


Figure 2.11: Tree for 3 words



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:

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

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

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 $[0t^1t^2t^3 \ldots t^L]$, e.g. $[010 \ldots 1]$, its probability can be found recursively as:

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

Thus:

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

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

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

The conditional probability chain rule, is applied to get:

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

Formulation

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

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

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

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

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

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

Loss Function

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

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

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

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

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

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

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

2.4.2 Negative Sampling

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

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

A Motivation of Negative Sampling

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

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

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

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.

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

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

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

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

where $D^{1\mathrm{g}}$ stands for the unigram distribution of the vocabulary and $\mathrm{samples}(D^{1\mathrm{g}})$ 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 ($P^{1\mathrm{g}}(w^k)$) gives:

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

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:

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

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

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

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 $D^{1\mathrm{g}}$ with D^{ns} being the distribution of negative samples from the vocabulary, to be specified as a hyper-parameter of training.

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

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

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:

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

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.

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

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

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

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

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

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

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

2.5 Natural Language Applications – beyond language modeling

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

2.5.1 Using Word Embeddings as Features

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

2.6 Aligning Vector Spaces Across Languages

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

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

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

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

Dhillon, Foster, and Ungar (2011) investigated using this to create word-embeddings. We noted in Equation (2.16), that skip-gram maximise the similarity of the output and input embeddings according to the dot-product. CCA also maximises similarity (according the correlation), between the vectors from one set, and the vectors for another. As such given representations for two words from the same context, initialised randomly, CCA could be used repeatedly to optimise towards good word embedding capturing shared meaning from contexts. This principle was used by Dhillon, Foster, and Ungar (2011), though their final process more complex than described here. It is perhaps one of the more 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 Shi et al. (2015), amongst others. Similarly, on the sentence embedding side Zou et al. (2013), and Socher et al. (2014) train embeddings from different languages and modalities (respectively) directly to be near to their partners (these are discussed in Chapter 4). A survey paper on such methods was recently published by Ruder (2017).

Chapter 3

Word Sense Representations

This chapter originally appears as Chapter 4 of the book Neural Representations of Natural Language, published by by Springer.

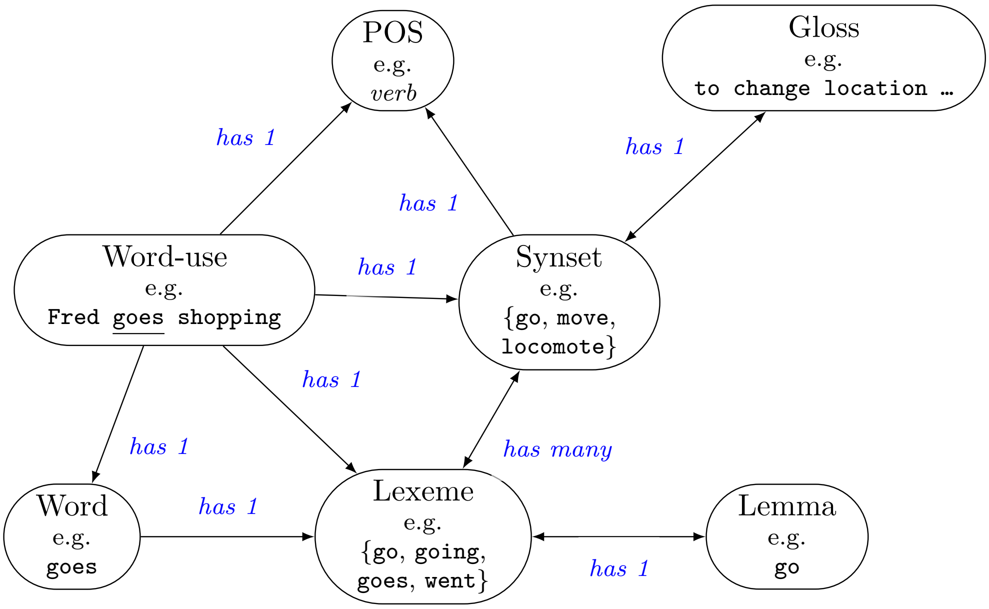
1a. *In a literal, exact, or actual sense; not figuratively, allegorically, etc.*
1b. *Used to indicate that the following word or phrase must be taken in its literal sense, usually to add emphasis.*
1c. *colloq. Used to indicate that some (frequently conventional) metaphorical or hyperbolical expression 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

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.

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.



3.1 Word Senses

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

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

3.1.1 Word Sense Disambiguation

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

Progress on the problem is made difficult by several factors.

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

There is only a relatively small amount of labelled data for word sense problems.

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

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

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

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

Most Frequent Sense

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

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

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

3.2 Word Sense Representation

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

3.2.1 Directly supervised method

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

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

3.2.2 Word embedding-based disambiguation method

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

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

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

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

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

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

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

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

3.3 Word Sense Induction (WSI)

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

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

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

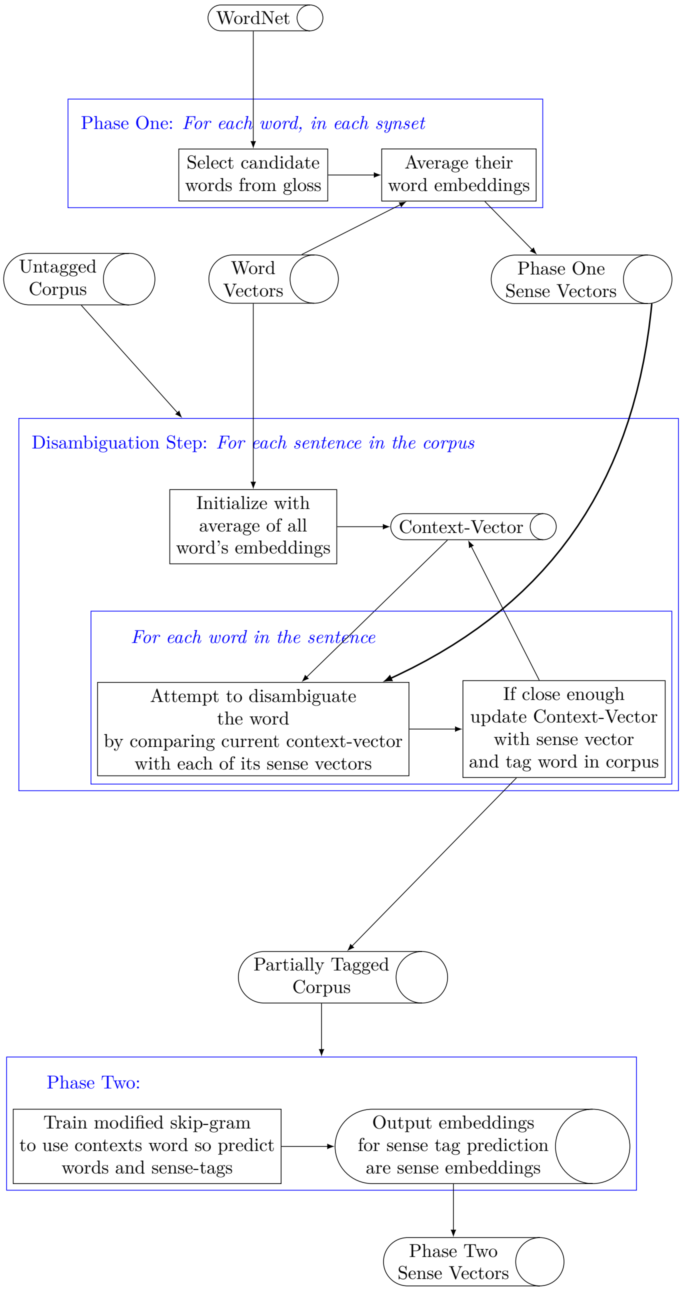
3.3.1 Context Clustering-based Approaches

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

Offline clustering

The fundamental method for most clustering-based approaches is as per Schütze (1998). That original work is not a neural word sense embedding, however the approach remains the same. Pantel and Lin (2002) and Reisinger and Mooney (2010)

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


WordNet
Phase One: For each word, in each synset
Select candidate words from gloss
Average their word embeddings
Untagged Corpus
Word Vectors
Phase One Sense Vectors
Disambiguation Step: For each sentence in the corpus
Initialize with average of all word's embeddings
Context-Vector
For each word in the sentence
Attempt to disambiguate the word by comparing current context-vector with each of its sense vectors
If close enough update Context-Vector with sense vector and tag word in corpus
Partially Tagged Corpus
Phase Two:
Train modified skip-gram to use contexts word so predict words and sense-tags
Output embeddings for sense tag prediction are sense embeddings
Phase Two Sense Vectors

are also not strictly neural word embedding approaches (being more classical vector representations), however the overall method is also very similar.

