From Classical to Neural Representations of Natural Language

Lyndon White

April 5, 2018

Contents

I Literature Review 5

1 Introduction 7

2 Neural Representations of Natural Language: Word Representations 9

3 Neural Representations of Natural Language: Word Sense Representations 49

4 Neural Representations of Natural Language: Sentence Representations and Beyond 70

5 Conclusion 97

II Contrasting Classical and Neural Representations 99

6 Introduction 101

7 How Well Sentence Embeddings Capture Meaning 101

8 Novel Perspective 110

9 Learning Distributions of Meant Color 120

III Connecting Classical to Neural Representations 131

10 Generating BOW from SOWE 133

11 Generating Sentences from SOWE 153

12 Finding Word Sense Embeddings of Known Meanings 164

IV Tooling 177

13 Repeatable Data Setup For Reproducible Data Science 179

Part I

Literature Review

Chapter 1

Introduction

I present here, the traditional literature review chapter in a nontrational form: as 3 chapters taken from a book I wrote during my candidature. The book "Neural Representations of Natural Language" is currently available from SpringerBriefs. I include here the three core, non-introductory chapters, in there original manuscript form. This skips over the original preface, and the chapters introducing machine learning and recurrent neural networks.

Neural Representations of Natural Language

Lyndon White [*]
Roberto Togneri [†]
Wei Liu [‡]
Mohammed Bennamoun[§]

March 9, 2018

SpringerBriefs in Computer Science

* lyndon.white@ucc.asn.au
† roberto.togneri@uwa.edu.au
‡ wei.liu@uwa.edu.au
§ mohammed.bennamoun@uwa.edu.au

4 Word Representations

You shall know a word by the company it keeps.

— J.R. Firth, 1957

The epigraph at the beginning of this section is overused. However, it is obligatory to include it in a work such as this, as it so perfectly sums up why representations useful for language modelling are representations that capture semantics (as well as syntax).

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.

Word Vector or Word Embedding?
Some literature uses the term *word vector*, or *vector-space model* to refer to representations from LDA and LSA etc. Other works use the terms are used synonymously with *word embedding*. Word embeddings are vectors, in any case.

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 probabilis-

Dumais et al. (1988), "Using latent semantic analysis to improve access to textual information"

Blei, Ng, and Jordan (2003), "Latent dirichlet allocation"

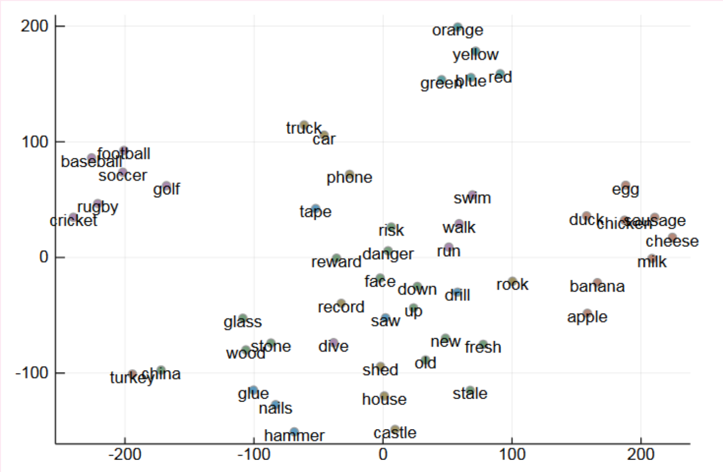


Figure 4.1: Some word embeddings from the FastText project (Bojanowski et al. 2016). They were originally 300 dimensions but have been reduced to 2 using t-SNE (Maaten and Hinton 2008) algorithm. The colors are from 5 manually annotated categories done before this visualisation was produced: `foods`, `sports`, `colors`, `tools`, `other objects`, `other`. Note that many of these words have multiple meanings (see ??), 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.

tic 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 4.1.

4.1 Representations for Language Modeling

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

$$P(W^i{=}w^i \mid W^{i-1}{=}w^{i-1}, \ldots, W^1{=}w^1) \qquad (4.1)$$

or to use the compact notation

$$P(w^i \mid w^{i-1}, \ldots, w^1) \qquad (4.2)$$

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:

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

Probability writing convention

We follow convention that capitalised W^i is a random variable, and w^i is a particular value which W^i may take. The probability of it taking that value would normally be written $P(W^i{=}w^i)$. We simply write $P(w^i)$ to mean the same thing. This is a common abridged (abuse-of) notation. The random variable in question is implicitly given by the name of its value.

Rosenfeld (2000), "Two decades of statistical language modeling: Where do we go from here?"

4.1 Representations for Language Modeling

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

Maaten and Hinton (2008), "Visualizing data using t-SNE"

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 (4.2) becomes:

$$P(w^i \mid w^{i-1}, \ldots, w^1) = P(w^i \mid w^{i-1}, w^{i-2}) \tag{4.3}$$

Google n-gram corpora Google has created a very large scale corpora of 1,2,3,4, and 5-grams from over 10^{12} words from the Google Books project. It has been made freely available at `https://books.google.com/ngrams/datasets` (Lin et al. 2012). Large scale n-gram corpora are also used outside of statistical language modeling by corpus linguists investigating the use of language.

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

Ha et al. (2009), "Extending Zipf's law to n-grams for large corpora"

55

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.

Katz (1987) and Kneser and Ney (1995), "Estimation of probabilities from sparse data for the language model component of a speech recognizer"; "Improved backing-off for m-gram language modeling"

An extended look at classical techniques in statistical language modelling can be found in Goodman (2001)

Brown et al. (1992), "Class-based n-gram models of natural language"

4.1.1 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.

Bengio et al. (2003), "A Neural Probabilistic Language Model"

4.1 Representations for Language Modeling

4.1.1.1 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:

$$P(w^i \mid w^{i-1}, w^{i-2}) = \mathrm{smax}\left(V\varphi\left(U\left[C_{:,w^{i-1}}; C_{:,w^{i-2}}\right] + \tilde{b}\right) + \tilde{k}\right) \quad (4.4)$$

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 4.2

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,

Lookup word embeddings: Hashmap or Array?
The question is purely one of implementation. For purposes of the theory, it does not matter if the implementation is using a String to Vector dictionary (e.g. a hashmap), or a 2D array from which a column is indexed-out (sliced-from) via an integer index representing the word. In the tokenization of the source text, it is common to transform all the words into integers, so as to save memory, especially if string interning is not in use. At that point it makes sense to work with an array. For our notational purposes in this book, we will treat the word w^i as if it were an integer index, though thinking of it as a string index into a hashmap changes little in the logic.

$C_{:,w^i}$ not $C_{:,i}$
Note that here we use the word w^i as the index to lookup the word embeddings. i is the index of the word index in the corpus. That is to say that if the ith word, and the jth word are the same: i.e $w^i = w^j$, then they will index out the same vector from C. $w^i = w^j \implies C_{:,w^i} = C_{:,w^j}$.

One-hot product or Indexed-lookup
In some works you may see the process of retrieving the word vector from a matrix of word vectors described as a one-hot multiplication. For a word represented

by the index w, where $\tilde{e}^w$ the one-hot vector with a 1 in the wth position, and for C, the table of word embeddings, one can write $C\,\tilde{e}^w$ to find the embedding for w. We will write $C_{:,w}$ and refer to this as looking up the word vector from the wth column. Of-course $C_{:,w} = C\,\tilde{e}^w$, however in practical implementation the performance ramifications are huge. Matrix column indexing is effectively an $O(1)$ operation (in a column major languages), whereas a dense matrix-vector product is $O(n^2)$. The one-hot product can be used in a pinch to support using embeddings in neural network toolkits that do not support lookup/embedding layers. However, we strongly suggest that if your toolkit does not support lookup/embedding layers then it is unsuitable for use in NLP applications. Some tool-kits, e.g. Flux.jl (https://github.com/FluxML/Flux.jl), explicitly handle sparse one-hot types, and automatically make this transformation. In that case, it is outright equivalent.

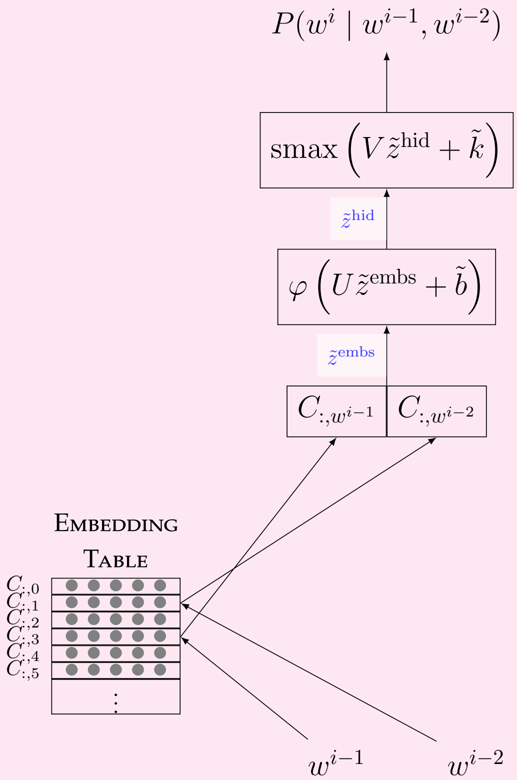


Figure 4.2: The Neural Trigram Language Model

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.

4.1.1.2 Simplified Model considered with input and output embeddings

We can actually reinterpret the softmax output layer as also having embeddings. An alternative but equivalent diagram is shown in Figure 4.3.

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

$$\mathrm{smax}(V\,\tilde{z}^{\mathrm{hid}} + \tilde{k})_{w^i} = \frac{\exp\left(V_{w^i,:}\tilde{z}^{\mathrm{hid}} + \tilde{k}_{w^i}\right)}{\sum_{\forall j} \exp\left(V_{j,:}\tilde{z}^{\mathrm{hid}} + \tilde{k}_j\right)} \tag{4.5}$$

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.

Consider, that the matrix product of a row vector with a column vector is the dot product $V_{w_i,:}\tilde{z}^{\mathrm{hid}}$ can be seen as computing the dot product between the output embedding for w_i and the hidden layer representation of the prior words/context (w^{i-1} and w^{i-2} in this case) in the form of $\tilde{z}^{\mathrm{hid}}$. This leads to an alternate interpretation of the whole process as minimising the dot-product distance between the output embedding and the context representation

This is particularly relevant for the skip-gram model discussed in Section 4.2.2 (with just one input word and no hidden layer).

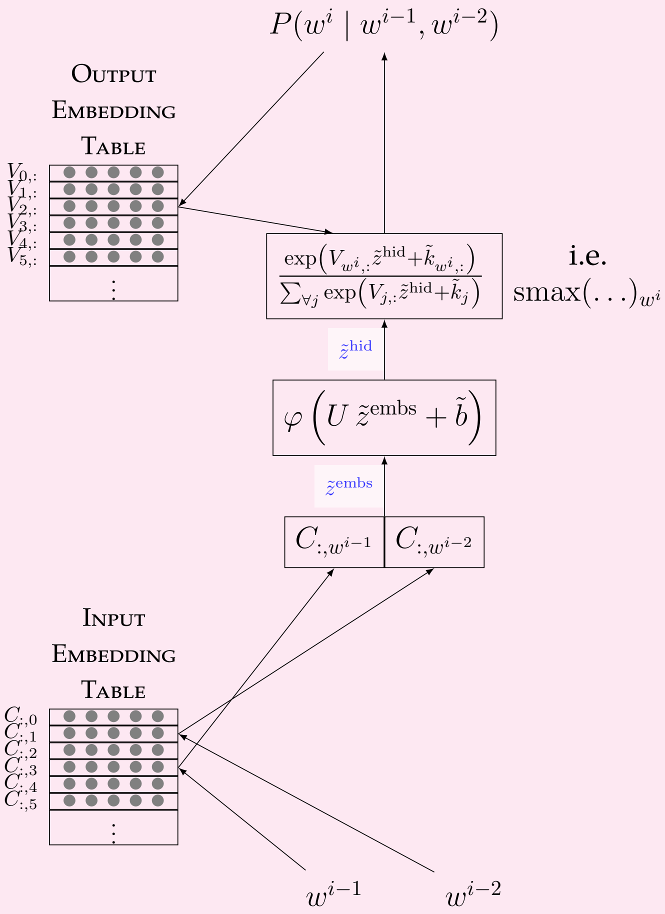


Figure 4.3: Neural Trigram Language Model as considered with output embeddings. This is mathematically identical to Figure 4.2

4.1.1.3 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 Section 2.5.1.1:

$$P(Y{=}i \mid Z{=}\tilde{z}) = \frac{R(Z{=}\tilde{z} \mid Y{=}i)\, R(Y{=}i)}{\sum_{\forall j} R(Z{=}\tilde{z} \mid Y{=}j)\, R(Y{=}j)} \tag{4.6}$$

which in this case is:

$$P(w^i \mid w^{i-1}, w^{i-2}) = \frac{R(Z{=}\tilde{z}^{\mathrm{hid}} \mid W^i{=}w^i)\, R(W^i{=}w^i)}{\sum_{\forall v \in \mathbb{V}} R(Z = \tilde{z}^{\mathrm{hid}} \mid W^i{=}v)\, R(W^i{=}v)} \tag{4.7}$$

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:

$$\frac{R(W^i{=}w^i)}{\sum_{\forall v \in \mathbb{V}} R(W^i{=}v)} = \frac{\exp\left(\tilde{k}_{w^i}\right)}{\sum_{\forall v \in \mathbb{V}} \exp\left(\tilde{k}_v\right)} \tag{4.8}$$

$$= \frac{1}{\sum_{\forall v \in \mathbb{V}} \exp\left(\tilde{k}_v - \tilde{k}_{w^i}\right)} \tag{4.9}$$

The term $R(W^i{=}w^i) = \exp\left(\tilde{k}_{w^i}\right)$ is linked to the unigram word probabilities: $P(Y = y)$. If $\mathbb{E}(R(Z \mid W_i) = 1$ then the optimal value for $\tilde{k}$ would be given by the log unigram probabilities: $k_{w^i} = \log P(W^i{=}w^i)$. This condition is equivalent to if $\mathbb{E}(V\tilde{z}^{\mathrm{hid}}) = 0$. 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.

[1]no pun intended

4.1.1.4 The Neural Probabilistic Language Model

Bengio et al. (2003), "A 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 4.4, it features a layer-bypass connection. For n prior words, the model is described by:

Input vocabulary and output vocabulary do not have to be the same Schwenk (2004) suggests using only a subset of the vocabulary as options for the output, while allowing the full vocabulary in the input space – with a fall-back to classical language models for the missed words. This decreases the size of the softmax output layer, which substantially decreases the time taken to train or evaluate the network. As a speed-up technique this is now eclipsed by hierarchical softmax and negative sampling discussed in Section 4.4. The notion of a different input and output vocabulary though remains important for word-sense embeddings as will be discussed in **??**.

$$\begin{aligned}P(w^i \mid w^{i-1}, \dots, w^{i-n}) = \mathrm{smax}(&\\ &+ V\,\varphi\left(U^{\mathrm{hid}}\left[C_{:,w^{i-1}}; \dots; C_{:,w^{i-n}}\right] + \tilde{b}\right)\\ &+ U^{\mathrm{bypass}}\left[C_{:,w^{i-1}}; \dots; C_{:,w^{i-n}}\right]\\ &+ \tilde{k})_{w^i}\end{aligned} \tag{4.10}$$

Schwenk (2004), "Efficient training of large neural networks for language modeling"

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.

4.1 Representations for Language Modeling

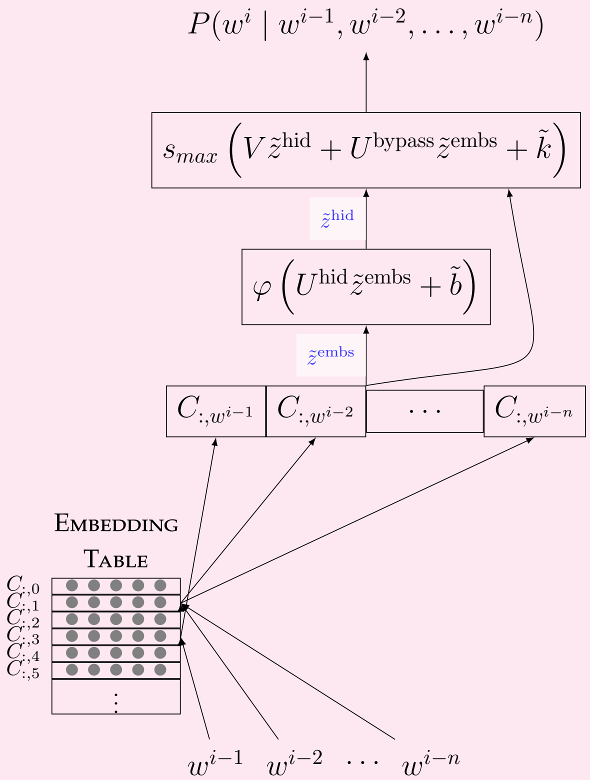


Figure 4.4: Neural Probabilistic Language Model

63

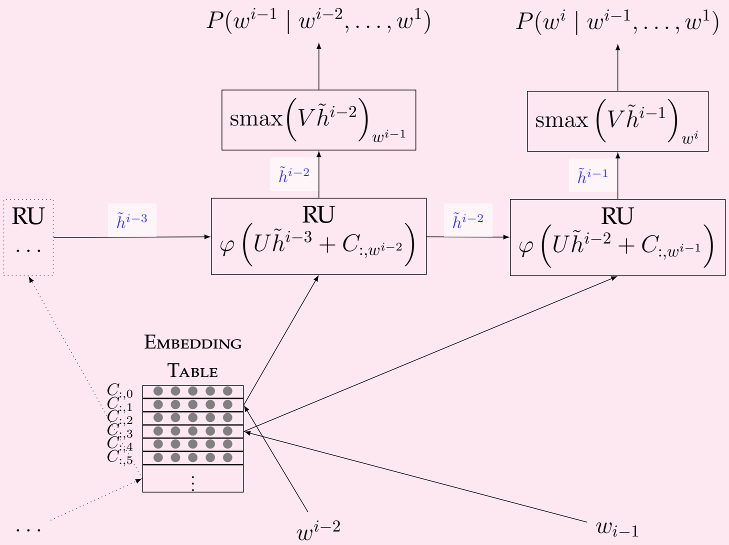


Figure 4.5: RNN Language Model. The RU equation shown is the basic RU used in Mikolov, Karafiát, et al. (2010). It can be substituted for a LSTM RU or an GRU as was done in Sundermeyer, Schlüter, and Ney (2012) and Jozefowicz, Zaremba, and Sutskever (2015), with appropriate changes.

4.1.2 RNN Language Models

Mikolov, Karafiát, et al. (2010), "Recurrent neural network based language model."

In Mikolov, Karafiát, et al. (2010) an RNN is used for language modelling, as shown in Figure 4.5. Using the terminology of Chapter 3, 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.

No Bias?
It should be noticed that Equations (4.11) and (4.12) are missing the bias terms. This is not commented on in Mikolov, Karafiát, et al. (2010). But in the corresponding chapter of Mikolov's thesis (Tomas 2012), it is explicitly noted that biases were not used in the network as it was not found that they gave a significant improvement to the result. This is perhaps surprising, particularly in the output softmax layer given the very unbalanced class (unigram) frequencies.
In the papers for several of Mikolov's other works, in-

$$\tilde{h}^i = \varphi\left(U\tilde{h}^{i-1} + C_{:,w_{i-1}}\right) \tag{4.11}$$

$$P(w^i \mid w^{i-1}, \ldots w^1) = \mathrm{smax}\left(V\tilde{h}^{i-1}\right)_{w^i} \tag{4.12}$$

Rather than using a basic RU, a more advanced RNN such as a LSTM or GRU-based network can be used. This was done by Sundermeyer, Schlüter, and Ney

(2012) and Jozefowicz, Zaremba, and Sutskever (2015), both of whom found that the more advanced networks gave significantly better results.

cluding those for skip-gram and CBOW discussed in Section 4.2, the bias terms are also excluded. We have matched those equations here. We do note though, that it is likely that many publicly available implementations of these algorithms would include the bias term: due either to a less close reading of the papers, or to the assumption that the equations are given in *design matrix* form: where the bias is not treated as a separate term to the weights, and the input is padded with an extra 1. We do not think this is at all problematic.
We discuss this further for the case of hierarchical softmax in Section 4.4.1, where the level is a proxy for the unigram frequency – and thus for the bias.

4.2 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 4.4. This section will focus just on the language model logic; and assume the output is a normal softmax layer.

Sundermeyer, Schlüter, and Ney (2012), "LSTM neural networks for language modeling"

Jozefowicz, Zaremba, and Sutskever (2015), "An empirical exploration of recurrent network architectures"

Are CBOW & Skip-Gram Neural Networks?
It is sometimes asserted that these models are not infact neural networks at all. This assertion is often based on their lack of a traditional hidden-layer, and similarities in form to several other mathematical models (discussed in Section 4.3). This distinction is purely academic though. Any toolkit that can handle the prior

4.2.1 Continuous Bag of Words

The continuous bag of words (CBOW) method was introduced by Mikolov, Chen, et al. (2013). 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).

discussed neural network models can be used to implement CBOW and Skip-Gram, more simply than using a non-neural network focused optimiser.
It also should be noted that embedding lookup is functionally an unusual hidden layer – this becomes obvious when considering the lookup as an one-hot product. Though it does lack a non-linear activation function.

Mikolov, Chen, et al. (2013), "Efficient estimation of word representations in vector space"

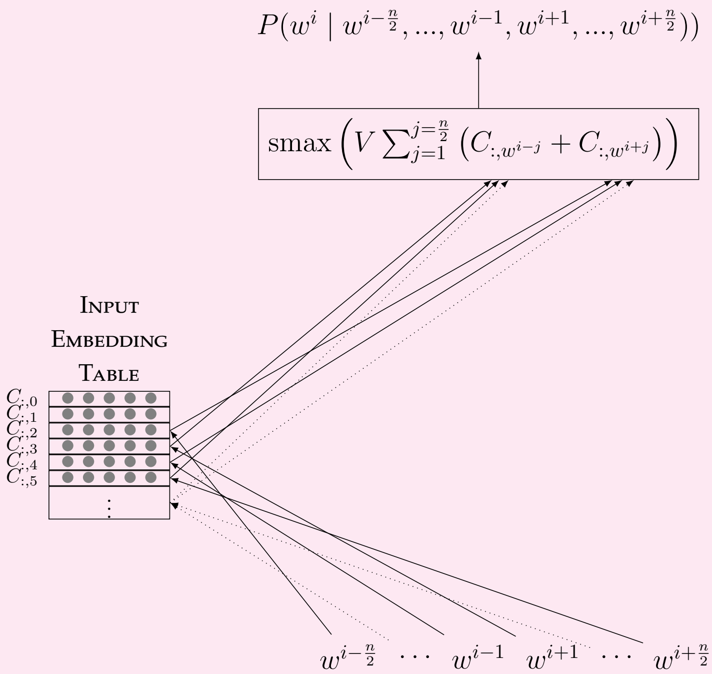


Figure 4.6: CBOW Language Model

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:

$$P(w^i \mid w^{i-\frac{n}{2}}, \ldots, w^{i-1}, w^{i+1}, \ldots, w^{i+\frac{n}{2}}) = \mathrm{smax}\left(V \sum_{j=i+1}^{j=\frac{n}{2}} (C_{:,w^{i-j}} + C_{:,w^{i+j}})\right)_{w^i} \tag{4.13}$$

This is shown in diagrammatic form in Figure 4.6. By optimising across a training dataset, useful word embeddings are found, just like in the normal language model approaches.

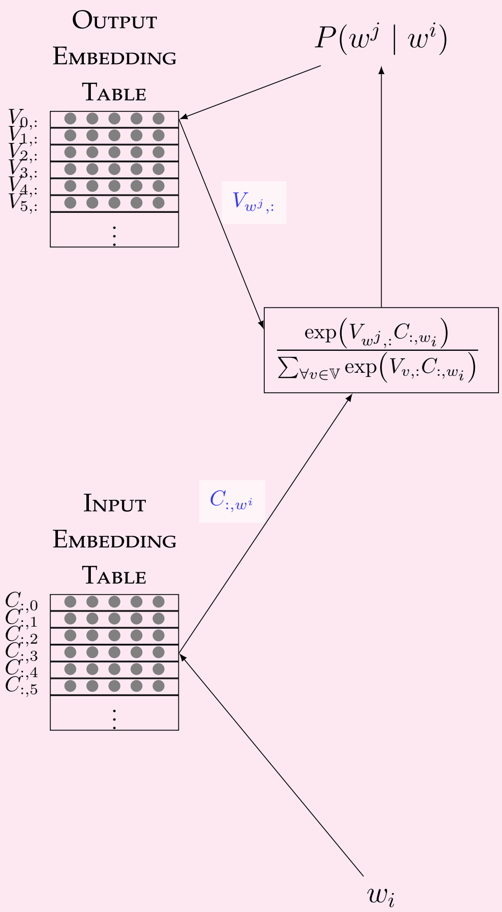


Figure 4.7: Skip-gram language Language Model. Note that the probability $P(w^j \mid w^i)$ is optimised during training for every w^j in a window around the central word w^i. Note that the final layer in this diagram is just a softmax layer, written in in output embedding form.