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

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

- Schütze (1998) uses variations of inverse-document-frequency (idf) weighted bags of words, including applying dimensionality reduction to find a dense representation.

- Pantel and Lin (2002) use the mutual information vectors between words and their contexts.

- Reisinger and Mooney (2010), use td-idf or χ^2 weighted bag of words.

- Huang et al. (2012) uses td-idf weighted averages of (their own) single sense word embeddings for all words in the context.

- Kågebäck et al. (2015) also uses a weighted average of single sense word skip-gram embeddings, with the weighting based on two factors. One based on how close the words were, and the other on how likely the co-occurrence was according to the skip-gram model.

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

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

- Schütze (1998) uses a group average agglomerative clustering method.

- Pantel and Lin (2002) use a custom hierarchical clustering method.

- Reisinger and Mooney (2010) use mixtures of von-Mises-Fisher distributions.

- Huang et al. (2012) use spherical k-means.

- Kågebäck et al. (2015) use k-means.

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

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

Online clustering

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

Kågebäck et al. (2015) investigated using their weighting function (as discussed in Section 3.3.1) with the online clustering used by Neelakantan et al. (2015). They found that this improved the quality of the representations. More generally any such weighting function could be used. This online clustering approach is loosely similar to the co-location prediction-based approaches.

3.3.2 Co-location Prediction-based Approaches

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

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

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

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

We can thus rewrite Equation (3.1) as:

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

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

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

w^i, and context word w^o, with $s^l \in \mathcal{S}(w^i)$, formally we have:

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

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

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

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

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

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

Considering again the form of Equation (3.2)

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

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

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

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

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

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

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

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

3.4 Conclusion

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

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

Chapter 4

Sentence Representations and Beyond

This chapter originally appears as Chapter 5 of the book Neural Representations of Natural Language, published by by Springer.

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

Abstract

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

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

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

4.1 Unordered and Weakly Ordered Representations

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

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

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

4.1.1 Sums of Word Embeddings

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

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

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

The natural analogy to this with word embeddings might seem to be to find n-gram embeddings by the concatenation of n word embeddings; and then to sum these. However, such a sum is less informative than it might seem. As the sum in each concatenated section is equal to the others, minus the edge words.

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

4.1.2 Paragraph Vector Models (Defunct)

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

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

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

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

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

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

4.2 Sequential Models

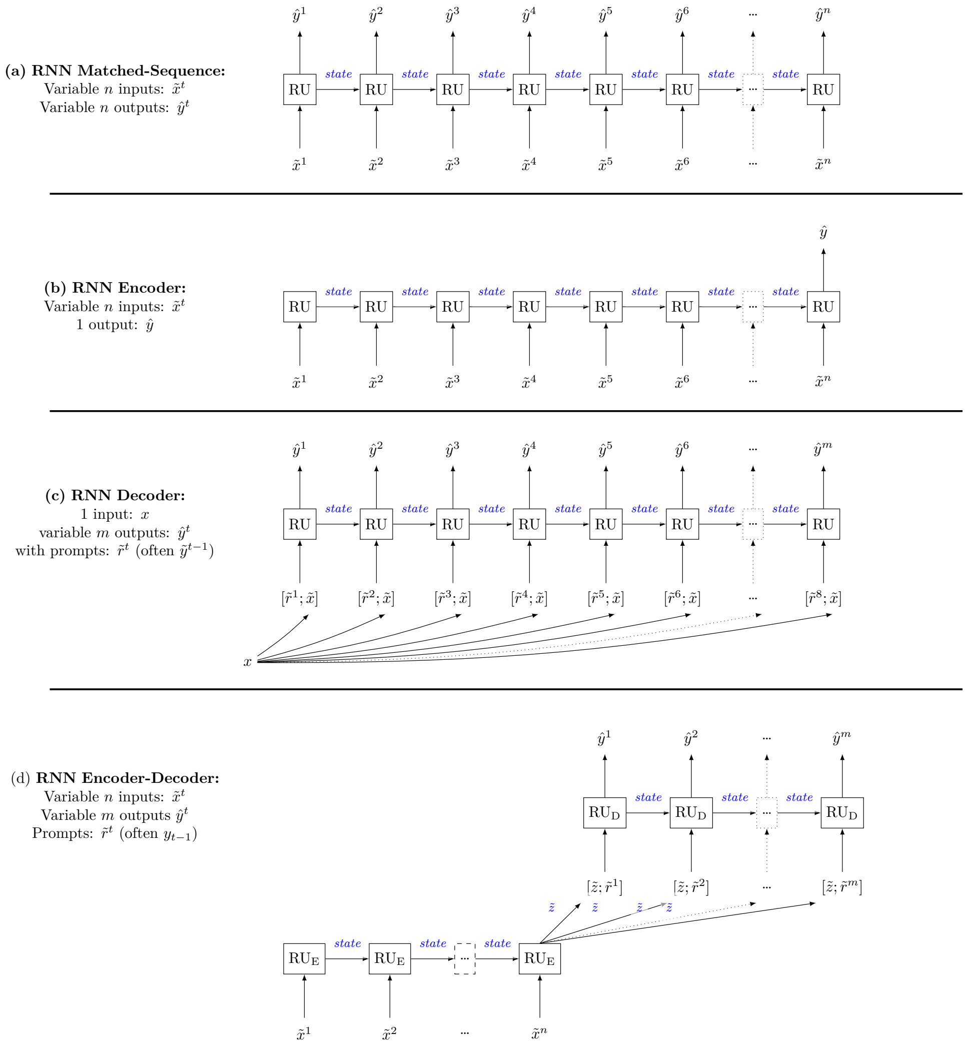
The majority of this section draws on the recurrent neural networks (RNN) as discussed in Chapter 2 of White et al. (2018a). Every RNN learns a representation of all its input and output in its state. We can use RNN encoders and decoders (as shown in Figure 4.1) to generate representations of sequences by extracting a coding layer. One can take any RNN encoder, and select one of the hidden state layers after the final recurrent unit (RU) that has processed the last word in the sentence. Similarly for any RNN decoder, one can select any hidden state layer before the first recurrent unit that begins to produce words. For an RNN encoder-decoder, this means selecting the hidden layer from between. This was originally considered in Cho et al. (2014), when using a machine translation RNN, to create embeddings for the translated phrases. Several other RNNs have been used in this way since.

4.2.1 VAE and encoder-decoder

Bowman et al. (2016b) presents an extension on this notion, where in-between the encode and the decode stages there is a variational autoencoder (VAE). This is shown in Figure 4.2. The variational autoencoder (Kingma and Welling 2014) has been demonstrated to have very good properties in a number of machine learning applications:

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

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.