4.2.2 Skip-gram

The converse of CBOW is the skip-grams model Mikolov, Chen, et al. (2013). 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:

$$P(w^j \mid w^i) = \operatorname{smax}\left(V\, C_{:,w^i}\right)_{w^j} \qquad (4.14)$$

Skip-gram naming
In different publications this model may be called skipgram, skip-gram, skip-ngram, skip gram etc. Further, it may be called `word2vec` after the publicly released implementation of the algorithm. Though the `word2vec` software can also be used for CBOW, so sometimes it can refer to CBOW.

Mikolov, Chen, et al. (2013), "Efficient estimation of word representations in vector space"

4 Word Representations

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 (4.14) in output embedding form:

$$P(w^j \mid w^i) = \operatorname{smax}\left(V\, C_{:,w^i}\right)_{w^j} \tag{4.15}$$

$$P(w^j \mid w^i) = \frac{\exp\left(V_{w^j,:} C_{:,w^i}\right)}{\sum_{\forall v \in \mathbb{V}} \exp\left(V_{v,:} C_{:,v}\right)} \tag{4.16}$$

The key term here is the product $V_{w^j,:}\, C_{:,w^i}$. The remainder of Equation (4.16) 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.

68

4.2.3 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

$$\mathrm{sim}(\tilde{u}, \tilde{v}) = \frac{\tilde{u} \cdot \tilde{v}}{\|\tilde{u}\| \, \|\tilde{v}\|} \tag{4.17}$$

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 $\mathbb{V}$:

$$\operatorname*{argmax}_{\forall d \in \mathbb{V}} \mathrm{sim}(C_{:,d} - C_{:,c}, C_{:,a} - C_{:,b}) \tag{4.18}$$

$$\text{i.e} \operatorname*{argmax}_{\forall d \in \mathbb{V}} \mathrm{sim}(C_{:,d},\ C_{:,a} - C_{:,b} + C_{:,c}) \tag{4.19}$$

Analogy Tasks uncover prejudice in corpora Bolukbasi et al. (2016) and Caliskan, Bryson, and Narayanan (2017) use analogy tasks, and related variant formulations to find troubling associations between words, such as Bolukbasi et. al's titular `Man` is to `Computer Programmer`, as `Woman` is to `Homemaker`. Finding these relationships in the embedding space, indicated that they are present in the training corpus, which in turn shows their prevalence in society at large. It has been observed that machine learning can be a very good mirror upon society.

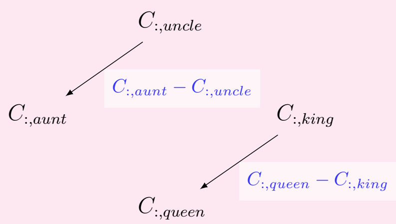


Figure 4.8: Example of analogy algebra

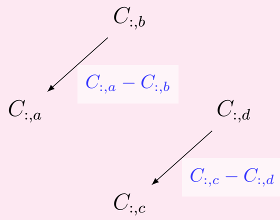


Figure 4.9: Vectors involved in analogy ranking tasks, this may help to understand the math in Equation (4.19)

Gladkova, Drozd, and Matsuoka (2016), "Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't."

This is shown diagrammaticality in Figures 4.8 and 4.9. 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, Chen, et al. (2013) 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.

Pennington, Socher, and Manning (2014), "GloVe: Global Vectors for Word Representation"

4.3 Co-location Factorisation

4.3.1 GloVe

Distance weighted co-occurrence and dynamic window sizing When training skip-gram and CBOW, Mikolov et al. used dynamic window sizing. This meant that if the specified window size was n, in any given training case being considered the actual window size was determined as a random number between 0 and n. Pennington et al. achieve a similar effect by weighting co-occurrences within a window with inverse proportion to the distance between the word. That is to say if w^i and w^j occur in the same window (i.e. $|i-j|<n$), then rather than contributing 1 to the entry in the co-occurrence count X_{w^i,w^j}, they contribute $\frac{1}{|i-j|}$.

Subsampling, and weight saturation

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

$$Loss = -\sum_{\forall w^i \in \mathbb{V}} \sum_{\forall w^j \in \mathbb{V}} X_{w^i,w^j} P(w^j \mid w^i) \log(V_{w^j,:} C_{:,w^i}) \tag{4.20}$$

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

4.3 Co-location Factorisation

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.

$$\begin{aligned} \text{optimise} \quad s(w^i, w^j) &= V_{w^j,:} C_{:,w^i} + \tilde{b}_{w^i} + \tilde{k}_{w^j} && (4.21) \\ \text{such that} \quad s(w^i, w^j) &\approx \log(X_{w^i,w^j}) && (4.22) \end{aligned}$$

This is done via the minimisation of

$$Loss = -\sum_{\forall w^i} \sum_{\forall w^j} f(X_{w^i,w^j}) \left(s(w^i, w^j) - \log(X_{w^i,w^j}) \right) \tag{4.23}$$

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

$$f(x) = \begin{cases} \left(\frac{x}{100}\right)^{0.75} & x < 100 \\ 1 & \text{otherwise} \end{cases} \tag{4.24}$$

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.

Skip-gram and CBOW models use a method called subsampling to decrease the effect of common words. The subsampling method is to randomly discard words from training windows based on their unigram frequency. This is closely related to the saturation of the co-occurrence weights as calculated by $f(X)$ used by GloVe. Averaged over all training cases the effect is nearly the same.

Key Factors Mentioned as Asides

There is an interesting pattern of factors being considered as not part of the core algorithm. We have continued this in the side-notes of this section; with the preceding notes on Distance weighting and subsampling. While the original papers consider these as unimportant to the main thrust of the algorithms Levy, Goldberg, and Dagan (2015) found them to be crucial hyper-parameters.

Pennington, Socher, and Manning (2014), "GloVe: Global Vectors for Word Representation"

Levy, Goldberg, and Dagan (2015), "Improving Distributional Similarity with Lessons

Learned from Word Embeddings"

Implementing GloVe
To implement GloVe in any technical programming language with good support for optimisation is quiet easy, as it is formed into a pure optimization problem. It is also easy to do in a neural network framework, as these always include an optimiser. Though unlike in normal neural network training there are no discrete training cases, just the global co-occurrence statistics.

Levy and Goldberg (2014), "Neural word embedding as implicit matrix factorization"

Li et al. (2015), "Word Embedding Revisited: A New Representation Learning and Explicit Matrix Factorization Perspective."

Cotterell et al. (2017), "Explaining and Generalizing Skip-Gram through Exponential Family Principal Component Analysis"

Landgraf and Bellay (2017), "word2vec Skip-Gram with Negative Sampling is a Weighted Logistic PCA"

4.3.2 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.

4.3.3 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.

4.4 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.

4.4.1 Hierarchical Softmax

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

Morin and Bengio (2005), "Hierarchical probabilistic neural network language model"

Mikolov, Chen, et al. (2013), "Efficient estimation of word representations in vector space"

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

SemHuff
It can be noted that the Huffman encoding scheme specifies only the depth of a given word in the tree. It does not specify the order. Yang et al. (2016) make use of the BlossomV algorithm (Kolmogorov 2009) to pair the nodes on each layer according to their similarity. They found that on the language modelling task this improved performance, in the way one would expect. They used a lexical resource to determine similarity, however noted that a prior trained word-embedding model could

The tree is normally a Huffman tree (Huffman 1952), as was found to be effective by Mikolov, Chen, et al. (2013). 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 be used to define similarity instead – the new encoding can then be used to define a new model which will find new (hopefully better) embeddings. This is similar to the original method used by (Morin and Bengio 2005), but only using the similarity measure for reordering nodes at the same depth, after the depth is decided by Huffman encoding. In our own experimentation, when applying it to other tasks, we did not see large improvements. It is nevertheless a very interesting idea, and quite fun to implement and observe the results.

Huffman (1952), "A method for the construction of minimum-redundancy codes"

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, Chen, et al. (2013), 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.

4.4.1.1 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, Chen, et al. (2013)). 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 $2^{-k} > P(w^i)$.

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

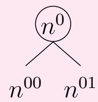


Figure 4.10: Tree for 2 words

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

Here we write n^{01} to represent the event of the first non-root node being the branch given by following left branch, while n^{01} being to follow the right branch. (The order within the same level is arbitrary in anycase, 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.

$$P(n^{01} \mid \tilde{z}) = 1 - P(n^{00} \mid \tilde{z}) \tag{4.25}$$

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 $V_{0,:}$.

$V_{0,:}\tilde{z}$ is a dot product
We mentioned in the marginalia earlier, but just as an extra reminder: the matrix product of a row vector like $V_{0,:}$ with a (column) vector like $\tilde{z}$ is their vector dot product.

$$P(n^{00} \mid \tilde{z}) = \sigma(V_{0,:}\tilde{z}) \tag{4.26}$$
$$P(n^{01} \mid \tilde{z}) = 1 - \sigma(V_{0,:}\tilde{z}) \tag{4.27}$$

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

$$P(n^{01} \mid \tilde{z}) = \sigma(-V_{0,:}\tilde{z}) \tag{4.28}$$

thus

$$P(n^{0i} \mid \tilde{z}) = \sigma((-1)^i V_{0,:}\tilde{z}) \tag{4.29}$$

Noting that in Equation (4.29), i is either 0 (with $-1^0 = 1$) or 1 (with $-1^1 = -1$)).

4.4.1.1.2 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.

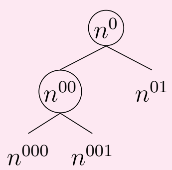


Figure 4.11: Tree for 3 words

From earlier we still have:

$$P(n^{00} \mid \tilde{z}) = \sigma(V_{0,:}\tilde{z}) \tag{4.30}$$
$$P(n^{01} \mid \tilde{z}) = \sigma(-V_{0,:}\tilde{z}) \tag{4.31}$$

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:

$$P(n^{000} \mid \tilde{z}) = P(n^{000} \mid \tilde{z},\, n^{00})\, P(n^{00} \mid \tilde{z}) \tag{4.32}$$
$$P(n^{000} \mid \tilde{z}, n^{00}) = \sigma(V_{00,:}\tilde{z}) \tag{4.33}$$
$$P(n^{001} \mid \tilde{z}, n^{00}) = \sigma(-V_{00,:}\tilde{z}) \tag{4.34}$$

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

$$P(n^{01} \mid \tilde{z}) = \sigma(-V_{0,:}\tilde{z}) \tag{4.35}$$
$$P(n^{000} \mid \tilde{z}) = \sigma(V_{00,:}\tilde{z})\sigma(V_{0,:}\tilde{z}) \tag{4.36}$$
$$P(n^{001} \mid \tilde{z}) = \sigma(-V_{00,:}\tilde{z})\sigma(V_{0,:}\tilde{z}) \tag{4.37}$$

4.4.1.1.3 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:

$$P(n^{0t^1 \ldots t^L} \mid \tilde{z}) = P(n^{0t^1 \ldots t^L} \mid \tilde{z},\, n^{0t^1 \ldots t^{L-1}})\, P(n^{0t^1 \ldots t^{L-1}} \mid \tilde{z}) \tag{4.38}$$

Thus:

$$P(n^{0t^1} \mid \tilde{z}) = \sigma\left((-1)^{t^1} V_{0,:}\tilde{z}\right) \tag{4.39}$$
$$P(n^{0t^1,t^2} \mid \tilde{z},\, n^{0t^1}) = \sigma\left((-1)^{t^2} V_{0t^1,:}\tilde{z}\right) \tag{4.40}$$
$$P(n^{0t^1 \ldots t^i} \mid \tilde{z},\, n^{0t^1 \ldots t^{i-1}}) = \sigma\left((-1)^{t^i} V_{0t^1 \ldots t^{i-1},:}\tilde{z}\right) \tag{4.41}$$

The conditional probability chain rule, is applied to get:

$$P(n^{0t^1 \ldots t^L} \mid \tilde{z}) = \prod_{i=1}^{i=L} \sigma\left((-1)^{t^i} V_{0t^1 \ldots t^{i-1},:}\tilde{z}\right) \tag{4.42}$$

4 Word Representations

4.4.1.2 Formulation

Combining multiplications
If one wants to find both $V_{00,:}\tilde{z}$ and $V_{0,:}\tilde{z}$, then this can be done using matrices simultaneously, thus potentially taking advantage of optimized matrix multiplication routines.
$\begin{bmatrix} V_{0,:} \\ V_{00,:} \end{bmatrix} z = \begin{bmatrix} V_{0,:}\tilde{z} \\ V_{00,:}\tilde{z} \end{bmatrix}$
Thus the whole product for all of the decisions can be written as $V\tilde{z}$. The problem then becomes indexing the relevant node rows.
However computing every single decision is beyond what is required for most uses: hierarchical softmax lets us only compute the decisions that are on the path to the word-leaf we which we wish to query. Computing all of them is beyond what is required.
Packing tree node elements into a matrix with fast indexing is a non-trivial problem. The details on optimising such multiplications and tree packing are beyond the scope of this book.
In general there may be very little scope here for optimisation, as on most hardware (and BLAS systems) matrix products with n columns, takes a similar amount of time to n vector dot products. As such storing the row vectors of V in a hashmap indexed by node-path, and looping over them as required may be more practical.
In languages/libraries with slow looping constructs (numpy, R, octave), where calling into suitable library routines is much faster, this may give some speed-up;

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, Sutskever, et al. (2013).

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:

$$P(n^{0t^1\ldots t^L} \mid \tilde{z}) = \prod_{i=1}^{i=L} \sigma\left((-1)^{t^i+1} V_{0t^1\ldots t^{i-1},:}\tilde{z}\right) \tag{4.43}$$

or to allow it to vary per node: as in the formulation of Mikolov, Sutskever, et al. (2013). 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.

$$P(n^{0t^1\ldots t^L} \mid \tilde{z}) = \prod_{i=1}^{i=L} \sigma\left(\left[\!\left[n^{0t^1\ldots t^i} = ch(n^{0t^1\ldots t^{i-1}})\right]\!\right] V_{0t^1\ldots t^{i-1},:}\tilde{z}\right) \tag{4.44}$$

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.

4.4.1.3 Loss Function

Using normal softmax, during the training, the cross-entropy between the model's predictions and the ground

78

truth as given in the training set is minimised. Cross entropy is given by

$$CE(P^{\star}, P) = \sum_{\forall w^i \in \mathbb{V}} \sum_{\forall z^j \in \mathbb{Z}} -P^{\star}(w^i \mid z^j) \log P(w^i \mid z^j) \tag{4.45}$$

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 $\mathcal{X}$.

$$Loss = CE(P^{\star}, P) = \frac{1}{|\mathcal{X}|} \sum_{\forall (w^i, z^i) \in \mathcal{X}} -\log P(w^i \mid z^i) \tag{4.46}$$

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 4.1.1.

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 backpropagation as usual.

4.4.2 Negative Sampling

Negative sampling was introduced in Mikolov, Sutskever, et al. (2013) as another method to speed up this problem.

but even here it is likely to be minor. The time may be better spent writing a C extension library to do this part of the program. Or learning to use a language with fast for loops (e.g. Julia (Bezanson et al. 2014)).

Mikolov, Sutskever, et al. (2013), "Distributed representations of words and phrases and their compositionality"

How does this relate to word vectors?
After the length of this section, one may have forgotten why we are doing this in the first place. Recall that CBOW, skip-gram and all other language modelling based word embedding methods are based around predicting $P(w^o \mid w^i, \ldots, w^j)$ for some words. For skip-gram that is just $P(w^o \mid w^i)$. The term $n^{0t^1 \ldots t^L}$ in $P(n^{0t^1 \ldots t^L} \mid z)$, just represents as a path through the tree to the leaf node which represents the word w^o. i.e $P(n^{0t^1 \ldots t^L} \mid z) = P(w^o \mid z)$. The output of the final hidden layer is z (i.e. the z is the input to the output layer) In normal language models z encodes all the information about what the model knows of predictions based on $w^i \ldots, w^j$. z is thus a proxy term in the conditional probability for those words. In skip-gram there is no hidden layer, and it is just $z = C_{:,w^i}$ proxying only for w_i, and the model defines its probability output by $P(w^o \mid w^i) = P(w^o \mid C_{:,w^i})$.

The gradient calculations

Much like hierarchical softmax in its purpose. However, negative sampling does not modify the network's output, but rather the loss function.

They are not fun. They never are for back-propagation. We recommend using a framework with automated differentiation, and/or performing gradient checks against a numerical differentiation tool (simple finite-differencing will do in a pinch).

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.

Mikolov, Sutskever, et al. (2013), "Distributed representations of words and phrases and their compositionality"

Gutmann and Hyvärinen (2012), "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics"

4.4.2.1 A Motivation of Negative Sampling

Recall from Section 4.2.2 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 4.2.2 we considered the representation of $P(w^j \mid w^i)$ as the w^jth element of the softmax output.

$$P(w^j \mid w^i) = \mathrm{smax}(V\, C_{:,w^i})_{w^j} \tag{4.47}$$

$$P(w^j \mid w^i) = \frac{\exp\left(V_{w^j,:} C_{:,w^i}\right)}{\sum_{k=1}^{k=N} \exp\left(V_{k,:} C_{:,k}\right)} \tag{4.48}$$

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.

Why is not using softmax wrong?
The notation abuse may be hiding just how bad it is to not use softmax. Recall that the true meaning of $P(w^j \mid w^i)$ is actually $P(W^j{=}w^j \mid W^i{=}w^i)$. By not using softmax, with its normalising denominator this means that: $\sum_{\forall w^j \in \mathbb{V}} P(w^j \mid w^i) \neq 1$ (except by coincidence).

4.4 Hierarchical Softmax and Negative Sampling

$$P(w^j \mid w^i) = \sigma(V\,C_{:,w^i})_{w^j} \quad (4.49)$$

$$\text{i.e.} \quad P(w^j \mid w^i) = \sigma(V_{w^j,:}C_{:,w^i}) \quad (4.50)$$

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 Function
Readers may want to reread Section 4.4.1.3 to brush up on how we use the training dataset as a ground truth probability estimate implicitly when using cross-entropy loss. When doing so one should remember that that the conditioning term, z, for skip-grams is the co-located words as there is no hidden layer.

$$Loss = \sum_{\forall (w^i,w^j)\in\mathcal{X}} -\log P(w^j \mid w^i) \quad (4.51)$$

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:

$$loss(w^j, w^i) = \log P(w^k \mid w^i) - \log P(w^j \mid w^i) \quad (4.52)$$

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

Most words do not co-occur
Some simple reasoning can account for this as a reasonable consequence of Zipf's law (Zipf 1949) and a prior of

81

the principle of indifference, but there is a further depth to it as explained by Ha et al. (2009).

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.

Is Equation (4.53) a function?
No, at the point at which the Loss started including randomly selected samples, it ceased to be a function in the usual mathematical sense. It is still a function in the common computer programming sense though – it is just not deterministic.

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

$$loss(w^j, w^i) = \left(\sum_{\forall w^k \in \mathrm{samples}(D^{1g})} \log P(w^k \mid w^i) \right) - \log P(w^j \mid w^i) \tag{4.53}$$

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

Consider, though is this fair to the samples? We are taking them as representatives of all words that do not co-occur. Should a word that is unlikely to occur at all, *but was unlucky enough to be sampled*, contribute the same to the loss as a word that was very likely to occur? More reasonable is that the loss contribution should be in proportion to how likely the samples were to occur. Otherwise it will add unexpected changes and result in noisy training. Adding a weighting based on the unigram probability ($P^{1g}(w^k)$) gives:

$$loss(w^j, w^i) = \left(\sum_{\forall w^k \in \mathrm{samples}(D^{1g})} P^{1g}(w^k) \log P(w^k \mid w^i) \right) - \log P(w^j \mid w^i) \tag{4.54}$$

4.4 Hierarchical Softmax and Negative Sampling

The expected value is defined by

$$\mathop{\mathbb{E}}_{X\sim D}[f(x)] = \sum_{\forall x \text{ values for } X} P^{\mathrm{d}} f(x) \qquad (4.55)$$

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

$$\sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k\sim D^{1\mathrm{g}}}[\log P(w^k \mid w^i)] \qquad (4.56)$$

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

Mikolov, Sutskever, et al. (2013), "Distributed representations of words and phrases and their compositionality"

$$loss(w^j, w^i) = \left(\sum_{k=1}^{k=n} \mathop{\mathbb{E}}_{w^k\sim D^{1\mathrm{g}}}[\log P(w^k \mid w^i)]\right) - \log P(w^j \mid w^i) \qquad (4.57)$$

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, Sutskever, et al. (2013) uses a distribution such that

$$P^{\mathrm{D^{ns}}}(w^k) = \frac{P^{\mathrm{D^{1g}}}(w^k)^{\frac{2}{3}}}{\sum_{\forall w^o \in \mathbb{V}} P^{\mathrm{D^{1g}}}(w^o)^{\frac{2}{3}}} \qquad (4.58)$$

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

4 Word Representations

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

$$loss(w^j, w^i) = \left(\sum_{k=1}^{k=n} \underset{w^k \sim D^{\text{ns}}}{\mathbb{E}}[\log \sigma(V_{w^k,:} C_{:,w^i}) \right) - \log \sigma(V_{w^j,:} C_{:,w^i}) \quad (4.59)$$

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.

$$loss(w^j, w^i) = \left(\sum_{k=1}^{k=n} \underset{w^k \sim D^{\text{ns}}}{\mathbb{E}}[-1 + \log \sigma(V_{w^k,:} C_{:,w^i}) \right) - \log \sigma(V_{w^j,:} C_{:,w^i}) \quad (4.60)$$

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

$$loss(w^j, w^i) = - \log \sigma(V_{w^j,:} C_{:,w^i}) - \sum_{k=1}^{k=n} \underset{w^k \sim D^{\text{ns}}}{\mathbb{E}}[\log \sigma(-V_{w^k,:} C_{:,w^i})] \quad (4.61)$$

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

$$Loss = - \sum_{\forall (w^i, w^j) \in \mathcal{X}} \left(\log \sigma(V_{w^j,:} C_{:,w^i}) + \sum_{k=1}^{k=n} \underset{w^k \sim D^{\text{ns}}}{\mathbb{E}}[\log \sigma(-V_{w^k,:} C_{:,w^i})] \right) \quad (4.62)$$

This is the negative sampling loss function used in Mikolov, Sutskever, et al. (2013). Perhaps the most confusing part of this is the notation. Without the abuses

84

around expected value, this is written:

$$Loss = -\sum_{\forall (w^i,w^j)\in\mathcal{X}} \left(\log\sigma(V_{w^j,:}C_{:,w^i}) + \sum_{\forall w^k \in \mathrm{samples}(D^{\mathrm{ns}})} P^{\mathrm{D^{ns}}}(w^k)\log\sigma(-V_{w^k,:}C_{:,w^i})\right) \tag{4.63}$$

4.5 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).

Collobert and Weston (2008), "A unified architecture for natural language processing: Deep neural networks with multitask learning"

4.5.1 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.

Pretrained Word-Embeddings
Pretrained Word Embeddings are available for most models discussed here. They are trained on a lot more data than most people have reasonable access to. It can be useful to substitute word embeddings as a representation in most systems, or to use them as initial value for neural network systems that will learn them as they train the system as a whole. There are many many

4.6 Aligning Vector Spaces Across Languages

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

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

By performing CCA one solves to find a series of vectors (also expressed as a matrix), $S = \left[\tilde{s}^1 \ldots \tilde{s}^{\mathrm{d}}\right]$ and $T = \left[\tilde{t}^1 \ldots \tilde{t}^{\mathrm{d}}\right]$, 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 pro-

online pretrained word embeddings. One of the more recent and comprehensive set is that of Bojanowski et al. (2016) (based on a skip-gram extension), https://fasttext.cc/docs/en/pretrained-vectors.html They provide embeddings for 294 languages, trained on Wikipedia based on the work of which is an extension to skip-grams.

Turian, Ratinov, and Bengio (2010), "Word representations: a simple and general method for semi-supervised learning"

Fu et al. (2016), "Efficient and Distributed Algorithms for Large-Scale Generalized Canonical Correlations Analysis"

Bojanowski et al. (2016), "Enriching Word Vectors with Subword Information" jection 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.

Gujral, Khayrallah, and Koehn (2016), "Translation of Unknown Words in Low Resource Languages"

Klein et al. (2015), "Associating neural word embeddings with deep image representations using fisher vectors"

Dhillon, Foster, and Ungar (2011) investigated using this to create word-embeddings. We noted in Equation (4.16), 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.

Dhillon, Foster, and Ungar (2011), "Multi-view learning of word embeddings via cca"

Dhillon, Foster, and Ungar (2011), "Multi-view learning of word embeddings via cca"

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 6). A

Shi et al. (2015), "Learning Cross-lingual Word Embeddings via Matrix Cofactorization."

Socher et al. (2014), "Grounded compositional semantics for finding and describing images with sentences"

Ruder (2017), "A survey of cross-lingual embedding models"

survey paper on such methods was recently published by Ruder (2017).

Bibliography

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]`.

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 (2016). "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606*.

Bolukbasi, Tolga, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai (2016). "Man is to computer programmer as woman is to homemaker? Debiasing word embeddings". In: *Advances in Neural Information Processing Systems*, pp. 4349–4357.

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.

Caliskan, Aylin, Joanna J. Bryson, and Arvind Narayanan (2017). "Semantics derived automatically from language corpora contain human-like biases". In: *Science* 356.6334, pp. 183–186. ISSN: 0036-8075. DOI: `10.1126/science.aal4230`. eprint: `http://science.sciencemag.org/content/356/6334/183.full.pdf`.

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.

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.

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.

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.