(a) RNN Matched-Sequence:
Variable n inputs: $\tilde{x}^t$
Variable n outputs: $\hat{y}^t$

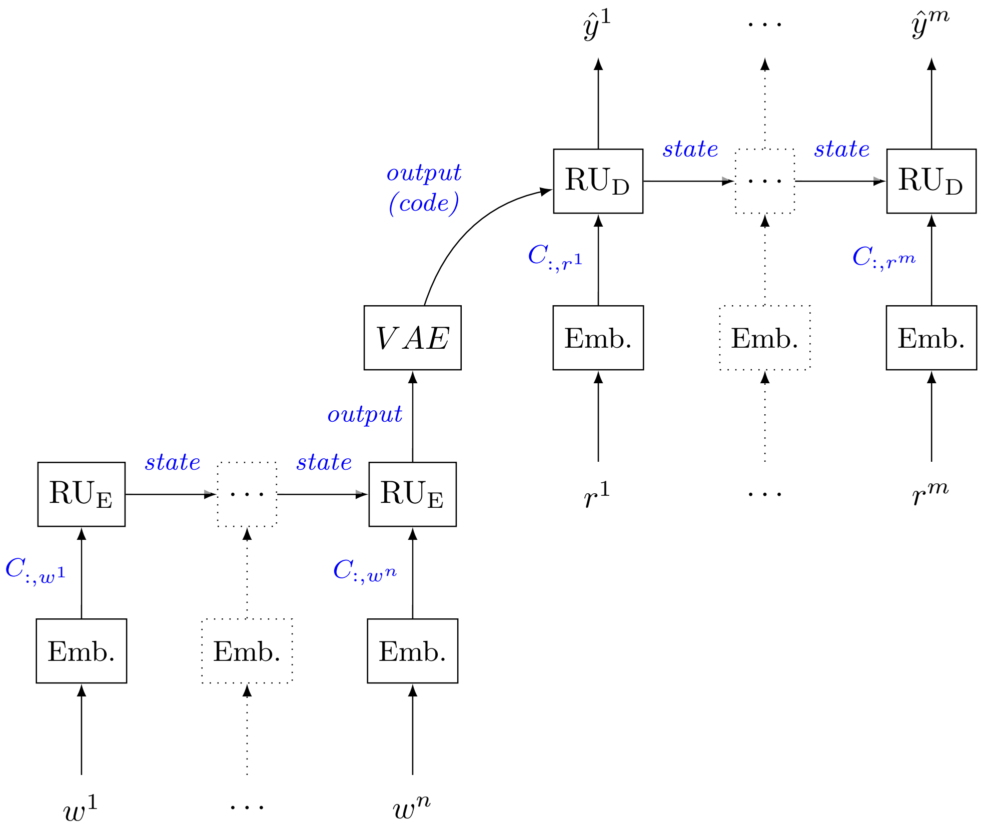
(b) RNN Encoder:
Variable n inputs: $\tilde{x}^t$
1 output: $\hat{y}$

(c) RNN Decoder:
1 input: x
variable m outputs: $\hat{y}^t$
with prompts: $\tilde{r}^t$ (often $\hat{y}^{t-1}$)

(d) **RNN Encoder-Decoder:**
Variable n inputs: $\tilde{x}^t$
Variable m outputs $\hat{y}^t$
Prompts: $\tilde{r}^t$ (often y_{t-1})

$\hat{y}^1$ $\hat{y}^2$ $\hat{y}^3$ $\hat{y}^4$ $\hat{y}^5$ $\hat{y}^6$ ... $\hat{y}^n$
RU
state
...
$\tilde{x}^1$ $\tilde{x}^2$ $\tilde{x}^3$ $\tilde{x}^4$ $\tilde{x}^5$ $\tilde{x}^6$ ... $\tilde{x}^n$
$\hat{y}$
$\hat{y}^m$
$[\tilde{r}^1; \tilde{x}]$ $[\tilde{r}^2; \tilde{x}]$ $[\tilde{r}^3; \tilde{x}]$ $[\tilde{r}^4; \tilde{x}]$ $[\tilde{r}^5; \tilde{x}]$ $[\tilde{r}^6; \tilde{x}]$ ... $[\tilde{r}^8; \tilde{x}]$
x
$\mathrm{RU_D}$
$\mathrm{RU_E}$
$[\tilde{z}; \tilde{r}^1]$ $[\tilde{z}; \tilde{r}^2]$ ... $[\tilde{z}; \tilde{r}^m]$
$\tilde{z}$
$\tilde{x}^1$ $\tilde{x}^2$ ... $\tilde{x}^n$

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



they are able to work to find continuous latent variable distributions over arbitrary likelihood functions (such as in the neural network); and are very fast to train. Using the VAE, it is hoped that a better representation can be found for the sequence of words in the input and output.

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

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

4.2.2 Skip-thought

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

That output code is the representation of the sentence.

4.3 Structured Models

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

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

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

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

4.3.1 Constituency Parse Tree (Binary)

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

$$\begin{aligned} f^{\mathrm{RV}}(\tilde{u},\tilde{v}) &= \varphi\left(\begin{bmatrix} S & R \end{bmatrix} \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} + \tilde{b}\right) \qquad (4.3) \\ &= \varphi\left(S\tilde{u} + R\tilde{v} + \tilde{b}\right) \qquad (4.4) \end{aligned}$$

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.

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

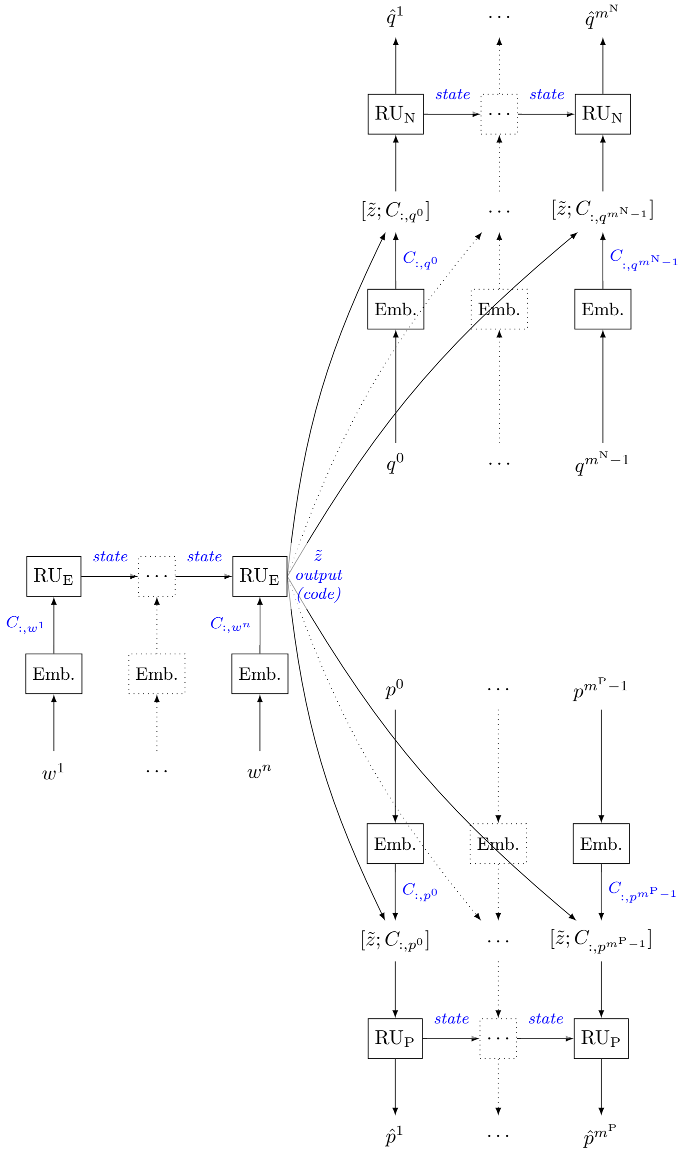


Figure 4.4: A constituency parse tree for the sentence: `This is a simple example of a parse tree`. In this diagram the leaf nodes are the input words, their intimidate parents are their POS tags, and the other nodes with multiple children represent subphrases of the sentence, for example NP is a Noun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