Bibliography

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

Fu, X., K. Huang, E. E. Papalexakis, H. A. Song, P. P. Talukdar, N. D. Sidiropoulos, C. Faloutsos, and T. Mitchell (Dec. 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.

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.

Goodman, Joshua (2001). "A Bit of Progress in Language Modeling". In: *CoRR* cs.CL/0108005.

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.

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

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.

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.

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.

Kolmogorov, Vladimir (2009). "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Mathematical Programming Computation* 1.1, pp. 43–67.

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

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, 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.

Lin, Yuri, Jean-Baptiste Michel, Erez Lieberman Aiden, Jon Orwant, Will Brockman, and Slav Petrov (2012). "Syntactic annotations for the google books ngram corpus". In: *Proceedings of the ACL 2012 system demonstrations*. Association for Computational Linguistics, pp. 169–174.

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

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

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 (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in Neural Information Processing Systems*, pp. 3111–3119.

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

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.

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.

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.

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.

Bibliography

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.

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

Tomas, Mikolov (2012). "Statistical language models based on neural networks". PhD thesis. PhD thesis, Brno University of Technology.

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.

Yang, Zhixuan, Chong Ruan, Caihua Li, and Junfeng Hu (2016). "Optimize Hierarchical Softmax with Word Similarity Knowledge". In: *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

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.

5 Word Sense Representations

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

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.

93

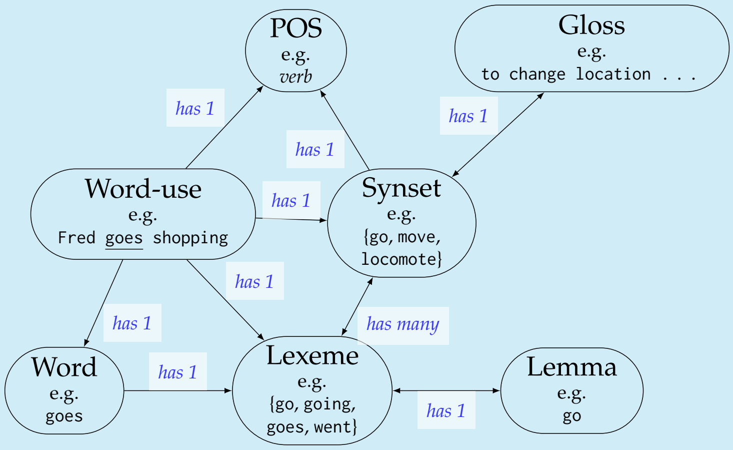


Figure 5.1: The relationship between terms used to discuss various word sense problems. The lemma is used as the representation for the lexeme, for WordNet's purposes when indexing. For many tasks each the word-use is pre-tagged with its lemma and POS tag, as these can be found with high reliability 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.

5.1 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.

Polysemous/Homonymous A word with multiple meanings i.e. senses. For NLP representational purposes polysemous and homonymous are synonymous.

Part of Speech/POS The syntactic category a word belongs to. Different POS tags come from different tag sets. Can be simple as the WordNet tag set: noun, adjective, verb, etc. or complex as in the Brown tag set: VBG-*verb,*

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 5.1

5.1.1 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.

gerund/present participle, NN- *noun, singular or mass*.

Word-use
An occurrence of a word in a text, such as a training corpus. Each word will have multiple uses in a text. Each word-use will only have one particular meaning and will thus belong to one synset.

Lemma
The base form of the word as defined by a lexicographical resource. It is normally closely related to (often identical to) the *stem* which is the word's root form with all morphological inflections (e.g. tenses) removed.

Lexeme
The set of words that share a common lemma: `go`, `going`, `goes`, and `went` all belong to the lexeme headed by the lemma `go`

Synset
A synset is a set of synonymous words: that is words that have the same meaning. In lexographic terms the synset is the core unit of meaning. Identifying the synset of a word-use is the the same as identifying the word sense. Every word sense corresponds to one synset.

Gloss
A gloss is the dictionary entry for a word sense, it normally includes both the definition and an example of use. In WordNet each synset shares a common gloss.

Lemmatization

Lemmatization is the method of converting a word into its lemma. Due to the similary of the lemma to the stem, this in essence means removing the tense and plurality information (stemming), with some additional special-cases. WordNet is indexed by lemmas, and comes with a lemmatizer called morphy allowing any word to be looked up by lemmatizing it to it's lemma

Unlemmatization
Given lemma (as one can extract from WordNet) and a full POS tag (such as a Brown-style tag) for a word, it is possible to undo the lemmatization with a high degree of reliability using relatively simple rules (again due to the similarity of the lemma to the stem). The POS tag encodes the key inflectional features that are lost. Patten.en (De Smedt and Daelemans 2012) is a python library encoding such rules (pluralisation, verb conjugation, etc.); though combining them with the POS tag to drive them is a task left for the reader. This can be used to find substitute words using WordNet's features, for finding synonyms, antonyms and other lemmas from lexically related categories.

Miller (1995), "WordNet: a lexical database for English"

Navigli and Ponzetto (2010), "BabelNet: Building a very large multilingual semantic network"

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 (G. K. 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 (G. 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.

5.1.1.1 Most Frequent Sense

Given a sense annotated corpus, it is easy to count how often each sense of a word occurs. Due to the overwhelming 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: $\operatorname{argmax}_{\forall s^j} P(s^j \mid w^i, p_i)$.

Deng et al. (2009), "ImageNet: A Large-Scale Hierarchical Image Database"

G. K. Zipf (1945), "The meaning-frequency relationship of words"

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

Kilgarriff (2004), "How Dominant Is the Commonest Sense of a Word?"

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.

5.2 Word Sense Representation

It is desirable to create a vector representation of a word sense much like in Chapter 4 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 5.3 considers the case when the senses are to also be discovered, as well as represented.

Semantic Syllepsis
(Also known as pathological sentences that kill almost all WSD systems.) Consider the sentence: `John used to work for the newspaper that you are carrying.`. In this sentence the word-use `newspaper` simultaneously have two different meanings: it is both the company, and the object. This violates our earlier statement that every word-use belongs to exactly one synset. WSD systems are unable to handle these sentences as they attempt to assign a single sense to each word-use. Most word sense induction systems cannot do much better: at best a new sense could be allocated for the joint use, which does not correspond to the linguistic notion of the word having two senses for different parts of the sentence. Most works on word sense disambiguation outright ignore these sentences, or consider them to be ungrammatical, or incorrect. However, they are readily

5.2.1 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 4. Iacobacci, Pilehvar, and Navigli (2015) use a CBOW language model (Mikolov et al. 2013) 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.

5.2.2 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 5.2.

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.

understood and used without thought by most native speakers. These constructions are also known as *zeugma*, although zeugma is itself a highly polysemous word, so its usage varies.

De Smedt and Daelemans (2012), "Pattern for python"

Moro and Navigli (2015), "SemEval-2015 Task 13: Multilingual All-Words Sense Disambiguation and Entity Linking"

WordNet is not a strong moral baseline
WordNet, as a resource,-based partly on the work of Princeton undergraduate students in the early 1990's, and on the literature of 1961, is not the kind of resource one might hope for from an AI information perspective. The glosses include a number of biases. These biases are reflective of the language use, but are not necessarily ideal to be encoded into a system. For example: `S: (v) nag, peck, hen-peck (bother persistently with trivial complaints) 'She nags her husband all day long's.` Other dictionaries regularly show up in the News for similar content. Another problem is the source of the word sense counts. As discussed in the main text, sense counts are important in WSD systems. The counts come from SemCor, a sense annotated subset of the Brown Corpus. The Brown Corpus is a sampling of American texts from 1961. The cultural norms of 1961 were not

5.2 Word Sense Representation

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. 2013). 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

the norms of today. (For context, note that the US did not pass the Civil Rights act to end segregation until 1964). As such, one should not trust WordNet (or SemCor) to reflect current sense counts, for words which have undergone usage change since 1961. Furthermore, when creating down stream resources-based on WordNet, one should not use these sense counts to determine how important it is to include a concept. If ImageNet (Deng et al. 2009) for example, had used SemCor counts to determine which synsets of images would be included, then items rarely discussed in 1961 literature, like `wheelchairs`, and `prosthesis` would be excluded. Which would in turn make many image processing systems systematically unhelpful in processing images relating to the disabled. (Do not fear: even the initial release of ImageNet contains hundreds of images of `wheelchairs`, and `prosthesis`) Unintentional biasing of data can have on-going effects on the behaviour of machine learning-based systems far beyond the original conception.

Iacobacci, Pilehvar, and Navigli (2015), "SensEmbed: learning sense embeddings for word and relational similarity"

Chen, Liu, and Sun (2014), "A Unified Model for Word 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.

5.3 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

Sense Representation and Disambiguation."

WSD with embeddings
It is beyond the scope of this work to fully discuss WSD systems. However, we will remark that (single sense) word embeddings are a generally useful feature as an input to any NLP ML system. As such they can be used as features in a fully supervised WSD system. The idea of using them in this way is is similar to the LSI enhanced Lesk WSD system of Basile, Caputo, and Semeraro (2014).

Cosine distance
Here we talk of cosine distance, where a smaller distance implies more similar (and 0.0 identical). Contrasting this with the cosine similarity, where higher value implies more similar (and 1.0 identical).
Cosine distance is still not a true metric as $d^{\cos}(v, kv) = 0$ for all $k \in \mathbb{R}_+$). Other times you may see cosine similarity, ranging between -1 (most different) and 1 (most similar. Cosine similarity is given by $sim(a, b) = \frac{\tilde{a}\cdot\tilde{b}}{\|\tilde{a}\|_2 \|\tilde{b}\|_2} = \cos(\angle\tilde{a}\tilde{b})$ i.e. the unit-length normalised dot product of the vectors. Cosine distance is usually defined as $d^{\cos}(\tilde{a}, \tilde{b}) = \frac{1 - sim(\tilde{a},\tilde{b})}{2}$. Ranging between 0 (most similar) and 1 (most different).

Can we go from induced senses to lexical senses
A natural question given the existence of many WSI systems, and the existing

wealth of lexically indexed resources, is if we can align induced senses to a set of lexically defined senses. Agirre, Martínez, et al. (2006) proposed a method for doing this using a weighted mapping-based on the probabilities found using induced sense WSD on a labelled "mapping" corpus. This has only been used on relatively small datasets with only hundreds of words (SenseEval 3 (Mihalcea, Chklovski, and Kilgarriff 2004) and SemEval-2007 Task 02 (Agirre and Soroa 2007)). Our own investigations in White et al. (2018) with the larger SemEval 2007 Task 7 (Navigli, Litkowski, and Hargraves 2007) suggest that it may not scale very well to to real-word WSD tasks. That work proposed an alternative method that worked better, though still not as well as could be hoped. Finding suitable methods to link unsupervised representations, to human defined senses remains a topic worthy of research.

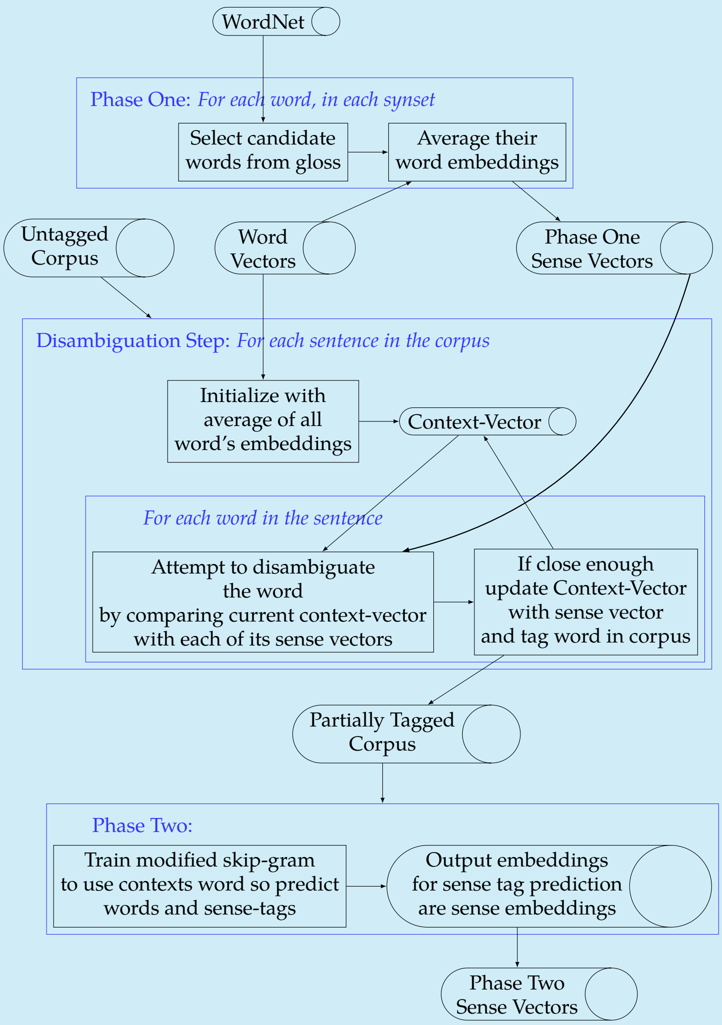


Figure 5.2: The process used by Chen, Liu, and Sun (2014) to create word sense embeddings.

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.

Why do skip-grams perform so well on SCWS? SCWS is a corpus designed for evaluating word sense embeddings. Single sense embeddings (e.g. skip-grams) cannot take advantage of the context information in the SCWS. However, they do often perform comparably to the word sense embeddings. Sometimes even outperforming them. It is unclear if this highlights the difficulty of the task (i.e. that the impact of context is hard to gauge), or it might be due to the (implicit) most frequent sense dominating both the use in the tasks, and the representation in a single sense. Alternatively, it may just be the result of the fine tuning of the more mature single-sense embedding methods (and that with more time and tuning multiple sense methods could do proportionally better).

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 4. 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.

5.3.1 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.

5.3.1.1 Offline clustering

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

Schütze (1998), "Automatic Word Sense Discrimination"

Pantel and Lin (2002), "Discovering word senses from text"

Reisinger and Mooney (2010), "Multi-prototype vector-space models of word meaning"

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.

On context representations These co-location clustering methods require finding a representation for the context (from this a similarity metric is applied, and the clustering is then done). More generally, this can be related to the next chapter: Chapter 6, as any of these methods could be used to derive a vector representation of a context. In most works (including all the works discussed here) comparatively simple representations of the contexts are used. It would be interesting to extend the sentence representation methods, and apply them to this use.

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,

Huang et al. (2012), "Improving word representations via global context and multiple word prototypes"

Kågebäck et al. (2015), "Neural context embeddings for automatic discovery of word senses"

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.

On clustering
Clustering can be defined as a (mixed integer) optimisation task, of assigning points to clusters so as to satisfy some loss function-based around minimising intra-cluster variance while maximising inter-cluster variance (or a similar measure). As this is NP-hard, most clustering methods are approximate. K-means is very popular because of its simplicity, however it easily falls into local minima, and so normally it is run dozens of times (at least) to obtain more optimal results. K-means also has the issue of having to select the number of clusters (k). It should be remembered that there exist many other clustering methods than k-means (and its variants). These other methods use different loss functions, and different strategies to overcome the NP-hard nature of the problem. In particular their mixture model methods, hierarchical methods, spectral methods, and others. We personally favour affinity propagation (Frey and Dueck 2007), though there is provably no ideal clustering algorithm even in the non-heuristic case (Kleinberg 2003). On

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 5.2.2. 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.

any clustering task (word sense or otherwise) it is worth investigating several clustering algorithms, and not just settling for k-means (particularly not settling for k-means run once.). A series of interesting and easy reading articles on clustering can be found at: http://alexhwilliams.info/itsneuronalblog/2015/09/11/clustering1/, http://alexhwilliams.info/itsneuronalblog/2015/10/01/clustering2/, and http://alexhwilliams.info/itsneuronalblog/2015/11/18/clustering-is-easy/

5.3.1.2 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.

Neelakantan et al. (2015), "Efficient non-parametric estimation of multiple embeddings per word in vector space"

Kågebäck et al. (2015) investigated using their weighting function (as discussed in Section 5.3.1.1) with the online clustering used by Neelakantan et al. (2015).

5 Word Sense Representations

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.

5.3.2 Co-location Prediction-based Approaches

Probability
One may wish to brush up on basic probability notions for this section. In particular joint, conditional and marginal probabilities definitions; as well as Bayes Theorem and the probability chain-rule which come from those. In brief these are as follows.

Conditional Probability:
$P(A \mid B) = \frac{P(A,B)}{P(B)}$

Marginal Probability:
$P(A) = \sum_{\forall b} P(A, B=b)$

Bayes Theorem:
$P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}$

Probability Chain-rule:
$P(A^n, \ldots, A^1)$
$= P(A^n \mid A^{n-1}, \ldots, A^1) P(A^{n-1}, \ldots, A^1)$
e.g.
$P(A, B, C)$
$= P(A \mid B, C) P(B \mid C) P(C)$

The latter three rules are consequences of the first.

Tian et al. (2014), "A Probabilistic Model for Learning Multi-Prototype Word Embeddings."

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:

$$P(w^o \mid w^i) = \sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k, w^i) P(s^k \mid w^i) \quad (5.1)$$

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 (5.1) as:

$$P(w^o \mid w^i) = \sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k)\, P(s^k \mid w^i) \quad (5.2)$$

A softmax classifier can be used to define $P(w^o \mid s^k)$, just like in normal language modelling. With output

106

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 (5.2) 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:

$$\hat{P}(s^l \mid w^o) = \frac{\hat{P}(s^l \mid w^i)P(w^o \mid s^l)}{\sum_{\forall s^k \in \mathcal{S}(w^i)} \hat{P}(w^o \mid s^k)P(s^k \mid w^i)} \tag{5.3}$$

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

$$\hat{P}(s^l \mid w^i) = \frac{1}{|\mathcal{X}|} \sum_{\forall (w^o, w^i) \in \mathcal{X}} \hat{P}(s^l \mid w^o) \tag{5.4}$$

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

5 Word Sense Representations

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 (G. K. 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), "Breaking Sticks and Ambiguities with Adaptive Skip-gram"

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.

WordNet and BabelNet
As mentioned in the previous sections, WordNet and BabelNet are the predominant lexicons used for word senses. It is not directly relevant to this section, but we have space here to remark upon them. WordNet (Tengi 1998) as a very well established tool has a have a binding in practically every modern programming language suitable for NLP. WordNet.jl (`https://github.com/JuliaText/WordNet.jl`) is the Julia binding. NLTK (Bird, Klein, and Loper 2009) includes one for Python. BabelNet (Navigli and Ponzetto 2010) is intended to be accessed as an online resource, via a RESTful API. Users receive 1000 free queries per day. Academic users can request an upgrade to 50,000 queries per day, or to download a copy of the database. From personal experience we found those requests to be handled easily and rapidly.

Considering again the form of Equation (5.2)

$$P(w^o \mid w^i) = \sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k) P(s^k \mid w^i) \tag{5.5}$$

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.

$$P(s^l \mid w^o) = \frac{P(w^o \mid s^l)P(s^l \mid w^i)}{\sum_{\forall s^k \in \mathcal{S}(w^i)} P(w^o \mid s^k)P(s^k \mid w^i)} \tag{5.6}$$

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:

$$P(s^l \mid \mathcal{W}^i) = \frac{\prod_{\forall w^j \in \mathcal{W}^i} P(w^j \mid s^l)P(s^l|w^i)}{\sum_{s^k \in \mathcal{S}(w^i)} \prod_{\forall w^j \in \mathcal{W}^i} P(w^j \mid s^k)P(s^k|w^i)} \tag{5.7}$$

Independence Assumption Technically, Equation (5.6) does not require the independence of the probabilities of the context words. Rather it only requires that the context words be conditionally independent on the word in question w^i. Nevertheless, even the conditional independence assumption is incorrect, except for a theoretical perfect embedding capturing perfect information. The conditional independence assumption remains useful as an approximation.

5.4 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.

Finding the nearest neighbours (Nearest Neighbour Trees) A common evaluation task with any representation is to find its nearest neighbours. The naïve solution is to check the distance to all points. For n points this is $O(n)$ operations. For word embeddings n is the size of the vocabulary, perhaps $100,000$ words. Performing $100,000$ operations per check, is not entirely unreasonable on modern computers (even when the operations are on 300 dimensional representations). However, for word sense embeddings, which have many senses per word in the vocabulary, this means many more points to check. 30 senses per word is not unusual for fine-grained word sense induction. Having a total $n = 3,000,000$ representations to check causes a noticeable delay. To solve this we can use data structures designed for fast nearest neighbour querying. A k-d tree takes at worst $O(n \log_2(n))$ time to construct. Once constructed on average it takes $O(\log(n))$ to find the nearest neighbour to any point. This makes checking the nearest neighbour nearly instantaneous for even the largest vocabularies.

Bibliography

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.

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

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

Basile, Pierpaolo, Annalina Caputo, and Giovanni Semeraro (Aug. 2014). "An Enhanced Lesk Word Sense Disambiguation Algorithm through a Distributional Semantic Model". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland: Dublin City University and Association for Computational Linguistics, pp. 1591–1600.

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural language processing with Python*. " O'Reilly Media, Inc.".

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

De Smedt, Tom and Walter Daelemans (2012). "Pattern for python". In: *The Journal of Machine Learning Research* 13.1, pp. 2063–2067.

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*.

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.

Frey, Brendan J and Delbert Dueck (2007). "Clustering by passing messages between data points". In: *Science* 315.5814, pp. 972–976.

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.

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.

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.

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, 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.

Kleinberg, Jon M (2003). "An impossibility theorem for clustering". In: *Advances in neural information processing systems*, pp. 463–470.

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

Mihalcea, Rada, Timothy Anatolievich Chklovski, and Adam Kilgarriff (2004). "The Senseval-3 English lexical sample task". In: Association for Computational Linguistics.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "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.

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

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

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

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.

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.

Schütze, Hinrich (Mar. 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.

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

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

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

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

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

6 Sentence Representations and Beyond

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

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.

Word Embeddings as a by-product Many sentence representation methods produce word embeddings as a by-product. These word embeddings are either output embeddings, from the softmax, or input embeddings from a lookup layer.

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.

Initialising input embeddings It is common (but not ubiquitous) to initialise the input embeddings using pretrained embeddings from one of the methods discussed in Chapter 4, then allow them to be fine-tuned while training the sentence representation method.

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).

Dumais et al. (1988), "Using latent semantic analysis to

6 Sentence Representations and Beyond

improve access to textual information"

Hofmann (2000), "Learning the similarity of documents: An information-geometric approach to document retrieval and categorization"

Blei, Ng, and Jordan (2003), "Latent dirichlet allocation"

Word Sense Embeddings in Sentence Embeddings
While **??** was all about sense embeddings, they are unmentioned here. One might think that they would be very useful for sentence embeddings. However, they are not as needful as one might expect. The sense of a word being used is determined by the context. Ideally, it is determined by what the context means. As a sentence embedding is a direct attempt to represent the meaning of such a context, determining the sense of each word within it is not required. Using sense embeddings instead of word embeddings is a valid extension to many of these methods. However it requires performing word sense disambiguation, which as discussed is very difficult.

Mitchell and Lapata (2008), "Vector-based Models of Semantic Composition."

SOWE is the product of the BOW with an embedding matrix
The reader may recall from Chapter 4, that a word-embedding lookup is the same as a one-hot vector product: $C_{:,w^i} = C\,\hat{e}_{w^i}$. Similar can be said for sum of

6.1 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.

6.1.1 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 (Lyndon White et al. 2016a).

Surprisingly, these unordered methods have been found on many tasks to be extremely well performing, bet-

116

6.1 Unordered and Weakly Ordered Representations

ter than several of the more advanced techniques discussed later in this chapter. This has been noted in several works including: Lyndon White et al. (2015), Ritter et al. (2015) and R. 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 (Lyndon 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.

word embeddings (SOWE) and bag of words (BOW). For some set of words $\mathcal{W} = \{w_1, \ldots, w_n\}$: the BOW representation is $B_{\mathcal{W}} = \sum_{w^i \in \mathcal{W}} \hat{e}_{w^i}$; the SOWE representation is $\sum_{w^i \in \mathcal{W}} C_{w^i} = CB_{\mathcal{W}}$. As with word-embeddings, it is immensely cheaper to calculate this via lookup and sum, rather than via matrix product; except on systems with suitable sparse matrix product tricks.

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

Lyndon White et al. (2015), "How Well Sentence Embeddings Capture Meaning"

Ritter et al. (2015), "Leveraging Preposition Ambiguity to Assess Compositional Distributional Models of Semantics"

R. Wang, Liu, and McDonald (2017), "A Matrix-Vector Recurrent Unit Model for Capturing Compositional Semantics in Phrase Embeddings"

Horvat and Byrne (2014), "A Graph-Based Approach to String Regeneration."

Lyndon White et al. (2016b), "Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem"

Li et al. (2017), "Neural Bag-of-Ngrams."

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 4, 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.

6.1.2 Paragraph Vector Models (Defunct)

Window vs Context
It is important to be clear in this section on the difference between the window and the context. The window is the words near the target word. The context (in this context) refers to the larger structure (sentence, paragraph, document) that a representation is attempting to be found for. The window is always a subset of the context. In modelling the context many windows within it will be considered (one per target word). Some works say sentence vector, document vector or paragraph vector. We say context vector as it could be any of the above. In theory it could even be a whole collection of documents.