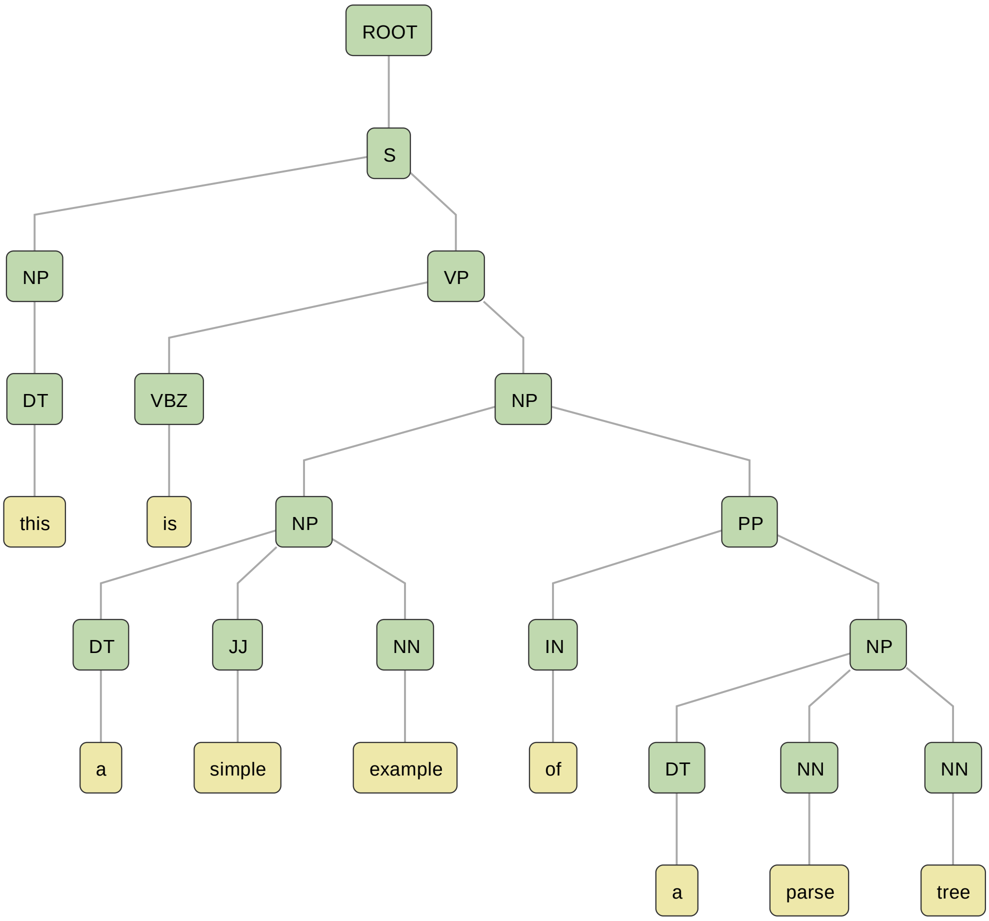
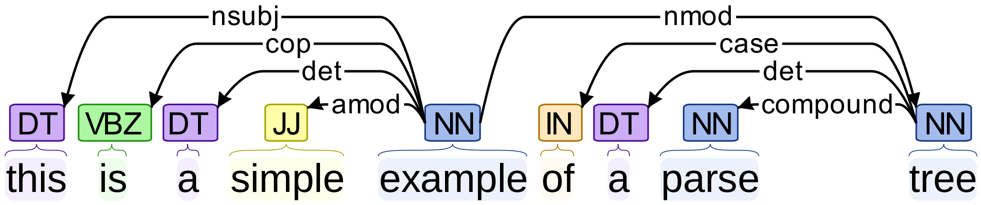


Figure 4.5: A dependency parse tree for the sentence `This is a simple example of a parse tree`, This flattened view may be misleading. `example` is at the peak of the tree, with direct children being: `this`,`is`,`a`,`simple`, and `tree`. `tree` has direct children being: `of`,`a`, and `parse`.



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

4.3.2 Dependency Tree

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

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

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

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

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

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

4.3.3 Parsing

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

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

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

4.3.4 Recursive Autoencoders

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

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

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

Semi-supervised

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

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

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

4.4 Matrix Vector Models

4.4.1 Structured Matrix Vector Model

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

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

$$\begin{aligned} f^{\mathrm{RV}}(\tilde{u}, \tilde{v}, U, V) &= \varphi\left([S\ R][U\tilde{v}; V\tilde{u}] + \tilde{b}\right) && (4.6)\\ &= \varphi\left(S\,U\tilde{v} + R\,V\tilde{u} + \tilde{b}\right) && (4.7)\\ F^{\mathrm{RV}}(U, V) &= W\,[U; V] = W^{\mathrm{l}}U + W^{\mathrm{r}}V && (4.8) \end{aligned}$$

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

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

4.4.2 Sequential Matrix Vector Model

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

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

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

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

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

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

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

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

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

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

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

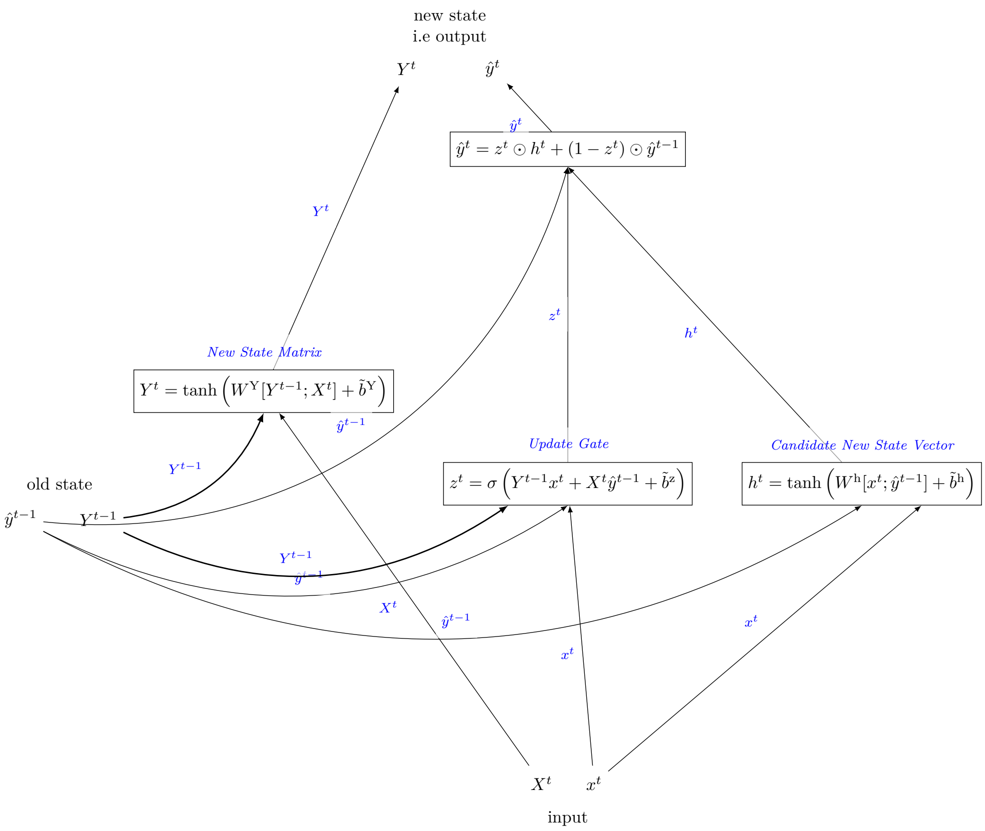
4.5 Conclusion, on compositionality

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

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

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

Figure 4.6: A Matrix Vector recurrent unit


new state
i.e output
Y^t
$\hat{y}^t$
$\hat{y}^t$
$\hat{y}^t = z^t \odot h^t + (1 - z^t) \odot \hat{y}^{t-1}$
Y^t
z^t
h^t
New State Matrix
$Y^t = \tanh\left(W^{\mathrm{Y}}[Y^{t-1}; X^t] + \tilde{b}^{\mathrm{Y}}\right)$
$\hat{y}^{t-1}$
Update Gate
Candidate New State Vector
Y^{t-1}
old state
$z^t = \sigma\left(Y^{t-1}x^t + X^t\hat{y}^{t-1} + \tilde{b}^{\mathrm{z}}\right)$
$h^t = \tanh\left(W^{\mathrm{h}}[x^t; \hat{y}^{t-1}] + \tilde{b}^{\mathrm{h}}\right)$
$\hat{y}^{t-1}$
Y^{t-1}
Y^{t-1}
$\hat{y}^{t-1}$
X^t
$\hat{y}^{t-1}$
x^t
x^t
X^t
x^t
input

tasks, this implicitly shows that they have sufficient representational capacity to learn sufficient degrees of compositional processing to accomplish these tasks.

In fact, it has been suggested that even some unordered models such as sum of word embeddings are able to capture some of what would be thought of as compositional information. Ritter et al. (2015) devised a small corpus of short sentences describing containing relationships between the locations of objects. The task and dataset was constructed such that a model must understand some compositionality, to be able to classify which relationships were described. Ritter et al. (2015) tested several sentence representations as the input to a naïve Bayes classifier being trained to predict the relationship. They found that when using sums of high-quality word embeddings as the input, the accuracy not only exceeded the baseline, but even exceeded that from using representation from a structural model. This suggests that a surprising amount of compositional information is being captured into the embeddings; which allows simple addition to be used as a composition rule. Though it being ignorant of word order does mean it certainly couldn't be doing so perfectly, however the presence of other words my be surprisingly effective hinting at the word order (White et al. 2016b), thus allow for more apparently compositional knowledge to be encoded than is expected.

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

Part III

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 Iyyer, Boyd-Graber, and Daumé III 2014. 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 Le and Mikolov 2014; recreation of input, as in the auto-encoders of Socher et al. 2011c; Socher et al. 2011a and Iyyer, Boyd-Graber, and Daumé III 2014; and syntactic structural classification Socher et al. 2013a; Socher, Manning, and Ng 2010. 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 Socher et al. 2011c; Socher et al. 2012; Le and Mikolov 2014 and paraphrase detection Socher et al. 2011a. 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) Dolan and Brockett 2005 and Opinosis Ganesan, Zhai, and Han 2010 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 Gershman and Tenenbaum 2015 and of Ritter et. al. Ritter et al. 2015. 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. Socher et al. 2011a; and the two word content predictive models, Distributed Memory (PV-DM) and Distributed Bag of Words by Le and Mikolov Le and Mikolov 2014. In addition to these advanced sentence embedding models, a simple average of word embeddings, from Mikolov et. al. Mikolov et al. 2013a, 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) Socher et al. 2011a 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) Le and Mikolov 2014 method is based on an extension of the Continuous Bag-of-Words word-embedding model Mikolov et al. 2013b. 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 Le and Mikolov 2014. 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) Le and Mikolov 2014, is based on the Skip-gram model for word-embeddings, also from Mikolov et al. 2013b. 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 Le and Mikolov 2014. The sum of word embeddings first considered in Mikolov et al. 2013a for short phrases, it was found to be an effective model for summarization in Kågebäck et al. 2014. 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 down-stream.

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, Dolan and Brockett 2005 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 Socher et al. 2011c and for the paragraph vector models in Le and Mikolov 2014. Sentiment Analysis is classifying a text as positive or negative, or assigning a score as in the Sentiment Treebank Socher et al. 2013b. 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 Maron 1961 and Borko and Bernick 1963. Much more recently it has been used to assess the convolution neural networks of Zhang and LeCun 2015, 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 downsteam 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.Mikolov, Yih, and Zweig 2013 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 $\mathtt{King} - \mathtt{Man} + \mathtt{Woman} \approxeq \mathtt{Queen}$, As man is to woman, king is to queen. In the rating task as described by Jurgens et al. 2012, 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 taskMikolov, Yih, and Zweig 2013. 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.

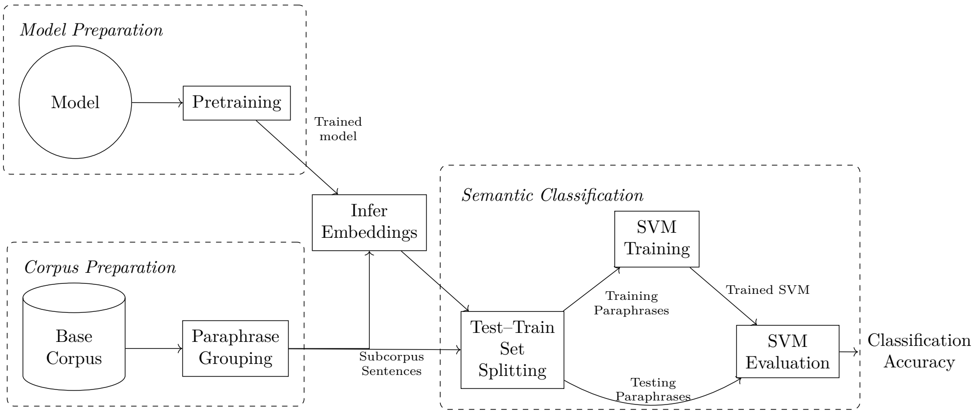


Figure 5.1: Process Diagram for the Evaluation of Semantic Consistency via our method

Gershman et. al. Gershman and Tenenbaum 2015, 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. Ritter et al. 2015, 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 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) Dolan and Brockett 2005 and of the Opinosis Ganesan, Zhai, and Han 2010 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 paperSocher et al. 2011a. It is initialized on the unsupervised Collobert and Weston word embeddingsCollobert and Weston 2008, 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 Rehůrek and Sojka 2010 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.

[1] http://www.socher.org/index.php/Main/DynamicPoolingAndUnfoldingRecursiveAutoencoders-ForParaphraseDetection

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

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 Mikolov et al. 2013a. 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 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 Pedregosa et al. 2011 was used to orchestrate the cross-validation and to interface with the LibLinear SVM implementation Fan et al. 2008. 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 Dolan and Brockett 2005. 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.

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

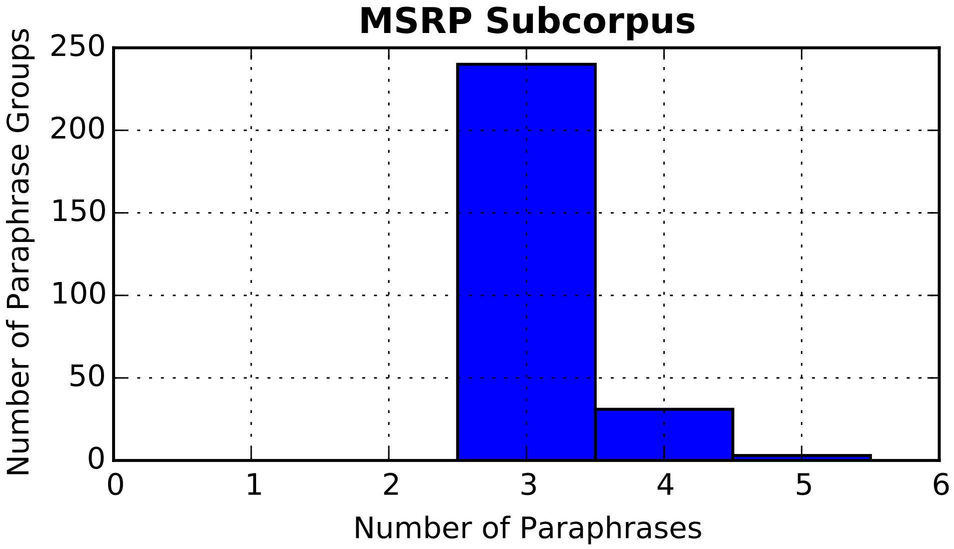


Figure 5.2: Break down of how many paraphrases groups are present in the MSRP subcorpus of which sizes.It contains a total of 859 unique sentences, broken up into 273 paraphrase groups.

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 CorpusGanesan, Zhai, and Han 2010 was used as secondary source of original real-world text. It is sourced from several online review sites: 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.

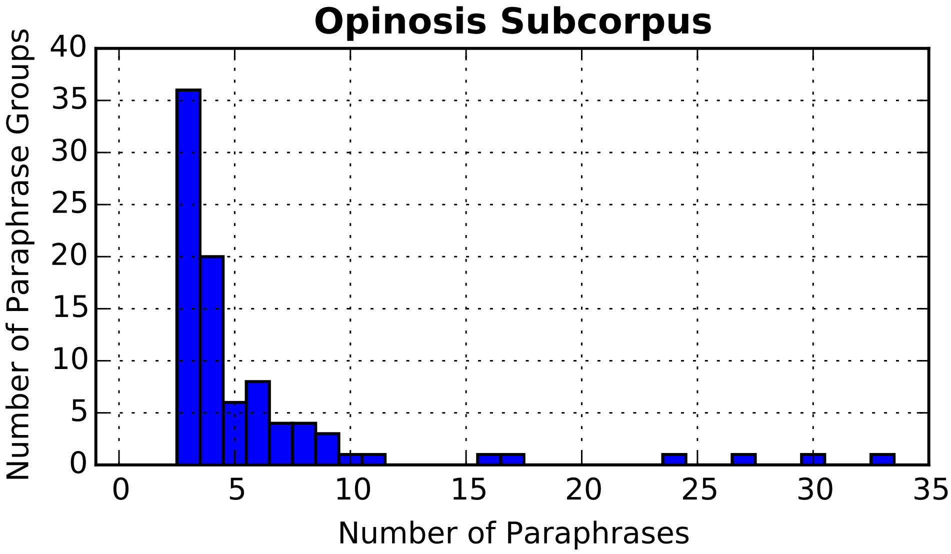


Figure 5.3: Break down of how many paraphrases groups are present in the Opinosis subcorpus of which sizes. It contains a total of 521 unique sentences, broken up into 89 paraphrase groups.