Le and Mikolov (2014), "Distributed Representations of Sentences and Documents"

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 Model Implementations
There is a popular third-party implementation of both the paragraph vector models, under the name `doc2vec` in the python gensim library (Řehůřek and Sojka 2010),

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.

along with many information retrieval vector models such as LDA.

$$P(w^i \mid d^k,\, w^{i-\frac{n}{2}}, ..., w^{i-1}, w^{i+1}, ..., w^{i+\frac{n}{2}}) = \mathrm{smax}(W D_{:,d^k} + U \sum_{j=i+1}^{j=\frac{n}{2}} \left(C_{:,w^{i-j}} + C_{:,w^{i+j}}\right)) \tag{6.1}$$

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.

$$P(w^j \mid d^k,\, w^i) = \left[\mathrm{smax}(W D_{:,d^k} + V C_{:,w^i})\right]_{w_j} \tag{6.2}$$

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 (S. Wang and Christopher D 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 (S. Wang and Christopher D Manning 2012). On a different short text task, Lyndon 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.

Mesnil et al. (2014), "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews"

S. Wang and Christopher D Manning (2012), "Baselines and bigrams: Simple, good sentiment and topic classification"

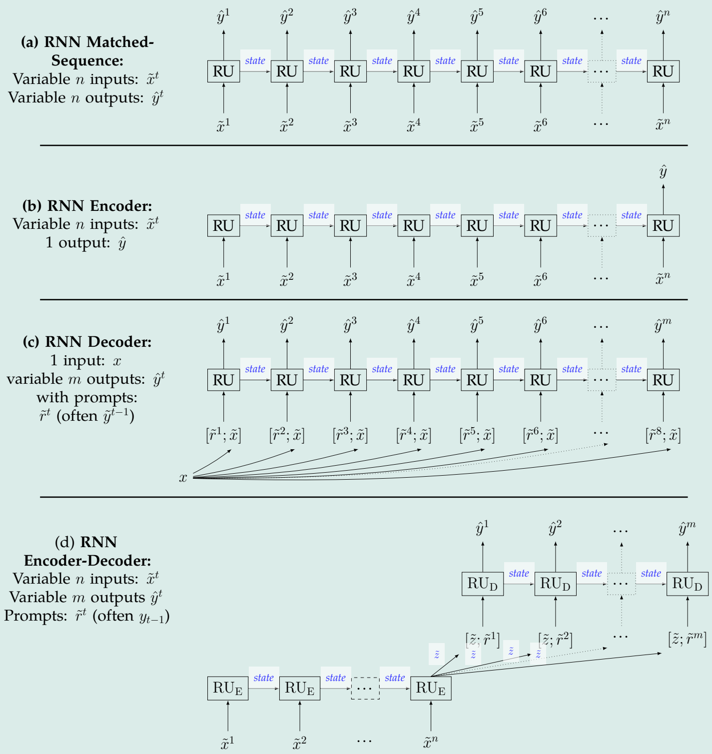


Figure 6.1: The unrolled structure of an RNN for use in (a) Matched-sequence (b) Encoding, (c) Decoding and (d) Encoding-Decoding (sequence-to-sequence) problems. RU is the recurrent unit – the neural network which reoccurs at each time step. (Repeated from Figure 3.1)

6.2 Sequential Models

The majority of this section draws on the recurrent neural networks (RNN) as discussed in Chapter 3. 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 6.1) 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

Cho et al. (2014), "Learning Phrase Representations using RNN Encoder–Decoder

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

RNN, to create embeddings for the translated phrases. Several other RNNs have been used in this way since.

for Statistical Machine Translation"

6.2.1 VAE and encoder-decoder

Bowman, Vilnis, et al. (2016) 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 6.2. The variational autoencoder (Kingma and Welling 2014) has been demonstrated to have very good properties in a number of machine learning applications: they are able to work to find continuous latent variable distributions over arbitrary likelihood functions (such as in the neural network); and are very fast to train. Using the VAE, it is hoped that a better representation can be found for the sequence of words in the input and output.

Bowman, Vilnis, et al. (2016), "Generating Sentences from a Continuous Space"

Kingma and Welling (2014), "Auto-Encoding Variational Bayes"

Bowman, Vilnis, et al. (2016) 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, Vilnis, et al. (2016) investigated such a configuration, and found that it did not yield an improvement in performance.

6.2.2 Skip-thought

Kiros et al. (2015) draws inspiration from the works on acausal language modelling, to attempt to predict

Kiros et al. (2015), "Skip-Thought Vectors"

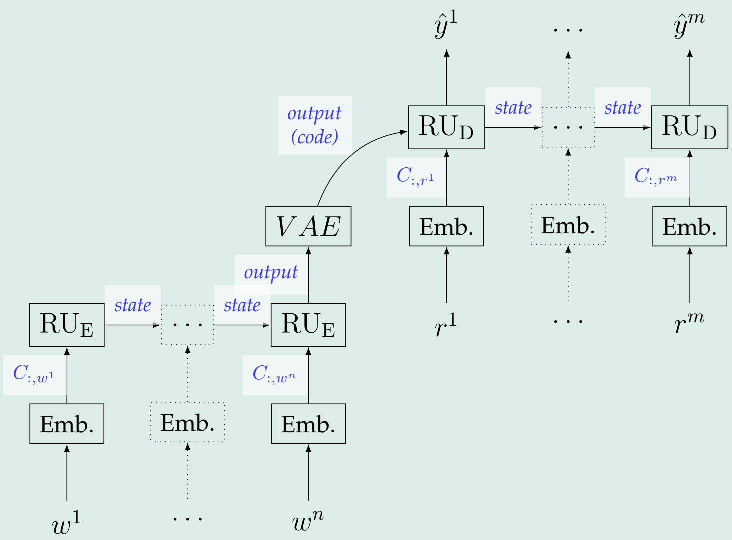


Figure 6.2: The VAE plus encoder-decoder of Bowman, Vilnis, et al. (2016). Note that during training, $\hat{y}^i = w^i$, as it is an autoencoder model. As is normal for encoder-decoders the prompts are the previous output (target during training, predicted during testing): $r^i = y^{i-1}$, with $r^1 = y^0 =$ `<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 4 we showed this as as a table but just as a block here for conciseness.

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 6.3 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 Section 3.2.3, 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.

6.3 Structured Models

Parsers
There are many high-quality parsing libraries available. The most well known is the Stanford CoreNLP library (Christopher D. Manning et al. 2014) for java. It has an interactive web-demo at `http://corenlp.run/`, which was used to produce Figures 6.4 and 6.5.

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, NLTK (Bird, Klein, and Loper 2009) contains several different parsers, including a binding to CoreNLP parsers. The newer spaCy library (Honnibal and Johnson 2015) for python, presently only features a dependency parser.

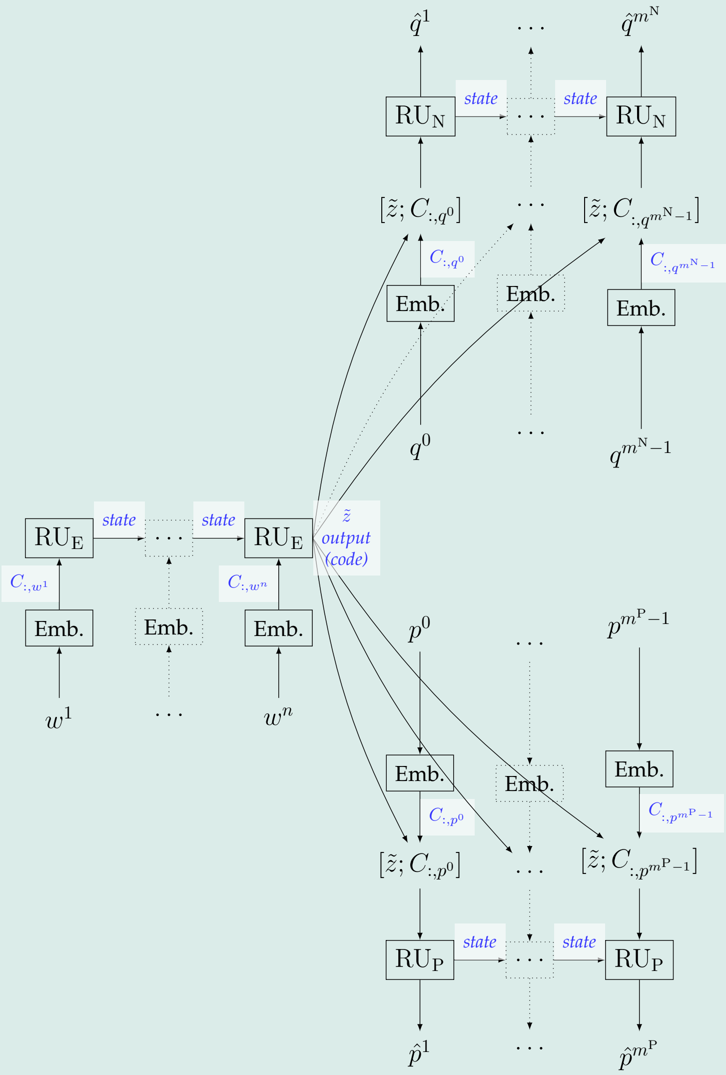


Figure 6.3: The skip-thought model (Kiros et al. 2015). Note that for the next and previous sentences respectively the outputs are $\hat{q}^i$ and $\hat{p}^i$, and the prompts are q^{i-1} and p^{i-1}. As there is no intent to use the decoders after training, there is no need to worry about providing 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.

and dependency parse trees. Examples of each are shown in Figures 6.4 and 6.5. 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

Socher (2014), "Recursive Deep Learning for Natural Language Processing and Computer Vision"

Goller and Kuchler (1996), "Learning task-dependent distributed representations by backpropagation through structure"

Pollack (1990), "Recursive distributed representations"

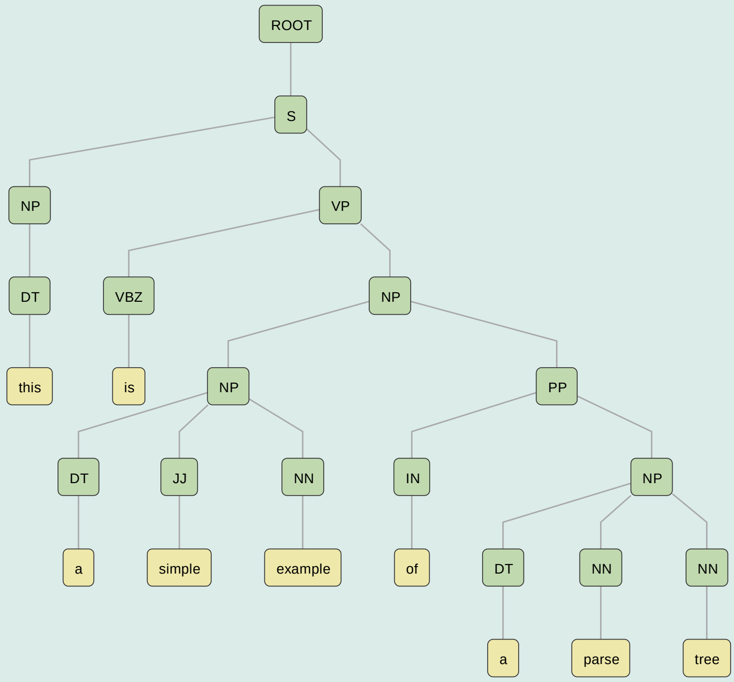


Figure 6.4: A constituency parse tree for the sentence: `This is a simple example of a parse tree.` In this diagram the leaf nodes are the input words, their intimidate parents are their POS tags, and the other nodes with multiple children represent sub-phrases of the sentence, for example NP is a Noun Phrase.

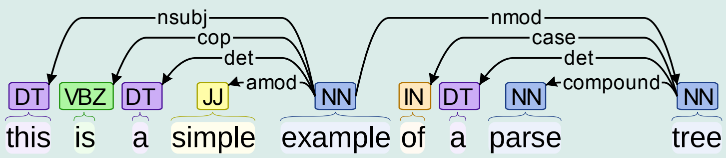


Figure 6.5: A dependency parse tree for the sentence `This is a simple example of a parse tree`, This flattened view may be misleading. `example` is at the peak of the tree, with direct children being: `this`,`is`,`a`,`simple`, and `tree`. `tree` has direct children being: `of`,`a`, and `parse`.

that structuring a network according to its parse tree is possible.

6.3.1 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

Machine learning frameworks for structural models
Structural networks cannot be easily defined in most static neural network libraries, such as TensorFlow. These implementations function by defining a single computational graph that is used to process each training/test case. The same graph is used for each input. By definition, the structure of the network differs from training/test

case to training/test case. Technically the same problems apply to RNNs, as each case can have a different number of inputs. This is normally worked around by defining the network graph for the longest input to be considered, then padding all the inputs to this length, and ensuring that the padding does not interfere with the gradient updates. The equivalent tricks for structured networks are significantly more complex. The exception to this is of-course via dynamic components to the static frameworks (which TensorFlow and other such frameworks certainly do have). Even in a dynamic framework it remains a non-trivial task to implement these networks.

Implementing Back-propagation through structure
Conceptually, back-propagation through structure is not significantly more complex than back-propagation through time. However, in practice it is a very difficult algorithm to get right. It is very important to test for correctness using gradient checks, as it is easy to make a mistake and end-up with models that seem to work ok, but are actually crippled due to some mistake in the coding. Unfolding recursive autoencoders are particularly difficult, as the gradient must be propagated from all leaves. And output interior nodes cannot have their gradients calculated

graph) can be defined by the composition function:

$$f^{\mathrm{RV}}(\tilde{u},\tilde{v}) = \varphi\left(\left[\begin{array}{cc} S & R \end{array}\right]\left[\begin{array}{c}\tilde{u}\\ \tilde{v}\end{array}\right] + \tilde{b}\right) \tag{6.3}$$

$$= \varphi\left(S\tilde{u} + R\tilde{v} + \tilde{b}\right) \tag{6.4}$$

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

This is a useful form as all constituency parse trees can be converted into binary parse trees, via left-factoring or right factoring (adding new nodes to the left or right to take some of the children). This is sometimes called binarization, or putting them into Chomsky normal form. This form of structured network has been used in many words, including Socher, Christopher D Manning, and Ng (2010), Socher, Pennington, et al. (2011), Socher, Huang, et al. (2011), Socher, Lin, et al. (2011) 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.

6.3.2 Dependency Tree

The dependency tree is the other commonly considered parse-tree. Structured networks based upon the dependency tree have been used by Socher, Karpathy, et al. (2014), Iyyer, Boyd-Graber, Claudino, 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 6.5, 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:

$$f^{\mathrm{RV}}(i) = \varphi\left(W^{\mathrm{head}} C_{:,w^i} + \sum_{j \in \mathrm{children}(i)} W^{\pi(\mathrm{i,j})}\, f_{RV}(j) + \tilde{b}\right) \tag{6.5}$$

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 6.4, 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.

until the gradients of their children are calculated. The solution to this is to process the node gradient calculation using a priority queue, where the priority is set by the depth of the node. Thus ensuring that all children are processed before their parents.

Socher, Christopher D Manning, and Ng (2010), "Learning continuous phrase representations and syntactic parsing with recursive neural networks"

Socher, Karpathy, et al. (2014), "Grounded compositional semantics for finding and describing images with sentences"

Iyyer, Boyd-Graber, Claudino, et al. (2014), "A neural network for factoid question answering over paragraphs"

Iyyer, Boyd-Graber, and Daumé III (2014), "Generating Sentences from Semantic Vector Space Representations"

Extended example

The full example for the $f^{\mathrm{RV}}(9)$ from Equation (6.5) is:

$$\begin{aligned} f^{\mathrm{RV}}(9) = \varphi(& W^{\mathrm{head}} C_{:,\texttt{tree}} \\ &+ W^{\mathrm{compound}}(W^{\mathrm{head}} C_{:,\texttt{parse}} + \tilde{b}) \\ &+ W^{\mathrm{det}}(W^{\mathrm{head}} C_{:,\texttt{a}} + \tilde{b}) \\ &+ W^{\mathrm{case}}(W^{\mathrm{head}} C_{:,\texttt{of}} + \tilde{b}) \\ &+ \tilde{b}) \end{aligned}$$

This in turn would be composed as part of $f^{\mathrm{RV}}(5)$ for the whole tree headed by $w^5 = \texttt{example}$. The output of each RV is a representation of that substructure.

No gates
No long-term memory
We note that a limitation of most structural models, compared to the sequential RNNs, is their lack of explicit gating on memory (e.g. as in GRU and LSTM). Any given path down a tree can be looked at as a simple RNN comprised only of basic recurrent units. However, these paths are much shorter (being the logarithm of) than the full sequential length of the sentence, which offsets the need for such gating. Recall that the gating is to provide the longer short term memory.

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 6.4.

6.3.3 Parsing

Stenetorp (2013), "Transition-based Dependency Parsing Using Recursive Neural Networks"

The initial work for both contingency tree structured networks (Socher, Christopher D 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.

The finer detail of parsing
Parsing is one of the most well studied problems in computational linguistics. Presented here is only the highest level overview. For more details on this, we recommend consulting the source materials. Ideally, with reference to a good traditional (that is to say non-neural network based) NLP textbook, such as: C. Manning and Schütze (1999).

When creating a binary contingency parse tree, any pair of nodes can only be merged if they are adjacent. The process described by Socher, Christopher D 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 4.4.2). 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, Christopher D 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.

Getting the Embeddings out of the Parser
The implementation of Socher, Christopher D Manning, and Ng (2010), is publicly available. However, it does not export embeddings. It is nested deep inside the Stanford Parser, and thus accessing the embeddings is not at all trivial.

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 (6.4), 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 6.3.2). This is required, as the relationships must be determined from the links made, and thus are not available before the parse. Bowman, Gauthier, et al. (2016), 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.

Bowman, Gauthier, et al. (2016), "A fast unified model for parsing and sentence understanding"

6.3.4 Recursive Autoencoders

Application to image retrieval
An interesting application of structured networks was shown in Socher, Karpathy, et al. (2014). A dependency tree structured network was

Recursive autoencoders are autoencoders, just as the autoencoder discussed in Section 2.5.2, they reproduce their input. It should be noted that unlike the encoder-decoder RNN discussed in Section 6.2.1, 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 6.3.3).

The models presented in Socher, Huang, et al. (2011) 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, Pennington, et al. (2011) 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.

trained on a language modelling task (not as a recursive autoencoder, although that would also have been a valid option). Then, separately a convolutional neural network was trained to produce a vector output of the same dimensionality – an image embedding – such that its distance to its caption's composed vector representation was minimised. The effect of this was that images and their captions are projected to a common vector space. This allowed for smart image retrieval, from descriptions, by having a set of all images, and storing their embedding representations. Then for any query, the sentence embedding can be found and the vector space of images can be searched for the nearest. The reverse is not generally as useful, as one can't reasonably store all possible captions describing an image, so as to be able to search for the best one for a user provided image. This process of linking a sequence representation to an image embedding is not restricted to structured networks, and can be done with any of the representation methods discussed in this chapter. Further, as discussed in Section 4.6 it can also be done using pretrained embedding on both sides through (kernel) CCA.

Unfolding RAE implementation
The implementation, and a pretrained model, of the unfolding recursive autoencoder of Socher, Huang, et

6.3.4.1 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.

al. (2011) is available online at https://tinyurl.com/URAE2011. It is easy to use as a command-line Matlab script to generate embeddings.

Socher, Christopher D Manning, and Ng (2010), "Learning continuous phrase representations and syntactic parsing with recursive neural networks"

- In Socher, Pennington, et al. (2011) this was for sentiment analysis.
- In Socher, Huang, et al. (2011) 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.

Sequential models are often preferred to structural models Sequential (RNN) models are much more heavily researched than structural models. They have better software libraries, are easier to implement, and have more known "tricks" (like gates and attention). In theory it is possible for a sequential model (with sufficiently deep and wide RUs) to internalise the connections that a structural model would possess. While structural models are theoretically nicer from a linguistics standpoint, pragmatically they are the last resort in modelling. When at-

6.4 Matrix Vector Models

6.4.1 Structured Matrix Vector Model

Socher, Huval, 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.

tempting to find a useful representation of a sentence for a task, one should first try a sum of word embeddings with a simple network on-top, then a sequential model (based on LSTM or GRU), and only if these fail then try a structured model. Arguably before using any neural network approach at all, one should eliminate bag of words, bag of n-grams, the dimensionality reduced version of those bags, and also eliminate LSI and LDA as inputs for the task.

Socher, Huval, et al. (2012), "Semantic compositionality through recursive matrix-vector spaces"

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

$$f^{\mathrm{RV}}(\tilde{u}, \tilde{v}, U, V) = \varphi\left([S\ R][U\tilde{v}; V\tilde{u}] + \tilde{b}\right) \tag{6.6}$$

$$= \varphi\left(S\,U\tilde{v} + R\,V\tilde{u} + \tilde{b}\right) \tag{6.7}$$

$$F^{\mathrm{RV}}(U, V) = W\,[U; V] = W^{\mathrm{l}}U + W^{\mathrm{r}}V \tag{6.8}$$

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

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

6.4.2 Sequential Matrix Vector Model

R. Wang, Liu, and McDonald (2017), "A Matrix-Vector Recurrent Unit Model for Capturing Compositional Semantics in Phrase Embeddings"

A similar approach, of capturing a per word matrix, was used on a sequential model by R. 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 3, 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 6.6 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.

Remember:
The product of a matrix with a concatenated vector, is the same as the sum of the two blocks of that matrix each multiplied by the blocks of that vector.

$$h^t = \tanh\left(W^{\mathrm{h}}[x^t; \hat{y}^{t-1}] + \tilde{b}^{\mathrm{h}}\right) \tag{6.9}$$

$$z^t = \sigma\left(Y^{t-1}x^t + X^t\hat{y}^{t-1} + \tilde{b}^{\mathrm{z}}\right) \tag{6.10}$$

$$\hat{y}^t = z^t \odot h^t + (1 - z^t) \odot \hat{y}^{t-1} \tag{6.11}$$

$$Y^t = \tanh\left(W^{\mathrm{Y}}[Y^{t-1}; X^t] + \tilde{b}^{\mathrm{Y}}\right) \tag{6.12}$$

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, Huval, et al. (2012) is applied in the gate z^t. The method for calculating h^t is similar to that of a normal RU.

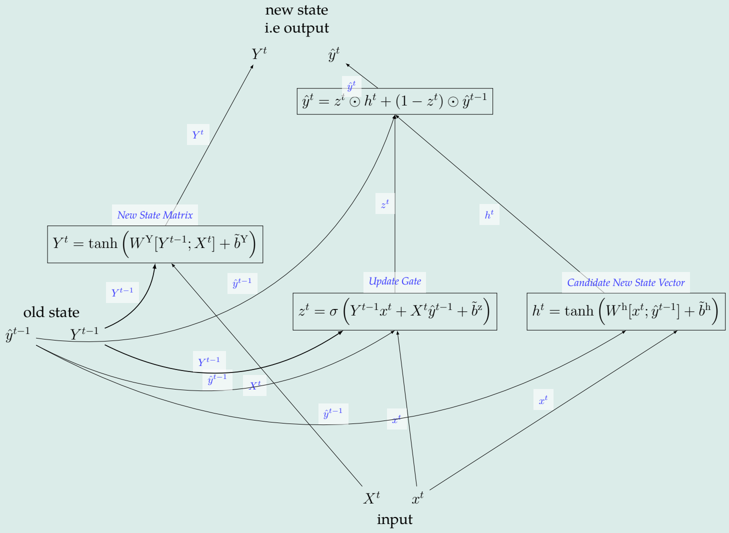


Figure 6.6: A Matrix Vector recurrent unit

The work of R. 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

6.5 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 (L. White et al. 2017). More generally as almost all aspects of language have some degree of compositionality, and sequential models work very well on most language tasks, this implicitly shows that they have sufficient representational capacity to learn sufficient degrees of compositional processing to accomplish these tasks.

L. White et al. (2017), "Learning Distributions of Meant Color"

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 (Lyndon 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.

Bibliography

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural language processing with Python*. " O'Reilly Media, Inc.".

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.

Bowman, Samuel R, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts (2016). "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 (2016). "Generating Sentences from a Continuous Space". In: *International Conference on Learning Representations (ICLR) Workshop*.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (Oct. 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.

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.

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.

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.

Honnibal, Matthew and Mark Johnson (Sept. 2015). "An Improved Non-monotonic Transition System for Dependency Parsing". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1373–1378.

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

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.

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

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.

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.

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

Manning, C.D. and H. Schütze (1999). *Foundations of Statistical Natural Language Processing*. MIT Press. ISBN: 9780262133609.

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky (2014). "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60.

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*.

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

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 (May 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.

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*.

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

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

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, 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.

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

Stenetorp, Pontus (Dec. 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.

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 (Sept. 2017). "Learning Distributions of Meant Color". In: *ArXiv e-prints*. arXiv: `1709.09360 [cs.CL]`.

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.

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

Chapter 5

Conclusion

In these three literature review chapters I have introduced the core notions used in modern machine learning based natural language processing. Each of the papers in the following sections also include there own background and related works sections detailing specifically relevant works to their area.

Part II

Contrasting Classical and Neural Representations

Chapter 6

Introduction

In the works in this section I contrast classical representation approaches with more modern neural network-based representational approaches.

Chapter 7

How Well Sentence Embeddings Capture Meaning

Lyndon White
lyndon.white@research.uwa.edu.au

Roberto Togneri
roberto.togneri@uwa.edu.au

Wei Liu
wei.liu@uwa.edu.au

Mohammed Bennamoun
mohammed.bennamoun@uwa.edu.au

The University of Western Australia
35 Stirling Highway, Crawley, Western Australia

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.