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 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." Dolan and Brockett 2005. 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 distinct classes in the MSRP subcorpus. The Opinosis subcorpus draws a finer line between sentences with similar meanings. As discussed

[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.

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 Kågebäck et al. 2014 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 Socher et al. 2011a. The exact same model from Socher et al. 2011a 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 Socher et al. 2011a, 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.

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

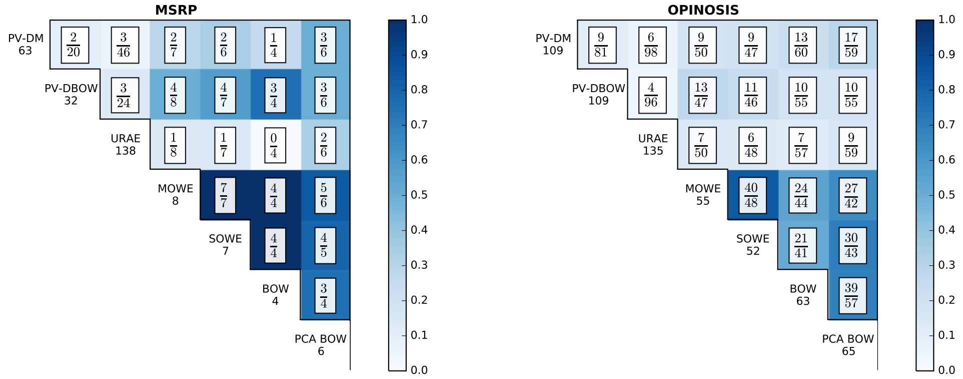


Figure 5.4: The misclassification agreement between each of the models for the MSRP (left) and Opinosis (right) subcorpora. Below each model name is the total mistakes made. The denominator of each fraction is the number of test cases incorrectly classified by both models. The numerator is the portion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

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 Yin and SchÃƒÆ’Ã†â€™Ãƒâ€šÃ‚Â¼tze 2015 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 hyper-parameters 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.Ritter et al. 2015 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 Ritter et al. 2015 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.Ritter et al. 2015. 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

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.

6.1 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 (Elson, Dames, and McKeown 2010; Wohlgenannt, Chernyak, and Ilvovsky 2016); to align scenes in movies with those from books (Zhu et al. 2015); 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"

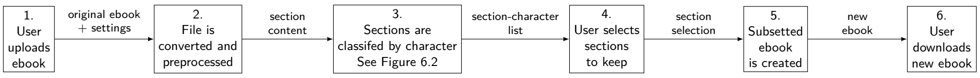

1. User uploads ebook
original ebook + settings
2. File is converted and preprocessed
section content
3. Sections are classifed by character See Figure 6.2
section-character list
4. User selects sections to keep
section selection
5. Subsetted ebook is created
new ebook
6. User downloads new ebook

Figure 6.1: The full NovelPerspective pipeline. Note that step 5 uses the original ebook to subset.

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 (Booth 1961). 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 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.

6.2 Character Classification Systems

The full NovelPerspective pipeline is shown in Figure 6.1. The core character classification step (step 3), is detailed in Figure 6.2. 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

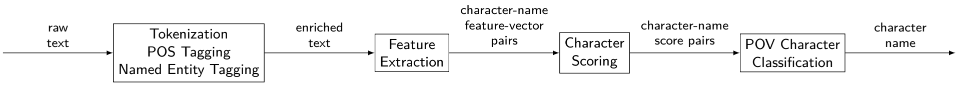


Figure 6.2: The general structure of the character classification systems. This repeated for each section of the book during step 3 of the full pipeline shown in Figure 6.1.

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 Imani et al. (2017).

6.2.1 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 6.5), and used to gauge the performance of the more complicated systems in the evaluations presented in Section 6.4.

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.

6.2.2 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 6.2.2). 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 6.2. 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 (Bojanowski et al. 2017). 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 (White et al. 2015; Mikolov et al. 2013a). 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

[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 6.1: The number of chapters and point of view characters for each dataset.

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.

6.3 Experimental Setup

6.3.1 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 6.4 we consider the use of each as a training and testing dataset. In the online demonstration (Section 6.5), 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 6.1. 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.

6.3.2 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.

6.3.3 Implementation

The full source code is available on GitHub. [3] Scikit-Learn (Pedregosa et al. 2011) is used for the machine learning and evaluations, and NLTK (Bird and Loper 2004) is used for textual preprocessing. The text is tokenised, and tagged with POS and named entities using NLTK's default methods. Specifically, these are the Punkt sentence tokenizer, the regex-based improved TreeBank word tokenizer, greedy averaged perceptron POS tagger, and the max-entropy binary named entity chunker. The use

[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 6.2: The results of the character classifier systems. The best results are **bolded**.

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.

6.4 Results and Discussion

Our evaluation results are shown in Table 6.2 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.

It is surprising that using the combination of two training sets does not always outperform 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 6.5), 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

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

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 6.3 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).

6.5 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] allows the user to apply any of the model discussed to their own novels.

The web-app functions as shown in Figure 6.1. 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.

6.6 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

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

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.

Part IV

Conclusion

Chapter 7

Conclusion

Current research in natural language understanding relies on creating computer manipulatable representations of natural language for purposes of making inferences about meaning.

While the normal machine learning adage that given enough data and a model with sufficiently high representational capacity any problem can be solved always applies, 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 various low-medium resource settings where this will be an ideal method for many problems.

The research presented here on linear combinations of embeddings has shown that this simple input representation technique is surprisingly powerful. This surprising power comes from the value of surface level information in many practical natural language understanding tasks. The word content being the most obvious of these. This word content is presented in a dense and informative form in a LCOWE; in a way that captures and preserves lexical similarity information. While the LCOWE loses word order information, it preserves the word content very well. That content proved very useful for the tasks considered in this research.

We considered a number of tasks to identify the utility of this representation for different uses. Chapter 5 investigated classifying paraphrases as means to investigate quality of SOWE as a sentence embedding method. **??** defined models for color estimation from short phrases. **??** considered if we could use weighted combinations of sense embeddings to better capture the sense used in a particular example. Chapter 6 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 for the important category of sentence representations. **??** demonstrated a method that could partially recover bags of words. **??** extended this work by attempting to order those bags of words into sentences. This demonstrated that a surprising amount of information is still available in the summed embeddings; which helps to explain why they work so well.

While it is clear that such linear combinations of embedding representation spaces are not perfect for representing all meanings, they are pragmatically very strong. They do not encode any information about word order. It is thus clear that there exist 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 a restricted context. 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 the left of the orange.` vs `The orange is`

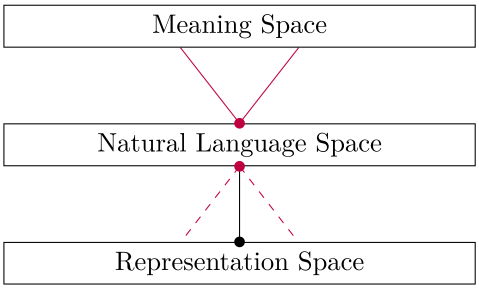


Figure 7.1: The representation space is a computationally manipulate representation of the meaning space. The natural language utterances come from points in the meaning space; though due to ambiguity we can only truly hope to estimate distributions when the interpret them. A single point embedding as an approximation to a distribution with a single tight peak.

`the left of the banana`. It is desirable that such sentences are nearby in a representational of semantic space.

7.1 Some reflections upon semantic spaces

We can consider that there is a true semantic space of ideas. When speaking, this space is projected down to natural languages space. To again quote Webber: "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 to utterance 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 meaning space that the speaker intends. Some times the natural language space alone is enough to recover a good idea of the the point in meaning space the speaker intends, but other times it is not.

The preimage of a point in natural language space (e.g. a sentence), is a probability distribution over 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 made by the eponymous company. Thus, on its own the sentence `Apples are good.` suggests a distribution with at least two peaks in meaning space. Combine that utterance, with the context of being in a computer store, rather than a grocer, and the probability of one peak 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. Meaning space is a continuous space, with every utterance corresponding to a unique point. It is an uncountably large space. In contrast 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 meaning space are projected down to a single point in natural language space.

When designing a 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 computationally manipulatable using simple operations (like sums). In particular it should have a continuous mapping from to and from embedding space. A neighbourhood in representation space, should correspond to a neighbourhood in meaning space. Chapter 5 investigated this for sentence embeddings. By taking points in natural language space known to come from very nearby points in meaning space, that is to say paraphrases, and checking that they belong to near points in embedding space.

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

This has 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 can't rule out the speaker using the word incorrectly or metaphorically which gives rise to nonzero values elsewhere in meaning space. Word-sense embeddings produce multiple sense embeddings – ideally one corresponding to each peak in meaning space. We know these peaks are only rough approximations to the true point in meaning space for a given usage of a word. **??** attempts find other points in the embeddings space, that better corresponds to the true point in meaning space for the particular use.

Unsupervised methods, in particular word embeddings, but also more generally, are ungrounded. They are based only on 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 natural language space to meaning space. While we 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 showed that it was. 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. It's 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 meaning space. Our results show that that strength is carried forth into linear combinations of such embeddings.

The color understanding task considered in **??** is interesting. It is a typical natural language understanding system, which takes a point in 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 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 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. The fact that even on the subset of the testing data where word order was ambiguous, SOWE was the best performing model highlights an important notion. 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 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 space that fully captures the natural language features, a single point in that representational space corresponds to two points in meaning space; as the single point in natural language space could have come from either point in meaning space. As such, ambiguity from loss of word order is not an unique and unsalvageable problem. If we thus had a distribution over 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 true meaning space, it remains interesting for cases where 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 define prior and thus decrease distributions associated with other ambiguities, we can use language models to provide a prior over those peaks; based on the likelihood of word orders. There exists a trivial extension of the work presented in **????**, where the mixed integer programming model is constrained to give the second (and so forth) most likely solution, together with it's probability. It is however 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 meaning space, it is our belief that further advancements would benefit from fully considering word embeddings and other objects from representation spaces, not as discrete points but as random variables with a linked distribution. This however comes with significant challenges in that manipulating the very high dimensional distributions that are required for this is computationally challenging

ere does para- ph belong?

Chapter 5 and **??** both consider represent contagious linguistic structures – sentences and shore phrases respectively, as input representations. Further, the output of the output modules discussed in **??** is a grounded representation space, though that work did not examine it directly. **??** considers the representation of word senses, and it navigates this representation space to find new representations which better describe a particular use of a word. Chapter 6 is more atypical: the features sets considered for the point of view detection task, characterize how a particular named entity token was used throughout a chapter. It is thus not a representation of the chapter meaning, as it varies for the different named entities. It is in fact a representation of how that named entity is related to the events described in the chapter. It seems like the MOWE feature set used in Chapter 6 it is vastly insufficient to represent such information given it is only a mean of the immediately adjacent words – nothing like the whole chapter's contents. However, as was demonstrated by the classical feature set considered in the same chapter, which primarily consisted of adjacent part of speech tag counts, and which also achieved very good results, surface information encodes a surprising amount of semantic depth.

7.2 Future work

7.2.1 Adversarial Test cases

It is worth consideration, that adversarial test cases allow advancement of the state of the art to increase the capacity of models to represent all possible inputs. However, understanding how common they are are is essential.

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 adversarial such a challenging case is. In **??**, 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.

7.3 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 excellent ability to capture word content; Pure language models have no ability to capture word content, but excellent ability to capture word order. Language modelling based tasks incorporating a representation stage, such as encoder-decoders (Cho et al. 2014), do not capture word content as well as LCOWE (Conneau et al. 2018). They do, however, have state of the art order order representation.

I would rea
like to talk
about how
word embe
dings are u
ful for capt
evaluation
here.

Bibliography

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

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

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

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

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

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah (2014). “Julia: A Fresh Approach to Numerical Computing”. In: arXiv: `1411.1607 [cs.MS]`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Elson, David K., Nicholas Dames, and Kathleen R. McKeown (2010). “Extracting Social Networks from Literary Fiction”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. ACL ’10. Uppsala, Sweden: Association for Computational Linguistics, pp. 138–147.

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

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

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

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

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

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

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

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

Goodman, Alyssa, Alberto Pepe, Alexander W. Blocker, Christine L. Borgman, Kyle Cranmer, Merce Crosas, Rosanne Di Stefano, Yolanda Gil, Paul Groth, Margaret Hedstrom, David W. Hogg, Vinay Kashyap, Ashish Mahabal, Aneta Siemiginowska, and Aleksandra Slavkovic (2014). “Ten Simple Rules for the Care and Feeding of

Scientific Data”. In: *PLOS Computational Biology* 10.4, pp. 1–5. DOI: 10.1371/journal.pcbi.1003542.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

White, Lyndon, Roberto Togneri, Wei Liu, and Mohammed Bennamoun (2016b). "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed

Integer Programming Problem”. In: *IEEE International Conference on Data Mining: High Dimensional Data Mining Workshop (ICDM: HDM)*. DOI: 10.1109/ICDMW.2016.0113.

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

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

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

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

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

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

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

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

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

Yin, Wenpeng and Hinrich SchÃfÆ’Ã†â€™Ãfâ€šÃ‚Â¼tze (2015). “Learning Word Meta-Embeddings by Using Ensembles of Embedding Sets”. In: eprint: 1508.04257.

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

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

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

Zipf, George Kingsley (1945). “The meaning-frequency relationship of words”. In: *The Journal of general psychology* 33.2, pp. 251–256.

Zipf, G.K. (1949). *Human behavior and the principle of least effort: an introduction to human ecology*. Addison-Wesley Press.

Zou, Will Y, Richard Socher, Daniel M Cer, and Christopher D Manning (2013). “Bilingual Word Embeddings for Phrase-Based Machine Translation.” In: *EMNLP*, pp. 1393–1398.

Part V

Appendix: Tooling

Chapter 8

Overview: Tooling

For the carrying out of this research a number of software packages were created for the Julia programming language. These contribute substantially to the Julia ecosystem for natural language processing, machine learning and more generally data science.

Chapter 9

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.

9.1 Introduction

In the movement for reproducible sciences there have been two key requests upon authors: **1.** Make your research code public, **2.** Make your data public (Goodman et al. 2014). In practice this alone is not enough to ensure that results can be replicated. To get another author's code running on a your own computing environment is often non-trivial. One aspect of this is data setup: how to acquire the data, and how to connect it to the code.

DataDeps.jl simplifies the data setup step for software written in Julia (Bezanson et al. 2014). DataDeps.jl follows the unix philosophy of doing one job well. It allows the code to depend on data, and have that data automatically downloaded as required. It increases replicability of any scientific code that uses static data (e.g. benchmark datasets). It provides simple methods to orchestrate the data setup: making it easy to create software that works on a new system without any user effort. While it has been argued that the direct replicability of executing the author's code is a poor substitute for independent reproduction (Drummond 2009), we maintain that being able to run the original code is important for checking, for understanding, for extension, and for future comparisons.