General Terms

Measurement, Performance, Experimentation

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Abstracting methods, Linguistic processing*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Language parsing and understanding*

ADCS, December 08-09, 2015, Parramatta, NSW, Australia
© 2015 ACM. ISBN 978-1-4503-4040-3/15/12…$15.00
DOI: http://dx.doi.org/10.1145/2838931.2838932

Keywords

Semantic vector space representations, semantic consistency evaluation, sentence embeddings, word embeddings

1. INTRODUCTION

Sentence embeddings are often referred to as semantic vector space representations [7]. 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 [10]; recreation of input, as in the auto-encoders of [22, 19] and [7]; and syntactic structural classification [18, 21]. 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 [22, 20, 10] and paraphrase detection [19]. 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) [3] and Opinosis [5] 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 [6] and of Ritter et. al. [17]. 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 §2 discusses the existing models being assessed, the traditional assessment methods, and the aforementioned more recent semantic correctness based assessments. Section §3 describes the processes by which the models are evaluated using our new method, and the parameters used in the evaluation. Section §4 continues into more details on the development of the evaluation corpora for the semantic classification evaluation task. Section §5 details the results from evaluating the models and discusses the implications for their semantic consistency. Section §6 closes the paper and suggests new directions for development.

2. BACKGROUND

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. [19]; and the two word content predictive models, Distributed Memory (PV-DM) and Distributed Bag of Words by Le and Mikolov [10]. In addition to these advanced sentence embedding models, a simple average of word embeddings, from Mikolov et. al. [13], 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.

2.1.1 Unfolding Recursive Auto-Encoder (URAE)

The Unfolding Recursive Autoencoder (URAE) [19] 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.

2.1.2 PV-DM

The Distributed Memory Paragraph Vectors (PV-DM) [10] method is based on an extension of the Continuous Bag-of-Words word-embedding model [12]. 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 [10]. 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.

2.1.3 PV-DBOW

Distributed Bag of Words Paragraph Vectors (PV-DBOW) [10], is based on the Skip-gram model for word-embeddings, also from [12]. 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.

2.1.4 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 [10]. The sum of word embeddings first considered in [13] for short phrases, it was found to be an effective model for summarization in [9]. 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.

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, [3] 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 [22] and for the paragraph vector models in [10]. Sentiment Analysis is classifying a text as positive or negative, or assigning a score as in the Sentiment Treebank

[23]. 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 [11] and [1]. Much more recently it has been used to assess the convolution neural networks of [25], where the articles of several news corpora were classified into categories such as "Sports", "Business" and "Entertainment". A huge spectrum of different sentences are assigned to the same topic. It is thus too board and insufficiently specific to evaluate the consistency of meanings. Information retrieval can be seen as the inverse of the document classification task.

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

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

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.[14] demonstrated that the word-embedding models are semantically consistent by showing that the semantic relations between words were reflected as a linear offset in the vector space. That is to say, for embeddings $\tilde{x}_a$, $\tilde{x}_b$, $\tilde{x}_c$, $\tilde{x}_d$ corresponding to words A, B, C and D, respectively; it was tested that if for a strong relationship matching between A/B and C/D, then the offset vector would be approximately equal: $\tilde{x}_b - \tilde{x}_a \approxeq \tilde{x}_d - \tilde{x}_c$. Rearranging this in word space gets the often quoted example of $\texttt{King} - \texttt{Man} + \texttt{Woman} \approxeq \texttt{Queen}$, As man is to woman, king is to queen. In the rating task as described by [8], the goal is to rank such analogous word pairs based on the degree the relation matches. Thus to evaluate the word-embedding model using this task, it was a matter of sorting closeness of the corresponding offset vectors. Surprisingly strong results were found on this task[14]. It was thus demonstrated that word embeddings were not simply semantically consistent, but more so that this consistency was displayed as local linearity. This result gives confidence in the semantic quality of the word embeddings. However, this relationship analogy test cannot be performed for sentence embeddings.

Gershman et. al. [6], 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. [17], 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.

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 1: Corpus Preparation,

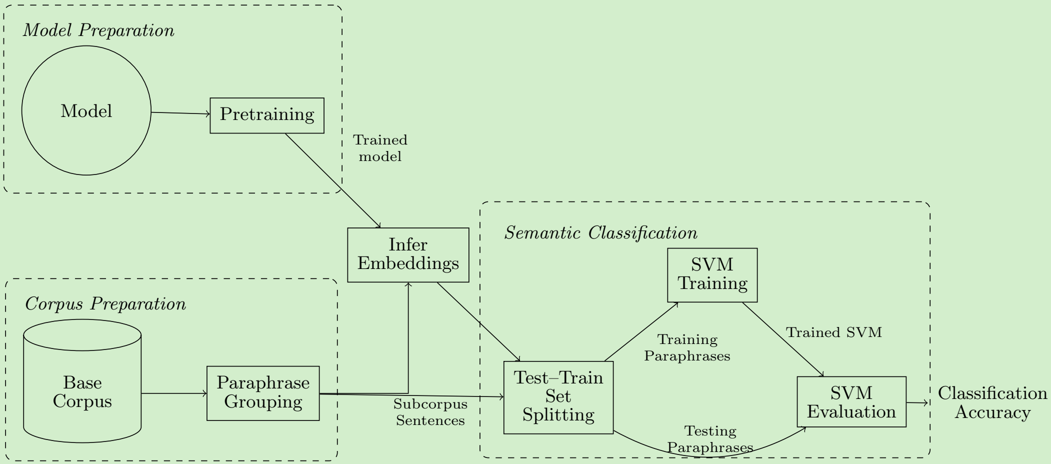


Figure 1: Process Diagram for the Evaluation of Semantic Consistency via our method

Model Preparation, and the Semantic Classification task itself.

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) [3] and of the Opinosis [5] 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.

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.

3.2.1 Unfolding Recursive Auto-Encoder (URAE)

In this evaluation we make use of the pretrained network that Socher et. al. have graciously made available[1], full information is available in the paper[19]. It is initialized on the unsupervised Collobert and Weston word embeddings[2], 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.

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

Both PV-DM and PV-DBOW, were evaluated using the GenSim implementation [16] 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.

3.2.3 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 [13]. 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.

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

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

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

3.2.4 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 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.

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 [15] was used to orchestrate the cross-validation and to interface with the LibLinear SVM implementation [4]. 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.

4. CORPUS CONSTRUCTION

4.1 Microsoft Research Paraphrased Grouped Subcorpus

The MSRP corpus is a very well established data set for the paraphrase detection task [3]. Sentences are presented as pairs which are either paraphrases, or not. A significant number of paraphrases appear in multiple different pairings. Using this information, groups of paraphrases can be formed.

The corpus was partitioned according to sentence meaning by taking the symmetric and transitive closures the set of paraphrase pairs. For example if sentences A, B, C and D were present in the original corpus as paraphrase pairs: A, B, D, A and B, C then the paraphrase group $\{A, B, C, D\}$ is found. Any paraphrase groups containing less than 3 phrases were discarded. The resulting sub-corpus has the breakdown as shown in Figure 2.

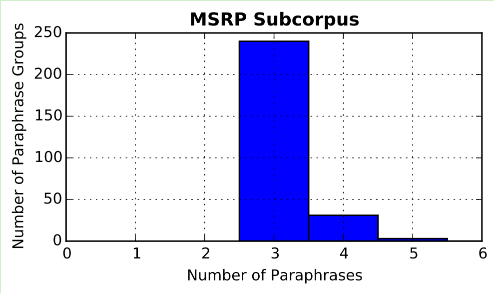


Figure 2: Break down of how many paraphrases groups are present in the MSRP subcorpus of which sizes.It contains a total of 859 unique sentences, broken up into 273 paraphrase groups.

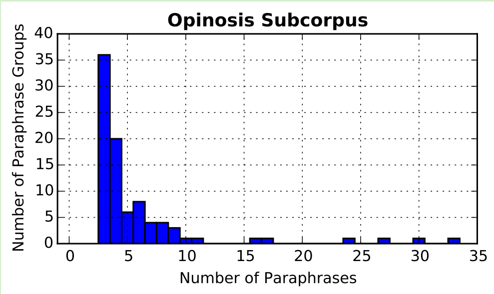


Figure 3: Break down of how many paraphrases groups are present in the Opinosis subcorpus of which sizes. It contains a total of 521 unique sentences, broken up into 89 paraphrase groups.

4.2 Opinosis Paraphrase Grouped Subcorpus

The Opinosis Corpus[5] 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.

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

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

The final point is the most significant change from the practices apparent in the construction of the MSRP corpus. Sentences with differing or additional information were classified as non-paraphrases. This requirement comes from the definition of bidirectional entailment. For example, *"The staff were friendly and polite."*, *"The staff were polite."* and *"The staff 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." [3]. 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 3.

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

5. RESULTS AND DISCUSSION

5.1 Classification Results and Discussion

The results of performing the evaluation method described in Section §3 are shown in Table 1.

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

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 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 [9] 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 [19]. The exact same model from [19] 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 [19], 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.

5.2 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 4 show the agreement in errors. Here misclassification agreement is given as an approximation to $P(m_1(x) = m_2(x) \mid m_1(x) \neq y \wedge m_2(x) \neq y)$, 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

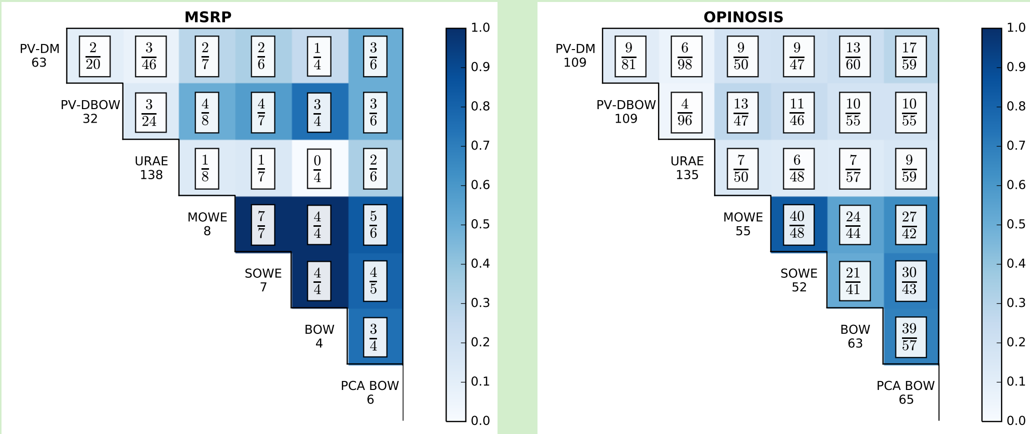


Figure 4: The misclassification agreement between each of the models for the MSRP (left) and Opinosis (right) subcorpora. Below each model name is the total mistakes made. The denominator of each fraction is the number of test cases incorrectly classified by both models. The numerator is the portion of those misclassifications which were classified in the same (incorrect) way by both models. The shading is in-proportion to that fraction.

models with the ground truth.

The word based models showed significant agreement. Unsurprisingly MOWE and SOWE have almost complete agreement in both evaluations. The other models showed less agreement – while they got many of the same cases wrong the models produced different misclassifications. This overall suggests that the various full sentence models are producing substantially dissimilar maps from meaning to vector space. Thus it seems reasonable that using a ensemble approach between multiple sentence models and one word-based model would produce strong results. Yin and Schütze [24] found this successful when combining different word embedding models.

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.[17] discussed in section 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 [17] 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.

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.[17]. 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.

Acknowledgement

This research is supported by the Australian Postgraduate Award, and partially funded by Australian Research Council DP150102405 and LP110100050.

7. REFERENCES

[1] H. Borko and M. Bernick. Automatic document classification. *Journal of the ACM (JACM)*, 10(2):151–162, 1963.

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

[3] W. B. Dolan and C. Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*. Asia Federation of Natural Language Processing, 2005.

[4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[5] K. Ganesan, C. Zhai, and J. Han. Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 340–348. Association for Computational Linguistics, 2010.

[6] S. J. Gershman and J. B. Tenenbaum. Phrase similarity in humans and machines. *Proceedings of the 37th Annual Conference of the Cognitive Science Society*, 2015.

[7] M. Iyyer, J. Boyd-Graber, and H. D. III. Generating sentences from semantic vector space representations. In *NIPS Workshop on Learning Semantics*, 2014.

[8] D. A. Jurgens, P. D. Turney, S. M. Mohammad, and K. J. Holyoak. Semeval-2012 task 2: Measuring degrees of relational similarity. In *Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 356–364. Association for Computational Linguistics, 2012.

[9] M. Kågebäck, O. Mogren, N. Tahmasebi, and D. Dubhashi. Extractive summarization using continuous vector space models. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, pages 31–39, 2014.

[10] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[11] M. E. Maron. Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417, 1961.

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

[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

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

[15] F. Pedregosa, 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. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[17] S. Ritter, C. Long, D. Paperno, M. Baroni, M. Botvinick, and A. Goldberg. Leveraging preposition ambiguity to assess compositional distributional models of semantics. *The Fourth Joint Conference on Lexical and Computational Semantics*, 2015.

[18] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *ACL*. 2013.

[19] R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems 24*. 2011.

[20] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. 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*, pages 1201–1211. Association for Computational Linguistics, 2012.

[21] R. Socher, C. D. Manning, and A. Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9, 2010.

[22] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[23] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[24] W. Yin and H. Schütze. Learning word meta-embeddings by using ensembles of embedding sets. Aug. 2015.

[25] X. Zhang and Y. LeCun. Text understanding from scratch. *CoRR*, Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, 2015.

NovelPerspective

Lyndon White, Roberto Togneri, Wei Liu, and **Mohammed Bennamoun**
lyndon.white@research.uwa.edu.au, roberto.togneri@uwa.edu.au,
wei.liu@uwa.edu.au, and mohammed.bennamoun@uwa.edu.au
The University of Western Australia. 35 Stirling Highway, Crawley, Western Australia

Abstract

We present NovelPerspective: a proof-of-concept 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.

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 et al., 2010; Wohlgenannt et al., 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" 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.

2 Character Classification Systems

The full NovelPerspective pipeline is shown in Figure 1. The core character classification step (step 3), is detailed in Figure 2. In this step the raw text is first enriched with parts of speech, and named entity tags. From this, features are extracted for each named entity. These feature vectors are used to score the entities for the most-likely POV character. The highest scoring character is returned by the system. The different systems presented modify the **Feature Extraction** and **Character Scoring** steps. A broadly similar idea, for detecting the focus location of news articles, was presented by (Imani et al., 2017).

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

2.1.1 "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.

2.1.2 "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

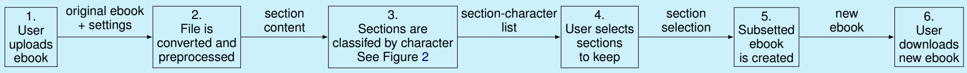


Figure 1: The full NovelPerspective pipeline. Note that step 5 uses the original ebook to subset.

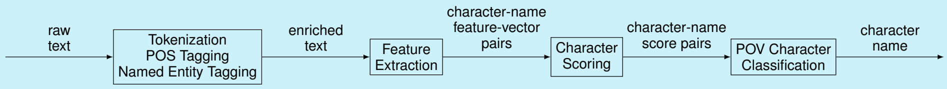


Figure 2: The general structure of the character classification systems. This repeated for each section of the book during step 3 of the full pipeline shown in Figure 1.

are about the POV character's relationship with a secondary character. In such cases the secondary character may be mentioned more often.

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 2.2.1). 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.

2.2.1 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 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., 2013). This has a total of 600 features.

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

2.2.2 Classifier

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

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

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

3 Experimental Setup

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 4 we consider the use of each as a training and testing dataset. In the online demonstration (Section 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.

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

Table 1: The number of chapters and point of view characters for each dataset.

The total counts of chapters and characters in the datasets, after preprocessing, is shown in Table 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.

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.

3.3 Implementation

The implementation 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 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 charac-

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

ter 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.

4 Results and Discussion

Our evaluation results are shown in Table 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 out-perform each on their own. For many methods training on just one dataset resulted in better results. We believe

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

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 5), we deploy models trained on all three datasets combined.

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

Table 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 would likely cause overfitting).

The results do not show a clear advantage to either machine learning feature set. Both the classical features and the word embeddings work well. It seems that the classical feature are more robust; both with smaller training sets (like SOC), and with more difficult test sets (like WOT). With more training data, it would be reasonable to more features (e.g. additional adjacent words), by doing this it may be that word embedding may prove better.

5 Demonstration System

We have deployed an online prototype to https://white.ucc.asn.au/tools/np. A video of 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 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 Conclusion

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

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

[4] http://cherrypy.org/

References

Bird, S. and Loper, E. (2004). Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

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

Elson, D. K., Dames, N., and McKeown, K. R. (2010). Extracting social networks from literary fiction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 138–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

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

White, L., Togneri, R., Liu, W., and Bennamoun, M. (2015). How well sentence embeddings capture meaning. In *Proceedings of the 20th Australasian Document Computing Symposium*, ADCS '15, pages 9:1–9:8. ACM.

Wohlgenannt, G., Chernyak, E., and Ilvovsky, D. (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)*, pages 18–25.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (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*, pages 19–27.

ACL 2018 Submission ***. Confidential Review Copy. DO NOT DISTRIBUTE.

NovelPerspective: Supplementary Materials

Anonymous ACL submission

Shown here are screen-shots of NovelPerspective web-app. They are taken from the online prototype; which is available at https://white.ucc.asn.au/tools/np. In this example we have used Brandon Sanderson's "The Way of Kings". A video showing this same content can be found at https://youtu.be/iu41pUF4wTY.

They show the screens the used goes through The user uploads their book and sets the configuration options as shown in Figure 1. Then the preprocessing is done as shown in Figure 2. The user interface for selecting which sections to keep (based on the POV characters) is shown in Figures 3 to 5.

ACL 2018 Submission ***. Confidential Review Copy. DO NOT DISTRIBUTE.

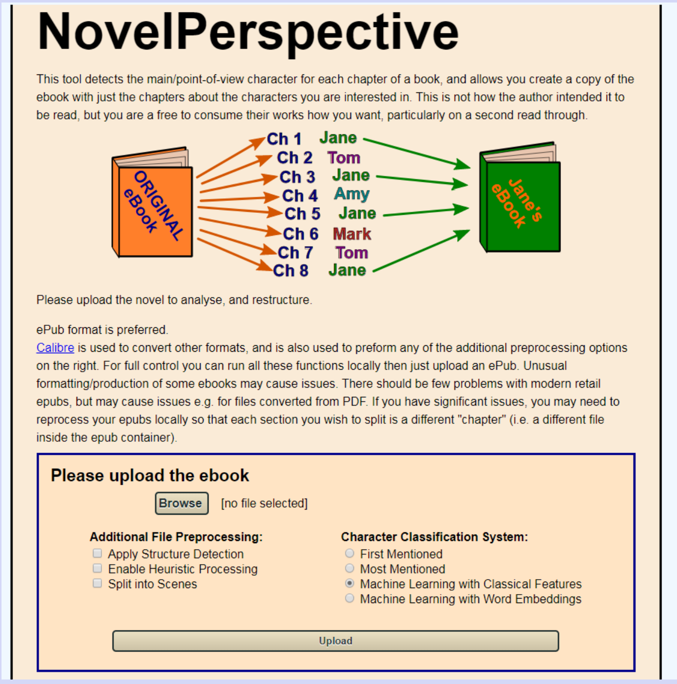


Figure 1: The start page for the NovelPerspectiveApp. Here the users uploads their book, and selects the POV character detection system, and any preprocessing options.

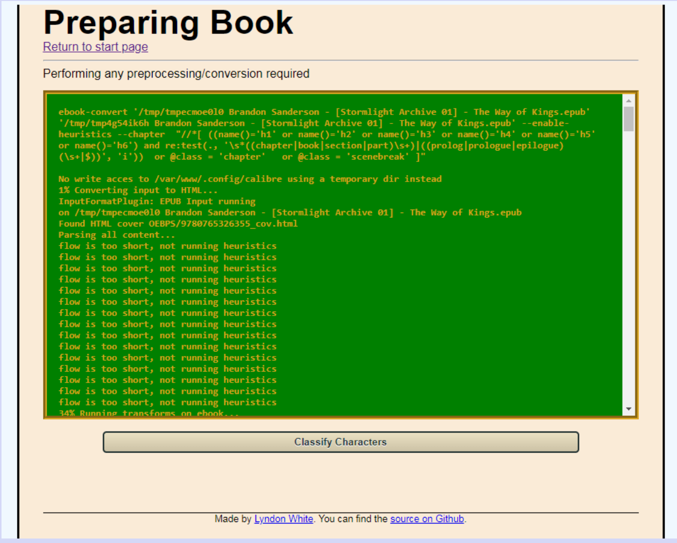


Figure 2: The preparation page. Here the Calibre ebook-conversion tool [2] is uses to convert the input into an epub, and to perform any requested preprocessing. The output of that tool is streamed to the user.

2

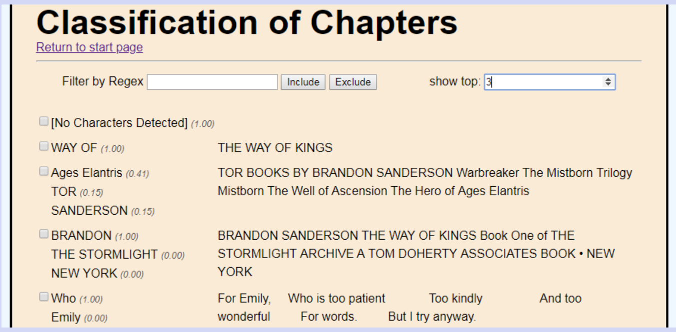


Figure 3: The section selection controls. The first few sections are preamble such as the title page and the dedication. Shown at the top are the show-top, and the regex selection controls. **On the right:** The user can use the show-top control to change the number of possible main characters to display. This is by default one for just the most likely POV character. For demonstration here we have set it to show the top three most likely. **On the left:** The user can use regex patterns for character names to include or exclude the selection of sections. They are matched against the currently displayed names. By alternating the manipulation of show-top, and the regex selection controls, the user can achieve a fine degree of control over their selection.

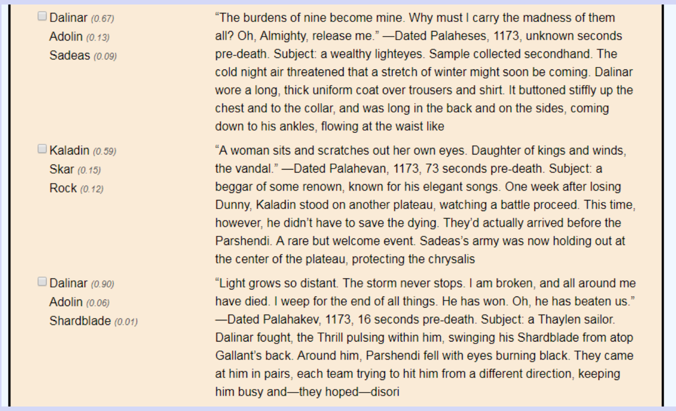


Figure 4: The main selection selection interface showing the sections that can be selected. This screenshot is taken from the same page as Figure 3, but from the middle of the page. Each row corresponds to a particular section of the book. On the left is shown the names of the characters who are most likely to be the POV character. The numbers of characters shown is determined by the show-top control. In smaller writing next to each is the characters score as determining by the classification method. On the right is shown the start of the section

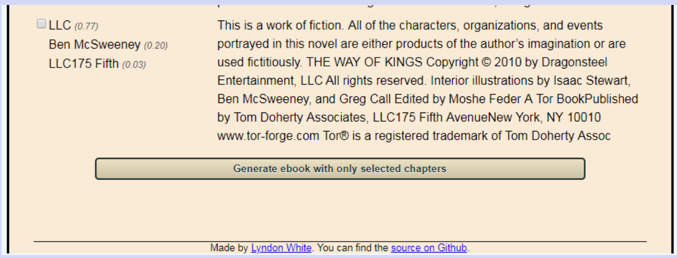


Figure 5: Generating a book from the main section selection interface. This screen-shot is taken from the same page as Figure 3, but from the end of the page. Once the user has made their selections, they press the Generate ebook button shown to create and download an ebook with only their selected content.

Learning Distributions of Meant Color

Anonymous ACL submission

Abstract

When a speaker says the name of a color, the color that they picture is not necessarily the same as the listener imagines. Color is a grounded semantic task, but that grounding is not a mapping of a single word (or phrase) to a single point in color-space. Proper understanding of color language requires the capacity to map a sequence of words to a probability distribution in color-space. A distribution is required as there is no clear agreement between people as to what a particular color describes – different people have a different idea of what it means to be "very dark orange". We propose a novel GRU-based model to handle this case. Learning how each word in a color name contributes to the color described, allows for knowledge sharing between uses of the words in different color names. This knowledge sharing significantly improves predicative capacity for color names with sparse training data. The extreme case of this challenge in data sparsity is for color names without any direct training data. Our model is able to predict reasonable distributions for these cases, as evaluated on a held-out dataset consisting only of such terms.

1 Introduction

Color understanding is an important subtask in natural language understanding. It is a challenging domain, due to ambiguity, multiple roles taken by the same words, the many modifiers, and shades of meaning. Due to its difficulty, texts containing color descriptions such as `the flower has petals that are bright pinkish purple with white stigma` are used as demonstrations for state of the art image generation systems (Reed et al., 2016; Mansimov et al., 2015).