Vandewalle, Kovacevic, and Vetterli (2009) distinguishes six degrees of replicability for scientific code. The two highest levels require that "The results can be easily reproduced by an independent researcher with at most 15 min of user effort". One can expend much of that time just on setting up the data. This involves reading the instructions, locating the download link, transferring it to the right location,

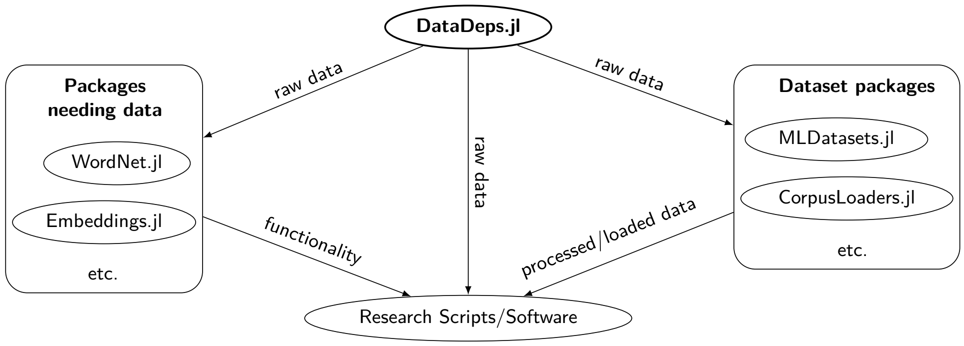


Figure 9.1: The current package ecosystem depending on DataDeps.jl.

extracting an archive, and identifying how to inform the script as to where the data is located. These tasks are automatable and therefore should be automated, as per the practice "Let the computer do the work" (Wilson et al. 2014).

DataDeps.jl handles the data dependencies, while Pkg[1] and BinDeps.jl,[2] (etc.) handle the software dependencies. This makes automated testing possible, e.g., using services such as TravisCI[3] or AppVeyor.[4] Automated testing is already ubiquitous amongst julia users, but rarely for parts where data is involved. A particular advantage over manual data setup, is that automation allow scheduled tests for URL decay (Wren 2008). If the full deployment process can be automated, given resources, research can be fully and automatically replicated on a clean continuous integration environment.

9.1.1 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?

9.2 DataDeps.jl

9.2.1 Ecosystem

DataDeps.jl is part of a package ecosystem as shown in Figure 9.1. 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 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 Section 9.4.1); 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.

[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

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

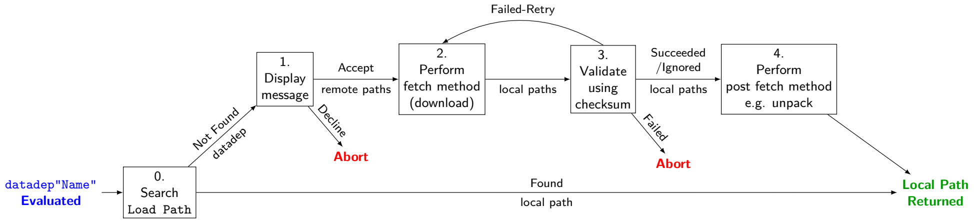


Figure 9.2: The process that is executed when a data dependency is accessed by name.

9.2.2 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 9.2. 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 Section 9.1.1 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.

9.2.3 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 Section 9.1.1. 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.

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

9.2.4 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.

9.3 Availability

9.3.1 Operating system

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

9.3.2 Programming language

Julia v0.6, and v0.7 (1.0 support forthcoming).

9.3.3 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

9.3.4 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

9.4 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.

9.4.1 Case Studies

Research Paper: White et al. (2016a) We criticize our own prior work here, so as to avoid casting aspersions on others. We consider it's limitations and how it would have been improved had it used DataDeps.jl. Two version of the script were provided[8] one with just the source code, and the other also including 3GB of data. It's license goes to pains to explain which files it covers and which it does not (the data), and to explain the ownership of the data. DataDeps.jl would avoid the need to include the data, and would make the ownership clear during setup. Further sharing the source code alone would have been enough, the data would have been downloaded when (and only if) it is required. The scripts themselves have relative paths hard-coded. If the data is moved (e.g. to a larger disk) they will break. Using DataDeps.jl to refer to the data by name would solve this.

Research Tool: WordNet.jl WordNet.jl is the Julia binding for the WordNet tool (Miller 1995). As of PR #8[9] it now uses DataDeps.jl. It depends on having the WordNet database. Previously, after installing the software using the package manager, the user had to manually download and set this up. The WordNet.jl author previously had concerns about handling the data. Including it would inflate the repository size, and result in the data being installed to an unreasonable location. They were also worried that redistributing would violate the copyright. The manual instructions for downloading and extracting the data included multiple points of possible confusion. The gzipped tarball must be correctly extracted. The user must know to pass in the *grand-parent* directory of the database files. Using DataDeps.jl all these issues have now been solved.

Acknowledgements

Thank particularly to Christof Stocker, the creator of MLDatasets.jl (and numerous other packages), in particular for his bug reports, feature requests and code reviews; and for the initial discussion leading to the creation of this tool.

Competing interests

The authors declare that they have no competing interests.

[8] Source code and data provided at `http://white.ucc.asn.au/publications/White2016BOWgen/`

[9] `https://github.com/JuliaText/WordNet.jl/pull/8`

9.5 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 .

Chapter 10

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.

10.1 Summary

DataDepsGenerators.jl is a tool written to help users of the Julia programming language (Bezanson et al. 2014), to observe best practices when making use of published datasets. Using the metadata present in published datasets, it generates the code for the data dependency registration blocks required by DataDeps.jl (White et al. 2018). These registration blocks are effectively executable metadata, which can be resolved by DataDeps.jl to download the dataset. They include a message that is displayed to the user whenever the data set is automatically downloaded. This message should include provenance information on the dataset, so that downstream users know its original source and details on its processing.

DataDepsGenerators.jl attempts to use the metadata available for a dataset to capture and record:

- The dataset name.
- A URL for a website about the dataset.
- The names of the authors and maintainers
- The creation date, publication date, and the date of the most recent modification.
- The license that the dataset is released under.
- The formatted bibliographic details of any paper about or relating to the dataset.
- The formatted bibliographic details of how to cite the dataset itself.
- A list of URLs where the files making up the dataset can be downloaded.
- A corresponding list of file hashes, such as MD5 or SHA256, to validate the files after download.
- A description of the dataset.

Depending on the APIs supported by the repository some of this information may not be available. DataDepsGenerators.jl makes a best-effort attempt to acquire as much provenance information as possible. Where multiple APIs are supported, it makes use of all APIs possible, merging their responses to fill any gaps. It thus often produces higher quality and more comprehensive dataset metadata than is available from any one source.

DataDepsGenerators.jl leavages many different APIs to support a very large number of repositories. By current estimates tens of millions of datasets are supported, from hundreds of repositories. The APIs supported include:

- DataCite / CrossRef
    - This is valid for the majority of all dataset with a DOI.
- DataOne
    - This supports a number of data repositories used in the earth sciences.
- FigShare
    - A popular general purpose data repository.
- DataDryad
    - A data repository particularly popular with evolutionary biology and ecology.
- UCI ML repository
    - A data repository commonly used for small-medium machine learning benchmark datasets.
- GitHub
    - Most well known for hosting code; but is fairly regularly used to host versioned datasets.
- CKAN
    - This is the system behind a large number of government open data initiatives;
    - such as Data.Gov, data.gov.au, and the European Data Portal
- Embedded JSON-LD fragments in HTML pages.
    - This is commonly used on many websites to describe their datasets.
    - Including many of those listed above.
    - But also Zenodo, Kaggle Datasets, all DataVerse sites and many others.

DataDepsGenerators.jl as the name suggests, generates static code which the user can add into their project's julia source code to make use of with DataDeps.jl. There are a number of reasons why static code generation is preferred over directly using the APIs. - On occasion the information reported by the APIs is wrong or incomplete. By generating code that the user may edit they may tweak the details as required. - The process of accessing the APIs requires a number of heavy dependencies, such as HTML and JSON parsers. If these APIs were to be access directly by a project, it would require adding this large dependency tree to the project. - It is important to know if a dataset has changed. As such retrieving the file hash and last modification date would be pointless if they are updated automatically. Finally: having the provenance information recorded in plain text, makes the dataset metadata readily accessible to anyone reading the source code; without having to run the project's application.

The automatic downloading of data is important to allow for robustly replicable scientific code. The inclusion of provenance information is required to give proper credit and to allow for good understanding of the dataset's real world context. DataDepsGenerators.jl makes this easy by automating most of the work.

10.1.1 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.