The core focus of the work we present here is addressing these linguistic phenomena around the descriptions of the color, in a single patch, as represented in a color-space such as HSV (Smith, 1978). Issues of illumination and perceived color based on visual context are considered out of the scope.

Consider that the word `tan` may mean one of many colors for different people in different circumstances: ranging from the bronze of a tanned sunbather, to the brown of tanned leather; `green` may mean anything from `aquamarine` to `forest green`; and even `forest green` may mean the rich shades of a rain-forest, or the near grey of the Australian bush. Thus the color intended cannot be uniquely inferred from the color name. Without further context, it does nevertheless remain possible to estimate likelihoods of which colors are be intended based on the population's use of the words. The primary aim of this work is to map a sequence of color description words to a probability distribution over a color-space. This is required for a proper understanding of color language.

Estimating color probabilities has a clear use as a subsystem in many systems. For example, in a human-interfacing system, when asked to select the `dark bluish green` object, each object can be ranked based on how likely its color is according to the distribution. This way if extra information eliminates the most-likely object, the second most likely object can immediately be determined. Further, if the probability of the color of the object being described by the user input is known, a threshold can be set to report that no object is found, or to ask for additional information. More generally, the distribution based on the color name alone can be used as a prior probability and combined with additional context information to yield better predictions.

Proper understanding requires considering *the color intended* as a random variable. In other words, a color name should map to a distribution, not just a single point or region. For a given color name, any number of points in the color-space could be intended, with some being more or less likely than others. Or equivalently, up to interpretation, it may intend a region but the likelihood of what points are covered is variable and uncertain. This distribution is often multimodal and has high and asymmetrical variance, which further renders regression to a single point, as has been done by Kawakami et al. (2016), unsuitable. We estimate a probability distribution over the color-space. To qualify our estimate of the distribution we discretize the space into a large number of patches, and produce an output much like a histogram. This allows us to take advantage of the well-known softmax based methods for estimating a probability mass distribution using a neural network.

This understanding of color language also requires a model capable of understanding linguistic compositionality. It must understand how modifiers such as `dark` modify basic colors; and how other modifiers such as `very` would interact as a modifier to modifiers. It also must understand the functioning of affixes such as `-ish` in `greenish`. This compositional understanding is needed both as a point of theory and practically. Practically, the generalisation ability from a compositional model allows it to handle color descriptions not seen in training. Due to the combinatorial nature of language, a data-sparsity problem exists: that for a large number of word combinations there are few examples in any given corpus. This is a well-known issue in n-gram language modelling (Kneser and Ney, 1995; Chen and Goodman, 1996; Rosenfeld, 2000). To handle this we take inspiration from a solution used in that area: the use of a recurrent neural network to process each color description as a compositional sequence of tokens (Mikolov et al., 2011). Processing per token allows for knowledge sharing between uses of the tokens in different terms, thus overcoming data sparsity programs (Bengio et al., 2003). Including, the extreme case of there being no direct training data at all.

The core contribution of this work is a novel method for estimating probability distributions over color-space for a color name, which is able to generalise to estimate distributions for color descriptions which are never seen during training. To handle distribution estimation we employ a discretization and blurring procedure. We define a GRU-based neural network to learn the compositional relationship from the term sequences describing the colors, overcoming the data-sparsity problem. We call this model the Color Distribution Estimation from Sequences of Terms (CDEST) model. As, to our knowledge, there is no existing work on estimating distributions from color-names, in order to evaluate the CDEST model we also define a histogram-based baseline method, which while lacking the generalisation capacity, more directly extracts the information from the training data.

2 Related Work

The understanding of color names has long been a concern of psycholinguistics and anthropology (Berlin and Kay, 1969; Heider, 1972; Heider and Olivier, 1972; Mylonas et al., 2015). It is thus no surprise that there should be a corresponds field of research in natural language processing.

The earliest works revolve around explicit color dictionaries. This includes the ISCC-NBS color system (Kelly et al., 1955) of 26 words, including modifiers, that are composed according to a context free grammar such that phrases are mapped to single points in the color-space; and the simpler, non-compositional, 11 basic colors of Berlin and Kay (1969). Works including Berk et al. (1982); Conway (1992); ele Lammens (1994); Mojsilovic (2005); Menegaz et al. (2007); Van De Weijer et al. (2009) which propose methods for the automatic mapping of colors to and from these small manually defined sets of colors. We note that Menegaz et al. (2007); Van De Weijer et al. (2009) both propose systems that discretize the color-space, though to a much courser level than we consider in this work.

More recent works, including the work presented here, function with much larger number of colors, larger vocabularies, and larger pools of respondents. In particular making uses of the large Munroe dataset Munroe (2010), as we do here. This allows a data driven approach towards the modelling.

McMahan and Stone (2015) and Meo et al. (2014) present color naming methods, mapping from colors to to their names, the reverse of our task. These works are based on defining fuzzy rectangular distributions in the color-space to cover the distribution estimated from the data, which are used in a Bayesian system to non-compositionally determine the color name. Monroe et al. (2016) maps a point in the color-space, to a sequence of distributions over color terms. They extends beyond, all prior color naming systems to produce a compositional color namer based on the Munroe dataset. Their method uses a recurrent neural network (RNN), which takes as input a color-space point, and the previous output word, and gives a probability of the next word to be output – this is a conditional language model. Our proposed CDEST model is the direct inverse of their conditional language model, CDEST use a RNN to map a sequence of color terms to a distribution over colors.

Kawakami et al. (2016) also propose a compositional color naming model. They use a per-character RNN and a variational autoencoder approach. It is in principle very similar to Monroe et al. (2016), but functioning on a character, rather than a word level. The work by Kawakami et al. also includes a method for generating colors. However it generates just single points, rather than distributions. This has significant limitations as discussed in Section 1, which our work attempts to overcome by modeling the distributions.

Monroe et al. (2017) presents a neural network solution to a communication game, where a speaker is presented with three colors and asked to describe one of them, and the listener is to work out which is being described. Speaker and listener models are trained, using LSTM-based decoders and encoders respectively. The final time-step of their model produces a 100 dimensional representation of the description provided. From this, a Gaussian distributed score function is calculated, over a high dimensional color-space from Monroe et al. (2016), which is then used to score each of the three options. While this method does work with a probability distribution, as a step in its goal, this distribution is always both symmetric and unimodal – albeit in a high-dimensional color-space. To the best of our knowledge no current work proposes as a distribution estimation system such as we describe in this paper.

3 Color Distribution Estimation Framework

We define two models for the estimation of colors from textual descriptions. A baseline histogram-based model and the GRU-based CDEST model. The baseline model estimates the distribution based on averaging the discretized observations of colors in the training set for each input color description. It cannot handle combinations of terms not seen during training as there is no data to average. The CDEST model relies on using machine learning to learn the relationship between words and the color distribution; and is trained on the same observations used in the baseline model. As it is learning a relationship between words and the color-space probability output, it can handle inputs made up of any words that were seen during training, even if the whole color description has never been used before. Both models rely on the same assumption of conditional independence, and the same method for discretization.

3.1 Conditional Independence Assumption

We make the assumption that given the name of the color, the distribution of the H, S and V channels are independent. That is to say, it is assumed if the color name is known, then knowing the value of one channel would not provide any additional information as to the value of the other two channels. The same assumption is made, though not remarked upon, in Meo et al. (2014) and McMahan and Stone (2015). This assumption of conditional independence allows considerable saving in computational resources. Approximating the 3D joint distribution as the product of three 1D distributions decreases the space complexity from $O(n^3)$ to $O(n)$ in the discretized step that follows.

Superficial checks were carried out on the accuracy of this assumption. Spearman's correlation on the training data suggests that for over three quarters of all color names, there is only weak correlation between the channels (Q3 = 0.187). However, this measure underestimates correlation for values that have circular relative value, such as hue. HSV had the lowest correlation by a large margin of the 16 color-spaces evaluated. Full details, including the table of correlations, are available in supplementary materials. These results are suggestive, rather than solidly indicative, on the degree of correctness of the conditional independence assumption. We consider the assumption sufficient for this investigation.

3.2 Discretization and Blurring

The core problem is to estimate a continuous probability distributions, conditional on the color name. Estimating a discrete conditional distributions is a significantly more studied application of neural networks – this is the basic function of any softmax classifier. To simplify the problem, we therefore transform it to be a discrete distribution estimation task, by discretizing the color-space. Discretization to a resolution of 64 and 256 bins per channel is considered.

Discretization to resolution n is the process by which a scalar observation[1] x from one of the continuous color channels (hue, saturation or value) is converted into an n-vector with the properties expected of a probability mass function. A naïve

[1] In the Munroe dataset, the provided HSV values are scaled to between 0 and 1 in all channels. We make use of this convention throughout this paper, and in our implementation.

approach is one-hot binning:

$$\Omega_n^{1hot}(x) = \left(\begin{cases} 1 & \text{if } \frac{i-1}{n} < x \leq \frac{i}{n} \\ 0 & \text{otherwise} \end{cases} \right)_{i=1}^{i=n}$$

This gives an n-vector that is zero everywhere, except for the element corresponding to the patch of color-space that the value x lies within. Discretization in this way loses all notion of continuousness of the color-space. In truth the distribution in color-space is intrinsically continuous – this comes as a logical consequence of human color sensitivity being continuous (Stockman et al., 1999). Points near each other in the color-space should have similar probabilities of being the intended color for a color name. While discretization inevitably renders the space discrete, it is desirable to bring back this notion of smoothness as prior knowledge.

We enhance the training data by adding a blur during discretization. Consider $\mathcal{D}(\mu, \sigma^2)$ some unimodal distribution, characterised by having an expected value μ and a variance parameter σ^2. For saturation and value this is a truncated Gaussian. Hue can elegantly be handled using a wrap-around Gaussian. We write $P_{\mathcal{D}}(y_1 < Y \leq y_2 \mid M=\mu, \Sigma=\sigma)$ to mean the probability of a value distributed according to $\mathcal{D}(\mu,\sigma^2)$ being in the patch bordered by y_1 and y_2. Using this, the blurred-binning function is defined:

$$\Omega_n^{blur}(x,\mathcal{D},\sigma) = \left(P_{\mathcal{D}}\left(\frac{i-1}{n} < Y \leq \frac{i}{n} \mid M=x, \Sigma=\sigma \right) \right)_{i=1}^{i=n}$$

This function maps points x in the continuous color-space, to probability mass vectors of length n. The majority of the mass will be in the bin that the value x would be in, but some will be shared with the bins either side, and further.

By applying more or less blurring to the training data, the priority of smoothness v.s. exact matching is controlled. Considering the limits: for all $\mathcal{D}$ and values x: $\lim_{\sigma\to 0} \Omega_n^{blur}(x,\mathcal{D},\sigma) = \Omega_n^{1hot}(x)$, and $\lim_{\sigma\to\infty} \Omega_n^{blur}(x,\mathcal{D},\sigma) = \left(\frac{1}{n}\right)_{i=1}^{i=n}$ (uniform). A coarse parameter sweep on the value of σ was carried out using the development portion of the dataset (see Section 4.1). Best results were found for $\sigma = \frac{1}{2n}$. For a training point that would be at the center of a bin, this roughly corresponds to 68.3% of the probably mass assigned to the central bin, 15.7% assigned to adjacent bins, and the remaining 0.3% distributed to the remaining bins. All results presented here are for this level of blurring.

Discretizing the data is is a useful solution used in several other machine learning systems. Oord et al. (2016); van den Oord et al. (2016) apply a similar discretization step and found their method to outperforming the more complex continuous distribution outputs. These works did not employ a blurring-step. We found the blurring step to consistently improve results for all models during preliminary investigation using the development dataset. This is expected as a blurred discrete distribution captures some of the notions of continuity that a truly continuous output distribution would intrinsically feature.

We note that a truly continuous output is pragmatically unnecessary as 24-bit color (as was used in the survey) can have all information captured by a 256 bin quantization per channel. 24 bit color allows for a total of 2^{24} colors to be represented, and even 1 hot encoding for each of the 256 bin quantized channels allows for the same.

3.3 Baseline Model

While the main interest in this work is in compositionally modelling the color language, we also define a non-compositional baseline model to allow for comparison. This model loosely resembles the histogram model discussed in Meo et al. (2014) and McMahan and Stone (2015). Existing works do not aim to estimate a general distribution, and they are therefore unsuitable for comparison. Our baseline must be able to estimate multimodal and asymmetric color distributions.

The baseline is defined using the the element-wise mean of discretized training observations, with add-one smoothing. During our investigations we found that without the add-one smoothing the baseline would predict a probability of zero for some observations in the development dataset. Applying add-one smoothing to each output distribution solves this.

For the training data $V \subset [0,1]^3 \times T$, where $[0,1]^3 \subset \mathbb{R}^3$ is the scaled HSV color-space, and T is the natural language space. The subset of the training data for the description $t \in T$ is given by $V_{|t} = \{\tilde{v}_i \mid (\tilde{v}_i, t_i) \in V \wedge t_i = t\}$. Per channel $c \in \{H,S,V\}$ the baseline model is defined by:

$$q_c(x_c \mid t) = \frac{\sum\limits_{\forall (v_H,v_S,v_V) \in V_{|t}} \Omega_n^{blur}(v_c,\mathcal{D}_c,\sigma) \cdot \Omega_n^{1hot}(x_c) + 1}{\left|V_{|t}\right| + n}$$

In this equation taking the dot-product with $\Omega_n^{1hot}(x_c)$ is selecting the bin containing x_c. Note the distinction between x_c and v_c: x_c is the point being queried, whereas v_c is a point from the training set. By the conditional independence assumption the overall baseline model is given by: $q(x_H,x_S,x_V \mid t) = \prod_{c\in\{H,S,V\}} q_c(x_c \mid t)$

The baseline model can be used to predict distribu-

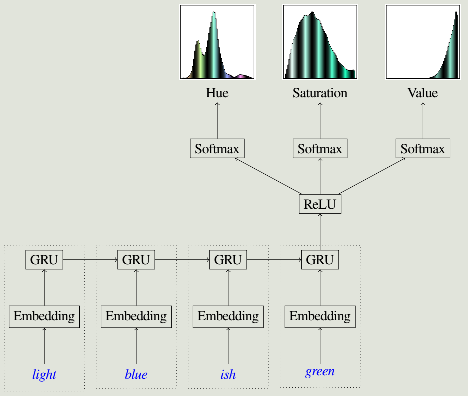


Figure 1: The CDEST model for predicting the color-space probability distributions of color. The section in the dotted-boxes is repeated for each time step.

tions for all color descriptions in the training set. This is inferior in generalisability to the CDEST model, which can handle any combination of tokens from the training set. We suggest that the baseline model is strong and reasonable. It is a much simpler modelling problem as it does not have a requirement to learn the how the multiple terms in the color name are compositionally combined. It directly captures the information from the training set. If the CDEST model can match its performance, that would at least show that it was capturing the information from the training data. If it can also have similar performance for cases that do require compositional understanding (see Section 4.2), that would show that it is indeed achieving the goal of properly modelling the language use.

3.4 CDEST Model

The CDEST model is an RNN which learns the compositional interactions of the terms making up a color description, to output a distribution estimate in color-space. The general structure of this network, shown in Figure 1 is similar to Monroe et al. (2016), or indeed to most other word sequence learning models. Each word is first transformed to an embedding representation. This representation is trained with the rest of the network allowing per word information to be efficiently learned. The embedding is used as the input for a Gated Recurrent Unit (GRU) (Cho et al., 2014). The output of the last time-step is fed to a Rectified Linear Unit (ReLU) (Dahl et al., 2013). Finally, the output of the ReLU is the shared input for three distinct softmax output layers – one for each of hue, saturation and value. These outputs are vectors $\hat{y}_H(t)$, $\hat{y}_s(t)$, and $\hat{y}_V(t)$. Using the conditional independence assumption the probability estimate is given by:

$$\hat{p}(x_H{,}x_S{,}x_V \,|\, t) = \prod_{c\in\{H,S,V\}} \hat{y}_c(t)\cdot\Omega_n^{1hot}(x_c))$$

As in the baseline model, the dot-product with $\Omega_n^{1hot}(x_c)$ serves to select the bin containing x_c.

The distinguishing features of this model compared to other word sequence learning models, is the use of GRU, rather than Long Short Term Memory (LSTM), and the three output layers.

We chose GRU as the basis of our reused structure in the recurrent network because it has fewer parameters to learn than the more established LSTM. It has generally been found to preform similarly well to LSTM (Chung et al., 2014); including on the color naming problem (Monroe et al., 2016). A component for processing per-term such as the GRU, is essential in allowing the model to learn the compositional function of each term, and thus to learn to handle color descriptions from outside the training set.

The three output layers are used to predict the discretized distributions for the three channels. Separating them like this requires a conditional independence assumption (see Section 3.1). The network is trained to minimize the sum of the three cross-entropy losses for these output layers. Similar multiple output layers as used in multitask learning (Caruana, 1997; Collobert and Weston, 2008). The layers prior to the output are shared, allowing common knowledge to be shared.

4 Experimental Setup

4.1 Data Preparation and Tokenization

We make use of the Munroe dataset as prepared by McMahan and Stone (2015) from the results of the XKCD color survey. The XKCD color survey (Munroe, 2010), collected over 3.4 million observations from over 222,500 respondents. McMahan and Stone take a subset from Munroe's full survey, by restricting it to the responses from native English speakers, and removing rare color names with less than 100 uses. This gives a total of 2,176,417 observations and 829 color names. They also define a standard test, development and train split.

In the dataset each observation is a textual color description, paired with a point in HSV color-space. We tokenized the textual color descriptions into separate words and affixes, using a short list of word replacement rules. Beyond simply breaking up a description `greenish blue` into words: `greenish` and `blue`, the suffixes `-ish` and

5

-y are also separated into their own tokens: `green`, `ish`, `blue`. Hyphens are also treated as their own tokens: `blue-green` becomes `blue`, `-`, `green`. The beginning and end of the color description is not demarcated with any form of marker token. Using this tokenization, each description is split into up to four tokens. This results in a total of 311 unique tokens used by the CDEST model. The baseline model does not function per token, and so uses the original 829 descriptions directly.

4.2 Extrapolation Sub-Dataset

The primary goal in constructing the CDEST model was for it to be able to to predict the distribution for never before seen descriptions of colors. For example, based on the learned understanding of `salmon` and of `bright`, from examples like `bright green` and `bright red`, our system can suggest the distribution in the color-space of `bright salmon`, even though that description never occurs in the training data. This would demonstrating proper compositional learning. To evaluate this generalisation capacity, we define an extrapolation sub-dataset. This is defined by selecting the rarest 100 color descriptions from the dataset, with the restriction that every token in a selected description must still have at least 8 uses in other descriptions. The selected examples include multi-token descriptions such as: `bright yellow green` and also single tokens that occur more commonly as modifiers than as stand-alone descriptions such as `pale`. The test and development datasets are restricted to contain only observations of these selected color descriptions. Conversely, the training set has no observations of these color descriptions. This produces a dataset suitable for evaluating the capacity of our model to estimate the distributions for color descriptions not seen in training. A similar approach was used in Atzmon et al. (2016).

4.3 CDEST Model Parameters

All hidden layers have width 128, except the embedding layer which has width 16. These values were found by a coarse search of the hyper-parameters using the development dataset with the output resolution being 64 bins. These parameters were also used for the 256 bin output resolution, though we suggest increasing the hidden layer size would give additional benefit for the higher output resolution case. During the hyper-parameter search, it was noted that the accuracy continued to improve as the hidden layer width was increased. However significantly diminishing returns in terms of training time v.s. accuracy lead us to limit the hidden layer sizes. Dropout (Srivastava et al., 2014) with a probability of 0.5 was used during training, on all hidden layers, except the embedding layer.

4.4 Perplexity in Color-Space

The perplexity allows us to evaluate how well our estimated distribution matches the distribution of the observations in the test set. Perplexity is commonly used for evaluating language models. However here it is being used to evaluate the discretized distribution estimate. It can loosely be thought of as to how well the model's distribution does in terms of the size of an equivalent uniform distribution. Note that this metrics does not assume conditional independence of the color channels.

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

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

As this varies depending on the output resolution, we define a standardized perplexity $\frac{PP(\tau)}{n^3}$, where n is the per channel output resolution of the model. The standardised perplexity allows us to compare models of different output resolutions. It is equivalent to comparing the relative performance of the model to that of a uniform distribution $PP_{uniform} = n^3$. Perplexity is a measure of how well the distribution, estimated by the model, matches reality according to the observations in the test set.

4.5 Implementation

The implementation of the CDEST and baseline models was in the Julia programming language (Bezanson et al., 2014). The full implementation is included in the supplementary materials.

5 Results and Discussion

5.1 Qualitative Comparison of the Distribution

Shown in Figures 2 to 4 are side-by-side comparisons of the output of the CDEST and the baseline models. Overall, it can be seen that the baseline model is has a lot more spikes, whereas the CDEST model tends to be much smoother, even though both use the same blurring during discretization. This smoothness is in line with the desired results from the model. As the bin boundaries are artificial and very narrow, it is not reasonable to expect that in reality viewers have such

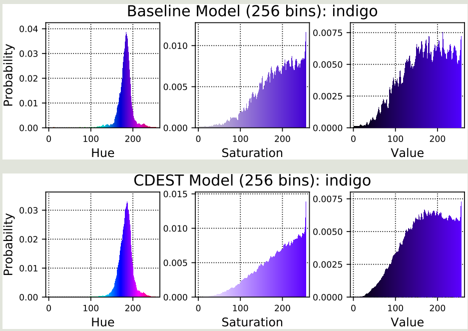


Figure 2: Distribution estimate for `indigo`

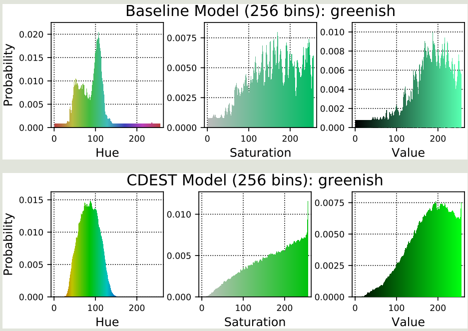


Figure 3: Distribution estimate for `greenish`

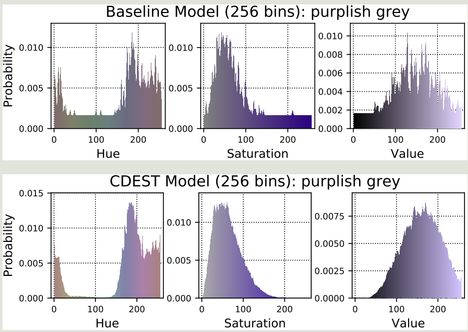


Figure 4: Distribution estimate for `purplish grey`

bands of colors that they think of as more connected to the color name than their neighbours. We expect (as discussed in Section 3.2) continuity, where adjacent points in color-space have similar probability values.

This smoothness can be taken too far, however, when it results in the filling-in of the area between peaks for multimodal colors. It can be seen that the CDEST model fails for some multimodal colors – such as the hue `greenish` (Figure 3 Hue) where the concave section is filled in; but succeeds for others such as `purplish grey` (Figure 4). We attribute this to the particular difficulty of `greenish` which functions very differently as a modifier vs as a standalone color, and suggest future models may benefit from tagging modifiers distinctly from the head-terms during preprocessing.

The horizontal bands in the baseline model outputs are the result of the add-one smoothing process. Notice that they are larger for colors with fewer examples – such as `purplish grey`. In the seminal work of Bengio et al. (2003) one of the motivations for employing neural networks in natural language processing was to better handle cases that do not occur in the training data, by sharing information between terms. CDEST efficiently applies the same core idea here for distribution estimation. The neural model of CDEST can, by knowledge sharing, better estimate the values for the unseen points in color-space, as compared to using smoothing. This is distinct from, but related to, its key capacity as a compositional model to handle unseen cases in the natural language space.

model	n	PP	$\frac{PP}{n^3}$
CDEST	64	20,200	0.077
Baseline	64	20,100	0.077
CDEST	256	1,210,000	**0.072**
Baseline	256	1,330,000	0.079

Table 1: The results of evaluation on the full Munroe dataset. Smaller $\frac{PP}{n^3}$ is better.

5.2 Direct Distribution Estimation

We first test the capacity of the model to estimate the distributions on the standard test dataset, using the standard training dataset. We perform this evaluation before the more difficult (and important) evaluation on the extrapolation task, to confirm that the models are capable of estimating distributions. The results are shown in Table 1. It can be seen that all models perform similarly. The CDEST model based on sequence of color tokens, reflects the real use of the color descriptions in the test set just well as the non-compositional baseline, which counts the exact uses of whole descriptions. This confirms that the CDEST model is able to learn to estimate a color distribution, and that the tokenization and sequential processing did not reduce the mapping ability of the model.

The CDEST model matches baseline performance, when trained on a full set of color terms with all com-

model	n	PP	$\frac{PP}{n^3}$
Extrapolating CDEST	64	20,400	0.078
Non-extrapolating CDEST	64	15,200	0.058
Non-extrapolating Baseline	64	18,100	0.069
Extrapolating CDEST	256	1,290,000	0.077
Non-extrapolating CDEST	256	851,000	0.051
Non-extrapolating Baseline	256	2,140,000	0.128

Table 2: The results of evaluation on the extrapolation sub-dataset. Smaller $\frac{PP}{n^3}$ is better.

binations of terms present in the training data. It seems there is little reason to use the CDEST model in this case, since the baseline model is simpler. However, the key advantage of the CDEST model is its ability to predict a distribution for an unseen combination of colors. This is evaluated using the extrapolation task.

5.3 Extrapolation to Unseen Color Names

A core motivation of using the CDEST model, is its ability to learn to combine tokens in a description in ways not seen in training. This demonstrates that the model is capable of learning the compositional effects of the tokens in the color name. That is to say learning how each token influences the final distribution – rather than simply memorising the training data, as is done in the case of the baseline.

When it comes to the extrapolation task, the best the baseline model can do is an uniform distribution as the color descriptions in the test set do not occur in the training set. This is an uninteresting comparison as it is always $\frac{PP}{n^3} = 1.0$ (and as such is not included in Table 2). Thus we look to comparing the results for extrapolation to the models when they are trained without the need for extrapolation.

We compare a CDEST model trained on the extrapolation sub-dataset, to the models trained on the full dataset. Both the non-extrapolating, and extrapolating models are evaluated on the same test set of rare color descriptions, but the non-extrapolating models are also shown these rare descriptions during training. The non-extrapolating models are expected to perform better given they have direct information on the rare color descriptions' distributions. The extrapolating model must use the knowledge of how those color terms influence the color distribution without direct training.

The results for this evaluation are shown in Table 2. As expected, the non-extrapolating CDEST outperforms the extrapolating CDEST. However, the decrease in performance when forced to extrapolate is relatively small. The extrapolation results are similar to the overall results from Table 1. These are good results, indicative that the model has learnt how the terms interact to define the color distribution. By training on uses of color terms in other descriptions the model learns these useful relationships and encodes them into the networks weights, such that when the terms are used new descriptions, the network can still estimate the distribution. This kind of learning allows knowledge sharing between color descriptions.

The non-extrapolating CDEST also benefits from the same knowledge sharing that enables the extrapolating CDEST model to function. This knowledge sharing allows it to outperform the baseline model, as the relationship between terms provides extra-data to better estimate the shape of the low-data curves. The baseline model does not have such knowledge sharing, thus has difficulties in estimating the curve of these rare descriptions. This is notable in the high resolution case (256 bin), where the sparsity of the training data is high enough to demonstrate the benefits of the know-sharing as shown by the extrapolating CDEST model outperforming the non-extrapolating baseline.

6 Conclusion

We have presented the CDEST model for estimating the probably distribution of colors that may be ascribed to an input name. For each input color name our model outputs a probability distribution over discrete regions of the color-space. Outputting a probability distribution, rather than a single point, allows for better handling of colors with observed distributions that are asymmetric, with high variance or which are multimodal in the color-space – which is the case for most colors.

The CDEST model learns the compositional structure of a color name, which allows it to predict distributions for color names which are not seen during training. As the it learns how each term influences the shape of the distribution, it can thus estimate a distribution for arbitrary compound color names, based on the learnt understanding of the individual terms. This allows it to excel when the sparsity of training data is high.

We find that the discretization process for representing the continuous probability distribution is pragmatically effective, but unsatisfying. While it is possible to simply fit a GMM or other continuous model to the final discretized output; in future work would investigate the extensions of works such as Magdon-Ismail and Atiya (1998); Likas (2001); Ambrogioni et al. (2017)..

References

L. Ambrogioni, U. Güçlü, M. A. J. van Gerven, and E. Maris. 2017. The Kernel Mixture Network: A Nonparametric Method for Conditional Density Estimation of Continuous Random Variables. *ArXiv e-prints*.

Yuval Atzmon, Jonathan Berant, Vahid Kezami, Amir Globerson, and Gal Chechik. 2016. Learning to generalize to new compositions in image understanding. *CoRR*, abs/1608.07639.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, pages 137–186.

Toby Berk, Arie Kaufman, and Lee Brownston. 1982. A human factors study of color notation systems for computer graphics. *Commun. ACM*, 25(8):547–550.

Brent Berlin and Paul Kay. 1969. *Basic color terms: Their university and evolution*. California UP.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2014. Julia: A fresh approach to numerical computing.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Stanley F Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Ronan Collobert 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*, pages 160–167. ACM.

Damian Conway. 1992. An experimental comparison of three natural language colour naming models. In *Proc. east-west int. conf. on human-computer interaction*, pages 328–339.

George E Dahl, Tara N Sainath, and Geoffrey E Hinton. 2013. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE.

Eleanor R Heider. 1972. Universals in color naming and memory. *Journal of experimental psychology*, 93(1):10.

Eleanor Rosch Heider and Donald C. Olivier. 1972. The structure of the color space in naming and memory for two languages. *Cognitive Psychology*, 3(2):337 – 354.

Kazuya Kawakami, Chris Dyer, Bryan R. Routledge, and Noah A. Smith. 2016. Character sequence models for colorfulwords. *CoRR*, abs/1609.08777.

Kenneth Low Kelly et al. 1955. Iscc-nbs method of designating colors and a dictionary of color names.

Reinhard Kneser 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*, volume 1, pages 181–184. IEEE.

Johan Maurice Gis ele Lammens. 1994. *A Computational Model of Color Perception and Color Naming*. Ph.D. thesis, State University of New York.

Aristidis Likas. 2001. Probability density estimation using artificial neural networks. *Computer physics communications*, 135(2):167–175.

Malik Magdon-Ismail and Amir Atiya. 1998. Neural networks for density estimation. In *NIPS*, pages 522–528.

E. Mansimov, E. Parisotto, J. Lei Ba, and R. Salakhutdinov. 2015. Generating Images from Captions with Attention. *ArXiv e-prints*.

Brian McMahan and Matthew Stone. 2015. A bayesian model of grounded color semantics. *Transactions of the Association for Computational Linguistics*, 3:103–115.

Gloria Menegaz, Arnaud Le Troter, Jean Sequeira, and Jean-Marc Boi. 2007. A discrete model for color naming. *EURASIP Journal on Applied Signal Processing*, 2007(1):113–113.

T. Meo, B. McMahan, and M. Stone. 2014. Generating and resolving vague color reference. *Proc. 18th Workshop Semantics and Pragmatics of Dialogue (SemDial)*.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan H Cernocky, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE.

Aleksandra Mojsilovic. 2005. A computational model for color naming and describing color composition of images. *IEEE Transactions on Image Processing*, 14(5):690–699.

W. Monroe, N. D. Goodman, and C. Potts. 2016. Learning to Generate Compositional Color Descriptions. *ArXiv e-prints*.

Will Monroe, Robert X. D. Hawkins, Noah D. Goodman, and Christopher Potts. 2017. Colors in context: A pragmatic neural model for grounded language understanding. *CoRR*, abs/1703.10186.

Randall Munroe. 2010. Xkcd: Color survey results.

Dimitris Mylonas, Matthew Purver, Mehrnoosh Sadrzadeh, Lindsay MacDonald, and Lewis Griffin. 2015. The use of english colour terms in big data. The Color Science Association of Japan.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. 2016. Generative adversarial text to image synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 3.

Ronald Rosenfeld. 2000. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278.

Alvy Ray Smith. 1978. Color gamut transform pairs. *ACM Siggraph Computer Graphics*, 12(3):12–19.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Andrew Stockman, Lindsay T. Sharpe, and Clemens Fach. 1999. The spectral sensitivity of the human short-wavelength sensitive cones derived from thresholds and color matches. *Vision Research*, 39(17):2901 – 2927.

Joost Van De Weijer, Cordelia Schmid, Jakob Verbeek, and Diane Larlus. 2009. Learning color names for real-world applications. *IEEE Transactions on Image Processing*, 18(7):1512–1523.

Learning Distributions of Meant Color – Supplementary Materials

Anonymous ACL submission

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

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

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

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

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

We consider the third quartile of that correlation as the indicative statistic in Table 1. That is to say for 75% of all color names, for the given color-space, the correlation is less than this value.

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

Table 1: The third quartile for the pairwise Spearman's correlation of the color channels given the color name.

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

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

1

Part III

Connecting Classical to Neural Representations

Generating Bags of Words from the Sums of their Word Embeddings

Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun

The University of Western Australia
35 Stirling Highway, Crawley, Western Australia
lyndon.white@research.uwa.edu.au, roberto.togneri@uwa.edu.au,
wei.liu@uwa.edu.au, mohammed.bennamoun@uwa.edu.au

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

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

1 Introduction

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

The motivations for this task are the same as in the related area of sentence generation. Dinu and Baroni (2014) observe that given a sentence has a given meaning, and the vector encodes the same meaning, then it must be possible to translate in both directions between the natural language and the vector representation. A sub-step of this task is the unordered case (BOW), rather than true sentences, which we tackle in this paper. The success of the implementation does indicates the validity of this dual space theory, for the representations considered (where order is neglected). There are also some potential practical applications of such an implementation, often ranging around common vector space representations.

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

Several approaches have been developed for representing images and sentences in a common vector space. This is then used to select a suitable caption from a list of candidates (Farhadi et al. 2010; Socher et al. 2014). Similar methods, creating a common space between images and SOWE of the keywords describing them, could be used to generate keyword descriptions using BOW resynthesis – without any need for a list. This would allows classical word-based information retrieval and indexing techniques to be applied to images.

A similar use is the replacement of vector based extractive summarisation (Kågebäck et al. 2014; Yogatama et al. 2015), with keyword based abstractive summarisation, which is the generation of a keyword summary from a document. The promising use of SOWE generation for all these applications is to have a separate model trained to take the source information (e.g. a picture for image description, or a cluster of sentences for abstract summarisation) as its input and train it to output a vector which is close to a target SOWE vector. This output can then be used to generate the sentence.

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

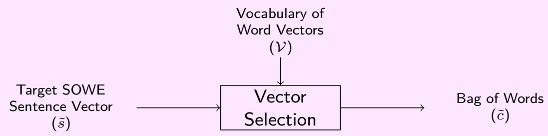


Fig. 1: The process for the regenerating BOW from SOWE sentence embeddings.

The rest of the paper is organized into the following sections. Section 2 introduces the area, discussing in general sentence models, and prior work on generation. Section 3 explains the problem in detail and our algorithm for solving it. Section 4 described the settings used for evaluation. Section 5 discusses the results of this evaluation. The paper presents its conclusions in Section 6, including a discussion of future work.

2 Background

The current state of the art for full sentence generation from sentence embeddings are the works of Iyyer et al. 2014 and Bowman et al. 2015. Both these advance beyond the earlier work of Dinu and Baroni 2014 which is only theorised to extend beyond short phrases. Iyyer et al. and Bowman et al. produce full sentences. These sentences are shown by examples to be loosely similar in meaning and structure to the original sentences. Neither works has produced quantitative evaluations, making it hard to determine between them. However, when applied to the various quantitative examples shown in both works neither is able to consistently reproduce exact matches. This motivates investigation on a simpler unordered task, converting a sum of word embeddings to bag of words, as investigated in this paper.

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

Word embeddings are vector representations of words. They have been shown to encode important syntactic and semantic properties. There are many different types of word embeddings (Yin and Schütze 2015). Two of the more notable are the SkipGrams of Mikolov et al. (2013a,b) and the Global Vector word representations (GloVe) of Pennington et al. (2014). Beyond word representations are sentence embeddings.

Sentence embeddings represent sentences, which are often derived from word embeddings. Like word embeddings they can capture semantic and syntactic features. Sentence vector creation methods include the works of Le and Mikolov (2014) and Socher (2014). Far simpler than those methods, is the sum of word embeddings (SOWE). SOWE, like BOW, draws significant criticism for not only disregarding sentence structure, but disregarding word order entirely when producing the sentence embedding. However, this weaknesses, may be offset by the improved discrimination allowed through words directly affecting the sentence embedding. It avoids the potential information loss through the indirection of more complex methods. Recent results suggest that this may allow it to be comparable overall to the more linguistically consistent embeddings when it comes to representing meaning.

White et al. (2015) found that when classifying real-world sentences into groups of semantically equivalent paraphrases, that using SOWE as the input resulted in very accurate classifications. In that work White et al. partitioned the sentences into groups of paraphrases, then evaluated how well a linear SVM could classify unseen sentences into the class given by its meaning. They used this to evaluate a variety of different sentence embeddings techniques. They found that the classification accuracy when using SOWE as the input performed very similarly to the best performing methods – less than 0.6% worse on the harder task. From this they concluded that the mapping from the space of sentence meaning to the vector space of the SOWE, resulted in sentences with the same meaning going to distinct areas of the vector space.

Ritter et al. (2015) presented a similar task on spacial-positional meaning, which used carefully constructed artificial data, for which the meanings of the words interacted non-simply – thus theoretically favouring the more complex sentence embeddings. In their evaluation the task was classification with a Naïve Bayes classifier into one of five categories of different spatial relationships. The best of the SOWE models they evaluated, outperformed the next best model by over 5%. These results suggest this simple method is still worth consideration for many sentence embedding representation based tasks. SOWE is therefore the basis of the work presented in this paper.

3 The Vector Selection Problem

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

Definition 1. *The Vector Selection Problem is defined on* $(\mathcal{V}, \tilde{s}, d)$ *for a finite vocabulary of vectors* $\mathcal{V}$*,* $\mathcal{V} \subset \mathbb{R}^n$*, a target sentence embedding* $\tilde{s}$*,* $\tilde{s} \in \mathbb{R}^n$*, and any distance metric* d*, by:*

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

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

3.1 NP-Hard Proof

The vector selection problem is NP-Hard. It is possible to reduce from any given instance of a *subset sum problem* to a vector selection problem. The *subset sum problem* is NP-complete (Karp 1972). It is defined: for some set of integers ($\mathcal{S} \subset \mathbb{Z}$), does there exist a subset ($\mathcal{L} \subseteq \mathcal{S}$) which sums to zero ($0 = \sum_{l_i \in \mathcal{L}} l_i$). A suitable metric, target vector and vocabulary of vectors corresponding to the elements $\mathcal{S}$ can be defined by a bijection; such that solving the vector selection problem will give the subset of vectors corresponding to a subset of $\mathcal{S}$ with the smallest sum; which if zero indicates that the subset sum does exists, and if nonzero indicates that no such subset ($\mathcal{L}$) exists. A fully detailed proof of the reduction from subset sum to the vector selection problem can be found on the first author's website. [1]

3.2 Selection Algorithm

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

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

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

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

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

The n-substitution phase has time complexity of $O(\binom{C}{n}V^n)$, for $C = \sum \tilde{c}$ i.e. current cardinality of bag_c. With large vocabularies it is only practical to consider 1-substitution. With the Brown Corpus, where $|\mathcal{V}| \approxeq 40,000$, it was found that 1-substitution provides a significant improvement over greedy addition alone. On a smaller trial corpora, where $|\mathcal{V}| \approxeq 1,000$, 2-substitution was used and found to give further improvement. In general it is possible to initially use 1-substitution,

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

```
Data: the metric d
the target sum s̃
the vocabulary of vectors 𝒱
the current best bag of vectors bag_c: initially ∅
Result: the modified bag_c which sum to be as close as greedy search can get to
        the target s̃, under the metric d
begin
 | t̃ ⟵ Σ_{x_i ∈ bag_c} x_i
 | while true do
 |  | x̃* ⟵ argmin_{x_j ∈ 𝒱} d(s̃, t̃ + x̃_j)      /* exhaustive search of 𝒱      */
 |  | if d(s̃, t̃ + x̃*) < d(s̃, t̃) then
 |  |  | t̃ ⟵ t̃ + x̃*  bag_c ⟵ bag_c ∪ {x̃*}
 |  | else
 |  |  | return bag_c      /* No further improving step found      */
 |  | end
 | end
end
```

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

and if the overall algorithm converges to a poor solution (given the distance to the target is always known), then the selection algorithm can be retried from the converged solution, using 2-substitution and so forth. As n increases the greed overall decreases; at the limit the selection is not greedy at all, but is rather an exhaustive search.

4 Experimental Setup and Evaluations

4.1 Word Embeddings

GloVe representations of words (Pennington et al. 2014) are used in our evaluations. There are many varieties of word embeddings which work with our algorithm. GloVe was chosen simply because of the availability of a large pre-trained vocabulary of vectors. The representations used for evaluation were pretrained on 2014 Wikipedia and Gigaword 5[2]. Preliminary results with SkipGrams from Mikolov et al. (2013a) suggested similar performance.

4.2 Corpora

The evaluation was performed on the Brown Corpus (Francis and Kucera 1979) and on a subset of the Books Corpus (Zhu et al. 2015). The Brown Corpus was sourced with samples from a 500 fictional and non-fictional works from 1961. The

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

Books Corpus was sourced from 11,038 unpublished novels. The Books Corpus is extremely large, containing roughly 74 million sentences. After preprocessing we randomly selected 0.1% of these for evaluation.

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

After the preprocessing the final corpora can be described as follows. The Brown Corpus has 42,004 sentences and a vocabulary of 40,485 words. Where-as, the Books Corpus has 66,464 sentences, and a vocabulary of 178,694 words. The vocabulary sizes are beyond what is suggested as necessary for most uses (Nation 2006). These corpora remain sufficiently large and complex to quantitatively evaluate the algorithm.

4.3 Vector Selection

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

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

The Julia programming language (Bezanson et al. 2014), was used to create the implementation of the method, and the evaluation scripts for the results presented in the next section. This implementation, evaluation scripts, and the raw results are available online.[4]. Evaluation was carried out in parallel on a 12 core virtual machine, with 45Gb of RAM. Sufficient RAM is required to load the entire vector vocabulary in memory.

5 Results and Discussion

Table 1 shows examples of the output. Eight sentences which were used for demonstration of sentence generation in Bowman et al. (2015) and Iyyer et al.

[3] The same is true for any number of repetitions of the word *buffalo* – each of which forms a valid sentence as noted in Tymoczko et al. (1995)

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

Table 1: Examples of the BOW Produced by our method using the Books Corpus vocabulary, compared to the Correct BOW from the reference sentences. The P and C columns show the the number of occurrences of each word in the Produced and Correct bags of words, respectively. **Bolded** lines highlight mistakes. Examples a-e were sourced from Iyyer et al. (2014), Examples f-h from Bowman et al. (2015). Note that in example a, the *"___…_(n)"* represents n repeated underscores (without spaces).

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

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

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

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

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

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

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

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

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

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

(f) how are you doing ?

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

(g) we looked out at the setting sun .

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

(h) i went to the kitchen .

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

Table 2: The performance of the BOW generation method. Note the final line is for the Books Corpus, where-as the preceding are or the Brown Corpus.

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

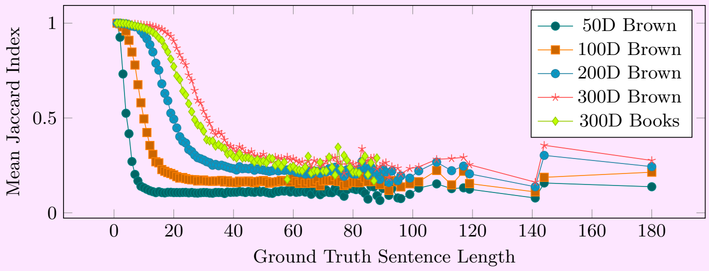


Fig. 2: The mean Jaccard index achieved during the word selection step, shown against the ground truth length of the sentence. Note that the vast majority of sentences are in the far left end of the plot. The diminishing samples are also the cause of the roughness, as the sentence length increases.

(2014) have the BOW generation results shown. All examples except *(a)* and *(f)* are perfect. Example *(f)* is interesting as it seems that the contraction token *'re* was substituted for *are*, and *do* for *doing*. Inspections of the execution logs for running on the examples show that this was a greedy mistake that would be corrected using 2-substitution. Example *a* has many more mistakes.

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

Table 2 shows the quantitative performance of our method across both corpora. Five measures are reported. The most clear is the portion of exact matches – this is how often out of all the trials the method produced the exact correct bag of words. The remaining measures are all means across all the values of the measures in each trial. The Jaccard index is the portion of overlap between the reference BOW, and the output BOW – it is the cardinality of the intersection divided by that of the union. The precision is the portion of the output words that were correct; and the recall is the portion of all correct words which were output. For precision and recall word repetitions were treated as distinct. The F_1 score is the harmonic mean of precision and recall. The recall is higher than the precision, indicating that the method is more prone to producing additional incorrect words (lowering the precision), than to missing words out (which would lower the recall).

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

As can be seen in Figure 2 sentence length is a very significant factor in the performance of our method. As the sentences increase in length, the number of mistakes increases. However, at higher embedding dimensionality the accuracy for most sentences is high. This is because most sentences are short. The third quartile on sentence length is 25 words for Brown, and 17 for the Books Corpus. This distribution difference is also responsible for the apparent better results on the Books Corpus, than on the Brown corpus.

While the results shown in Table 2 suggest that on the Books corpus the algorithm performs better, this is due to its much shorter average sentence length. When taken as a function of the sentence length, as shown in Figure 2, performance on the Books Corpus is worse than on the Brown Corpus. It can be concluded from this observation that increasing the size of the vocabulary does decrease success in BOW regeneration. Books Corpus vocabulary being over four times larger, while the other factors remained the same, resulted in lower performance. However, when taking all three factors into account, we note that increasing the *vocabulary size* has significantly less impact than increasing the *sentence length* or the *embedding dimensionality* on the performance.

6 Conclusion

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

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

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

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

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

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

References

Bezanson, Jeff et al. (2014). “Julia: A Fresh Approach to Numerical Computing”. In: arXiv: `1411.1607 [cs.MS]`.

Bowman, Samuel R et al. (2015). “Generating Sentences from a Continuous Space”. In: *arXiv preprint arXiv:1511.06349*.

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

Farhadi, Ali et al. (2010). “Every picture tells a story: Generating sentences from images”. In: *Computer Vision–ECCV 2010*. Springer, pp. 15–29.

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

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

Kågebäck, Mikael et al. (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.

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

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.

Mikolov, Tomas et al. (2013a). “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781*.

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

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

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.

Ritter, Samuel et al. (2015). “Leveraging Preposition Ambiguity to Assess Compositional Distributional Models of Semantics”. In: *The Fourth Joint Conference on Lexical and Computational Semantics*.

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

Socher, Richard et al. (2014). “Grounded compositional semantics for finding and describing images with sentences”. In: *Transactions of the Association for Computational Linguistics* 2, pp. 207–218.

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

White, Lyndon et al. (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.

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

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

Zhu, Yukun et al. (2015). “Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books”. In: *arXiv preprint arXiv:1506.06724*.

THE VECTOR SELECTION PROBLEM

LYNDON WHITE, WEI LIU

Definition 1. the Vector Selection problem is defined on $(\mathcal{V},\, \tilde{s},\, d)$ for

- A finite vocabulary of vectors $\mathcal{V}$, $\mathcal{V} \subset \mathbb{R}^n$
- a target vector $\tilde{s}$, $\tilde{s} \in \mathbb{R}^n$
- any metric d

  by

$$\underset{\{\forall \tilde{c} \in \mathbb{N}_0^V\}}{\operatorname{argmin}} \; d(\tilde{s},\, \sum_{j=1}^{j=V} \tilde{x}_j c_j)$$

- V is size of the vocabulary $\mathcal{V}$. (~1300 for ATIS2, ~50,000 for Brown, ~10,000 for daily English)
- $\tilde{x}_j$ is the vector embedding for the jth word in the vocabulary $\tilde{x}_j \in \mathcal{V}$
  - We can express the embedding vocabulary $\mathcal{V} = \{\tilde{x}_j \,|\, \forall j \in \mathbb{N} \,\wedge\, 1 \leq j \leq V\} \subset \mathbb{R}^n$
  - If we treat $\mathcal{V}$ as a matrix with vectors $\tilde{x}_j$ for rows,(and treat length 1 vectors as scalars) we get the compact notation

$$\underset{\{\forall \tilde{c} \in \mathbb{N}_0^J\}}{\operatorname{argmin}} \; d(\tilde{s},\, \sum_{j=1}^{j=V} \tilde{x}_j c_j) = \underset{\{\forall \tilde{c} \in \mathbb{N}_0^J\}}{\operatorname{argmin}} \; d(\tilde{s},\, \mathcal{V}\,\tilde{c}^T)$$

- c_j is the count of how many times the jth word in the vocabulary occurs. $\tilde{c} \in \mathbb{N}_0^V$, so $c_j \in \mathbb{N}_0$
- n is the dimensional of the word vectors, $n = 300$ in current trials.

0.1. **An Analogy for the problem.** (which may or may not help)

Imaging you are in a tile shop which as a variety of rectangular (2D) tiles. They have many copies of each tile (an unlimited number in-fact), but only a finite number of different sizes. This tiles have connectors on them, like jigsaw pieces, such that you can attach a North/South side to another North/South side even on a tile of different size, and similar for the East/West sides. But you can't attach a North/South side to a East/West side. i.e. You can not rotated the tiles.

You have 2 lengths given as your target when choosing tiles: a North/South length, and a East/West length.

Your task is to select a collection of tiles from the store, such that when connected on the north/south and east/west the total length in those directions is as close as possible to those to targets.

A formula is given for how your solution will be judged. It takes the form of some distance metric. E.g it might be the your distance from the target east/west length with your connected tiles, plus your distance from the target north/south length. Or maybe accuracy on north/south is twice as important as east/west. Or north/south difference squared etc. Your task it to minimize that score.

For example, if the store had 3 types of tiles. 4.1×4.1,1.5×5.0 and 100×1, and your targets were 13.1 and 20.0, and the scoring was Manhattan.

you might choose one 4.1×4.1 tiles and three 1.5×5.0 tiles, giving you length totals 8.6 and 19.1 and a score of 5.4

Had you chosen to take an extra 4.1×4.1 tile though given totals of 12.7 and 23.2 giving a better score of 4.5

Now generalize it from 2D tiles to hyperblocks of some arbitrary dimensionality.

0.2. **Reduction from Subset sum.** The subset sum problem is well known to be NP-complete. First shown in by Karp under the name "Knapsack"[1] which has since come to be used for the more general problem.

It can be defined with the question: for a given set $\mathcal{S} \subset \mathbb{Z}$, does there exists $\mathcal{L} \subseteq \mathcal{S}$ such that $\sum_{l_i \in \mathcal{L}} \; l_i = 0$?

We reduce from subset sum to the Vector Selection Problem by showing any general solution to the Vector Selection Problem could be used to solve subset sum with only linear time additional work.

Claim 2. Any method which can solve the Vector Selection Problem will allow Subset sum to be completed with only linear time additional operations

- Let $\mathcal{S} = \{w_1, w_2, ..., w_m\}$
- Let $\Omega = 2m \left(\max_{i \in [1,m]} \, |w_i| + 1\right)$ and thus larger than the largest possible sum of elements of $\mathcal{S}$.

1

- Finding this is a linear time operation, the only such operation in this method.
- Let $\omega = \frac{1}{2m}$ and thus smaller than any element of $\mathcal{S}$, except if $0 \in \mathcal{S}$ (in which case the solution is trivial)
- then we can define an embedding vocabulary $\mathcal{V}_s$ from based on $\mathcal{S}$ by

$$\mathcal{V}_s = \{[[w_i,\, 1]\,;\, \hat{e}_i] \,:\, w_i \in \mathcal{S}\}$$

  - By imposing some arbitrary total ordering on $\mathcal{S}$.
  - where ; is the concatenation operator,
  - and $\hat{e}_i$ is the elementary basis unit vector for dimension i. ie a vector with all zeros, except at index i, where it is 1.
  - i.e. we take the image of $\mathcal{S}$ into $\mathcal{V}_S$, by the function:

$$w_i \mapsto \begin{bmatrix} w_i \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \hat{e}_{i+2}$$

  - In doing so we map each integer w_i in $\mathcal{S}$ to a point in $\mathbb{R}^{m+1}$ where
    * the first index is the integer, w_i,
    * the second a term is used to force a solution that is nonempty to be better than an empty solution – all other things being equal;
    * the remaining m terms are used to force a solution which uses the same element more than once to be worse than one which uses it once or zero times.
  - Note that $\mathcal{V}_S \subset \mathbb{R}^{m+2}$
- we define the target vector by $\tilde{s_s} = \left[[0,m]\,;\, 0.5\sum_{j=1}^{j=m}\ \hat{e}_j\right] = \begin{pmatrix} 0 \\ m \\ 0.5 \\ \vdots \\ 0.5 \end{pmatrix}$
- we define the distance metric being given by a weighed Manhattan distance (i.e. weighted L1 Norm).

$$d_s\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}\right) = |x_1 - y_1| + \omega\,|x_2 - y_2| + \Omega \sum_{j=3}^{j=n} |x_j - y_j|$$

  - We will prove that d_s is a metric below.
- The procedure for using these once defined to solve subset sum is:
  - the Vector Selection Problem for $(\mathcal{V}_s, \tilde{s}_s, d_s)$ is solved getting back $\tilde{c}^\star$
  - if $\tilde{c}^\star = \mathbf{0}$, or $\sum_{j=1}^{j=m}\ \tilde{x}_{j,1} c^\star_j \neq 0$ then no such solution exists, otherwise:
  - such a subset $\mathcal{L} \subset \mathcal{S}$ does exist, and is given by $\mathcal{L} = \{w_i \in \mathcal{S} \,:\, c^\star_i \geq 1\}$.
  - Note that it does not matter if $c^\star_i > 1$ as for such cases clipping it to multiplicity 1 is just as optimal. Which we will prove below.

Proof. d_s is a metric
The d_s is a special case of

$$d\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}\right) = \sum_{1 \leq j \leq n} \omega_i\, d'\left(x_j, y_j\right)$$

for d' a metric defined on scalars and $\forall i$ where $\omega_i, x_i, w_i \in \mathbb{R}$ and $\omega_i > 0$

Below it is shown that that this is always a metric, and thus d_s is a metric, by showing it meets the 3 requirements: of the coincidence axiom, being symmetric, and of the triangle inequality. The following properties hold for all $\tilde{a}, \tilde{b}, \tilde{c} \in \mathbb{R}^n$.

If d follows the coincidence axiom: $d(x,y) = 0 \iff x = y$. This is shown by:
if $\tilde{a} = \tilde{b}$ then $\forall j \in [1,n]\ d'(a_j, b_j) = 0$ thus $d(\tilde{a}, \tilde{b}) = 0$.

and if $d(\tilde{a},\tilde{b}) = 0$ then as all $\forall j \in [1,n]\ w_j > 0$, therefore $d'(a_j,b_j) = 0$.

If d is symmetric then $d(x,y) = d(y,x)$. Shown by:

$$d(\tilde{a},\tilde{b}) = \sum_{j=1}^{j=n} \omega_i\, d'\,(a_j,b_j) = \sum_{j=1}^{j=n} \omega_i\, d'\,(b_j,a_j) = d(\tilde{b},\tilde{a})$$

Thus d is symmetric

if d follows the triangle inequality then $d(x,z) \leq d(x,y) + d(y,z)$.
making use of $d'\,(a_j,b_j) +\ d'\,(b_j,c_j) \geq d'(a_j,c_j)$, It is shown:

$$\begin{aligned}
d(\tilde{a},\tilde{b}) + d(\tilde{b},\tilde{c}) &= \sum_{1\leq j\leq n} \omega_i\, d'\,(a_j,b_j) + \sum_{1\leq j\leq n} \omega_i\, d'\,(b_j,c_j)\\
d(\tilde{a},\tilde{b}) + d(\tilde{b},\tilde{c}) &= \sum_{1\leq j\leq n} \omega_i\,(\,d'\,(a_j,b_j) +\ d'\,(b_j,c_j))\\
d(\tilde{a},\tilde{b}) + d(\tilde{b},\tilde{c}) &\leq \sum_{1\leq j\leq n} \omega_i\,(d'(a_j,c_j))\\
d(\tilde{a},\tilde{b}) + d(\tilde{b},\tilde{c}) &\leq d(\tilde{a},\tilde{b})
\end{aligned}$$

Thus d is a metric, and so d_s is a metric. □

Proof. Show for $c_j \notin \{0,1\}$ a better or at least equally good solution can be found for $c_j^{alt} \in \{0,1\}$

For a proof by contraction, we assume the existence of some optimal solution the Vector Selection Problem, c^* where for at least one index i,$c_i^* \geq 2$.

Consider also some alternative count vector (which by our assumption, can not be more optimal)

$$c' = c^* - \hat{e}_i$$

That is, c' is the same as c^*except that there is one less count for c_i^*

recalling $\tilde{s}_s = \begin{pmatrix} 0 \\ m \\ 0.5 \\ \vdots \\ 0.5 \end{pmatrix}$

we define the optimal sum of vectors by $\tilde{t}^*$

$$\tilde{t}^* = \sum_{j=1}^{j=V} \tilde{x}_j c_j^* = \begin{bmatrix} \left(\sum_{j=1}^{j=m} w_j c_j^*\right) \\ \left(\sum_{j=1}^{j=m} w_j c_j^*\right) \\ c_1^* \\ \vdots \\ c_n^* \end{bmatrix}$$

we define the alternative sum of vectors by $\tilde{t}'$

$$\tilde{t}' = \sum_{j=1}^{j=V} \tilde{x}_j c'_j = \left(\sum_{j=1}^{j=V} \tilde{x}_j c^*_j\right) - \tilde{x}_i = \begin{bmatrix} \left(\sum_{j=1}^{j=m} w_j c^*_j\right) - w_i \\ \left(\sum_{j=1}^{j=m} w_j c^*_j\right) - 1 \\ c^*_1 \\ \vdots \\ c^*_{i-1} \\ c^*_i - 1 \\ c^*_{i+1} \\ \vdots \\ c^*_n \end{bmatrix}$$

Note: we know that $c^*_i > 0.5$ as $c^*_i \geq 2$. Similarly we know $c'_i \geq 0.5$ for the as $c'_i = c^*_i - 1$

$$d_s(\tilde{s}_s, \tilde{t}^*) = \begin{array}{cc} & \left(\sum_{j=1}^{j=m} w_j c^*_j\right) \\ + & \omega \left|\left(\sum_{j=1}^{j=m} c^*_j\right) - m\right| \\ + & \Omega \left|c^*_1 - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_{i-1} - 0.5\right| \\ + & \Omega (c^*_i - 0.5) \\ + & \Omega \left|c^*_{i+1} - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_n - 0.5\right| \end{array}$$

and

$$d_s(\tilde{s}_s, \tilde{t}') = \begin{array}{cc} & \left(\sum_{j=1}^{j=m} w_j c^*_j\right) - w_i \\ + & \omega \left|\left(\sum_{j=1}^{j=m} c^*_j\right) - 1 - m\right| \\ + & \Omega \left|c^*_1 - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_{i-1} - 0.5\right| \\ + & \Omega (c^*_i - 0.5 - 1) \\ + & \Omega \left|c^*_{i+1} - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_n - 0.5\right| \end{array}$$

Since $\tilde{t}^*$ from the more optimal solution: $d_s(\tilde{s}_s, \tilde{t}') - d_s(\tilde{s}_s, \tilde{t}^*) \geq 0$

$$\begin{array}{cc} & \left(\sum_{j=1}^{j=m} w_j c^*_j\right) - w_i \\ + & \omega \left|\left(\sum_{j=1}^{j=m} c^*_j\right) - 1 - m\right| \\ + & \Omega \left|c^*_1 - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_{i-1} - 0.5\right| \\ + & \Omega (c^*_{i+1} - 0.5 - 1) \\ + & \Omega \left|c^*_{i+1} - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_n - 0.5\right| \end{array} - \begin{array}{cc} & \left(\sum_{j=1}^{j=m} w_j c^*_j\right) \\ + & \omega \left|\left(\sum_{j=1}^{j=m} c^*_j\right) - m\right| \\ + & \Omega \left|c^*_1 - 0.5\right| \\ & \vdots \\ + & \Omega \left|c^*_{i-1} - 0.5\right| \\ + & \Omega (c^*_{i+1} - 0.5) \\ + & \Omega \left|c^*_{i+1} - 0.5\right| \\ + & \vdots \\ + & \Omega \left|c^*_n - 0.5\right| \end{array} \geq 0$$

And after canceling terms:

$$-w_i + \omega \left(\left| \left(\sum_{j=1}^{j=m} c^*_j\right) - 1 - m \right| - \left| \left(\sum_{j=1}^{j=m} c^*_j\right) - m \right| \right) - \Omega \geq 0$$

let $K = \left(\sum_{j=1}^{j=m} c_j^*\right) - m$

$$-w_i + \omega\left(|K-1| - |K|\right) - \Omega \geq 0$$

The largest value $|K-1| - |K|$ can take is 1. (The other cases are 0, and -1, both of which result in the contradiction of the sum of 2 and 3 negative values respectively being greater than or equal to zero)

$$-w_i + \omega - \Omega \geq -w_i + \omega\left(|K-1| - |K|\right) - \Omega \geq 0$$

$$w_i + \Omega \leq \omega$$

Substituting in the values from the definitions:
$\Omega = 2m\left(\max_{j\in[1,m]} |w_j| + 1\right)$ and $\omega = \frac{1}{2m}$

$$w_i + 2m\left(\max_{j\in[1,m]} |w_j|\right) + 2m \leq \frac{1}{2m}$$

$$2mw_i + 4m^2\left(\max_{j\in[1,m]} |w_j|\right) + 4m^2 \leq 1$$

Assume w_i takes the most negative value possible: $w_i = -\left(\max_{j\in[1,m]} |w_j|\right)$ giving:

$$\left(4m^2 - 2m\right)\left(\max_{j\in[1,m]} |w_j|\right) + 4m^2 \leq 2mw_i + 4m^2\left(\max_{j\in[1,m]} |w_j|\right) + 4m^2 \leq 1$$

As $m \geq 1$, consider it taking that the smallest value it can take (so $m = 1$)
$2\left(\max_{j\in[1,m]} |w_j|\right) + 4 \leq \left(4m^2 - 2m\right)\left(\max_{j\in[1,m]} |w_j|\right) + 4m^2 \leq 1$
requiring, $\max_{j\in[1,m]} |w_j| \leq -\dfrac{3}{2}$
Which is impossible as the absolute value of an integer is always non-negative.
Thus a contradiction.
Thus c' is at least as optimal as c^*.
We may apply this proof to all claimed optimal solutions with a $c_i > 1$ to show that an equally optimal (or more so), solution has that c_i at 1 lower.
Thus if some solution with any count $c_i > 1$ is found, it can be transformed into a solution that is equally (or more so) optimal, by clipping all counts c_i at one. □

A finer proof could be developed showing strict inequality and that c' yields a strictly better solution that c^*

Proof. Proof of Correctness
let $\tilde{c}'$ be the solution to the Vector Selection Problem on $(\mathcal{V}_s, \tilde{s}_s, d_s)$
As it was shown above that for any $c_i' > 1$ an equally optimal solution can be created by clipping $c_i'$to 1.
We will thus assume $c_i' \in \{0,1\}$.
let $L' = \{w_i \in \mathcal{S} \,:\, c_i' = 1\}$

Case 1. Subset Sum Exists, but the Vector Selection Problem based method says it does not
We assume for a proof by contradiction that the the Vector Selection Problem based method states that no such subset sub exists,
however it is incorrect and such a subset does and is given by $L^* \subseteq \mathcal{S}$.
Then $\mathcal{L}^*$defines a indicator vector $c^* \in \{0,1\}^m$, given by $c_j^* = \begin{cases} 1 & w_j \in \mathcal{L}^* \\ 0 & w_j \notin \mathcal{L}^* \end{cases}$, where w_j is the jth element of $\mathcal{S}$.
so $\sum_{j=1}^{j=m} w_j c_j^* = 0$.
Note also as $\mathcal{L}^* \neq \emptyset$ (by definition of subset sum) $\exists i \in [1,m]$ such that $c_i^* = 1$.
we define $\tilde{t}^*$ to be the sum of the vectors which correspond to c_j^* by

$$\tilde{t}^* = \sum_{j=1}^{j=V} \tilde{x}_j c_j^* = \begin{bmatrix} \sum_{j=1}^{j=m} w_j c_j^* \\ \sum_{j=1}^{j=m} c_j^* \\ c_1^* \\ \vdots \\ c_m^* \end{bmatrix} = \begin{bmatrix} 0 \\ \sum_{j=1}^{j=m} c_j^* \\ c_1^* \\ \vdots \\ c_m^* \end{bmatrix}$$

and so

$$d_s(\tilde{s},\, \tilde{t}^*) = d_s\left(\begin{bmatrix} 0 \\ m \\ 0.5 \\ \vdots \\ 0.5 \\ \vdots \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0 \\ \sum_{j=1}^{j=m} c_j^* \\ c_1^* \\ \vdots \\ c_m^* \end{bmatrix} \right) = \begin{matrix} & 0 \\ + & \omega \left| \left(\sum_{j=1}^{j=m} c_j^* \right) - m \right| \\ + & \Omega \left| c_1^* - 0.5 \right| \\ & \vdots \\ + & \Omega \left| c_n^* - 0.5 \right| \end{matrix} = \omega \left| \left(\sum_{j=1}^{j=m} c_j^* \right) - m \right| + 0.5 m \Omega$$

since $0 < \sum_{j=1}^{j=m} c_j^* \leq m$ as it is sum of m variables $0 \leq c_j^* \leq 1$ and not all $c_j^* = 0$, we can that to simplify to

$$d_s(\tilde{s},\, \tilde{t}^*) = \omega \left(m - \sum_{j=1}^{j=m} c_j^* \right) + 0.5 m \Omega$$

Now then consider the cases when the method (incorrectly) reports no such subset exists:

Case i. $\tilde{c}' = [0, ..., 0]$ (the zero vector)

We define the total sum of vectors given by $\tilde{t}'$

$$\tilde{t}' = \sum_{j=1}^{j=V} \tilde{x}_j c_j' = \begin{bmatrix} \sum_{j=1}^{j=m} w_j c_j' \\ \sum_{j=1}^{j=m} c_j' \\ c_1' \\ \vdots \\ c_m' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

So find

$$d_s(\tilde{s},\, \tilde{t}') = Multidimentional d_s \left(\begin{bmatrix} 0 \\ m \\ 0.5 \\ \vdots \\ 0.5 \\ \vdots \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right) = \omega m + 0.5 m \Omega$$

thus $d_s(\tilde{s},\, \tilde{t}^*) < d_s(\tilde{s},\, \tilde{t}')$ and so $c' = [0, ..., 0]$ could not have been the solution returned for solving the Vector Selection Problem as it is not the correct selection for the argmax. Thus a contradiction.

Case ii. $\tilde{c}' \neq \mathbf{0}$ thus $\sum_{j=1}^{j=m} \tilde{x}_{j,1} c_j' = k$ for $k \neq 0$

So thus the method reports that there is no nonempty subset which sums to zero; (the closest it can get is summing to k.

We redefine $\tilde{t}'$ for this case to be

$$\tilde{t}' = \sum_{j=1}^{j=V} \tilde{x}_j c_j' = \begin{bmatrix} \sum_{j=1}^{j=m} w_j c_j' \\ \sum_{j=1}^{j=m} c_j' \\ c_1' \\ \vdots \\ c_m' \end{bmatrix} = \begin{bmatrix} k \\ \sum_{j=1}^{j=m} c_j' \\ c_1' \\ \vdots \\ c_m' \end{bmatrix}$$

and so

THE VECTOR SELECTION PROBLEM 7

$$d_s(\tilde{s},\,\tilde{t}') = d_s(\begin{bmatrix} 0 \\ m \\ 0.5 \\ \vdots \\ 0.5 \\ \vdots \\ 0.5 \end{bmatrix}, \begin{bmatrix} k \\ \sum_{j=1}^{j=m} c_j' \\ c_1' \\ \vdots \\ c_m' \end{bmatrix}) = \begin{matrix} & |k| \\ + & \omega\left|\left(\sum_{j=1}^{j=m} c_j'\right) - m\right| \\ + & \Omega\left|c_1' - 0.5\right| \\ & \vdots \\ + & \Omega\left|c_n' - 0.5\right| \end{matrix} = |k| + \omega\left|\left(\sum_{j=1}^{j=m} c_j'\right) - m\right| + 0.5m\Omega$$

As $c'$was selected over c^* then
by our
recalling:

$$d_s(\tilde{s},\,\tilde{t}^*) = \omega\left|\left(\sum_{j=1}^{j=m} c_j^*\right) - m\right| + 0.5m\Omega$$

as $\tilde{t}'$ is the sum of vectors giving min value for the distance to the target point $\tilde{s_s}$ thus

$$d_s(\tilde{s_s},\,\tilde{t}') \leq d_s(\tilde{s_s},\,\tilde{t}^*)$$

i.e.

$$|k| + \omega\left|\left(\sum_{j=1}^{j=m} c_j'\right) - m\right| + 0.5m\Omega \leq \omega\left|\left(\sum_{j=1}^{j=m} c_j^*\right) - m\right| + 0.5m\Omega$$

let $C' = \left(\sum_{j=1}^{j=m} c_j'\right)$ and $C^* = \left(\sum_{j=1}^{j=m} c_j^*\right)$
as $0 \leq C' \leq m$ and $0 \leq C^* \leq m$ as both are sums of indicator variables (0,1)
$\left|\left(\sum_{j=1}^{j=m} c_j^*\right) - m\right| = |C^* - m| = m - C^*$ and similarly for C'
substituting in:
$|k| + \omega\,(m - C') + 0.5m\Omega \leq \omega\,(m - C^*) + 0.5m\Omega$
i.e $|k| + \omega C' \leq \omega C^*$
i.e $|k| \leq \omega\,(C^* - C')$
thus $(C^* > C')$ it can not be equal as otherwise $k = 0$ which would be a contradiction.
let $(C^* - C') = C_d$, $C_d \in \mathbb{N}$
$0 < C_d$ as otherwise $|k| = 0$ (which would be the contradiction)
$C_d \leq m$ as the largest case is $C^* = m$ and $C' = 0$
Substitute

$$|k| \leq \omega C_d$$

substitute from the definition of $\omega = \frac{1}{2m}$

$$|k| \leq \frac{1}{2m} C_d$$

consider the largest value C_d can take: $C_d = m$

$$|k| \leq \frac{m}{2m}$$

$$|k| \leq \frac{1}{2}$$

As k is an integer this would mean $k = 0$
But this is a contradiction as $|k| > 0$.
Therefore it is not possible for the the Vector Selection Problem based method to say there is no solution if there is a solution.
i.e. if a solution exists, the the Vector Selection Problem based method will find it.

Case 2. Case: Subset sum does not exists, but the Vector Selection Problem based method says it does

Then:

$$\sum_{j=1}^{j=m} \tilde{x}_{j,1} c'_j = 0$$

.

We know that $\tilde{c}' \neq \mathbf{0}$ as other wise the the Vector Selection Problem based method would have said no solution exists.

Thus $\mathcal{L}' \neq \emptyset$

further we know by definition of $\tilde{x_j}$ that $\tilde{x}_{j,1} = w_j$ for $w_j \in \mathcal{S}$

thus we have $\sum_{j=1}^{j=m} w_j c'_j = 0$

thus in fact the sum of the elements of $\mathcal{L}'$ is zero.

And so a subset sum does exist.

This is a contradiction, thus the the Vector Selection Problem based method will never say there is a solution unless one exists.

Thus the method described in Claim 2 is a correct method to solve subset sum.

□

0.2.1. *Subset Sum Reduction Concluding note:* Thus it has been shown that if a general solution to the vector selection problem can be found a solution to subset sum could be found which would take at most a linear amount of additional time. Thus were a polynomial time solution for the Vector Selection Problem found, it would show that $P = NP$. However, the proof above is only for the general case, which is defined over $(\mathcal{V},\ \tilde{s},\ d)$ for finite subsets of $\mathbb{R}^n$, $\mathcal{V}$; and any$\tilde{s} \in \mathbb{R}^n$, using any metric d. Thus the hardness result is only for the general case. Like for many problems from the knapsack family, there certainly exists special cases for which faster solutions are possible. For example $\mathcal{V} \subset \mathbb{R}^1_+$, $\tilde{s} = [0]$ and $d = (x, y) \mapsto |x - y|$, a linear time solution exists, found by finding the index of the smallest member of $\mathcal{V}$. The general problem however is not expected to have an exact solution in polynomial time.

References

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

1

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

Lyndon White, Roberto Togneri, Wei Liu Mohammed Bennamoun
The University of Western Australia
35 Stirling Highway, Crawley, Western Australia
lyndon.white@research.uwa.edu.au
{roberto.togneri, wei.liu, mohammed.bennamoun}@uwa.edu.au

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

1 Introduction

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

Dinu and Baroni [1] motivates this work from a theoretical perspective given that a sentence encodes its meaning, and the vector encodes the same meaning, then it must be possible to translate in both directions between the natural language and the vector representation. In this paper, we present an implementation that indicates to some extent the equivalence between the natural language space and the sum of word embeddings (SOWE) vector representation space. This equivalence is shown by demonstrating a lower bound on the capacity of the vector representation to be used for sentence generation.

The current state of the art methods for sentence generation produce human readable sentences which are rough approximations of the intended sentence. These existing works are those of Iyyer, Boyd-Graber, and Daumé III [2] and Bowman, Vilnis, Vinyals, *et al.* [3]. Both these have been demonstrated to produce full sentences. These sentences are qualitatively shown to be loosely similar in meaning to the original sentences. Neither work has produced quantitative evaluations, making it hard to compare their performance. Both are detailed further in Section 2. Both these methods use encoder/decoder models trained through machine learning; we present here a more deterministic algorithmic approach, but restrict the input sentence vector to be the non-compositional sum of word embeddings representation.

Ritter, Long, Paperno, *et al.* [4] and White, Togneri, Liu, *et al.* [5] found that when classifying sentences into categories according to meaning, simple SOWE outperformed more complex sentence vector models. Both works used sentence embeddings as the input to classifiers. Ritter, Long, Paperno, *et al.* [4] classified challenging artificial sentences into categories based on the positional relationship described using Naïve Bayes. White, Togneri, Liu, *et al.* [5] classified real-world sentences into groups of semantically equivalent paraphrases. In the case of Ritter, Long, Paperno, *et al.* [4] this outperformed the next best representation by over 5%. In the case of White, Togneri, Liu, *et al.* [5] it was within a margin of 1% from the very best performing method. These results suggest that there is high consistency in the relationship between a point in the SOWE space, and the meaning of the sentence.

Wieting, Bansal, Gimpel, *et al.* [6] presented a sentence embedding based on the related average of word-embedding, showing excellent performance across several competitive tasks. They compared their method's performance against several models, including recurrent neural networks, and long short term memory (LSTM) architectures. It was found that their averaging method outperformed the more complex LSTM system, on most

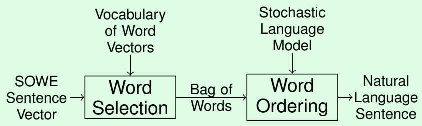


Figure 1. The Sel. BOW+Ord. process for the regenerating sentences from SOWE-type sentence vectors.

sentence similarity and entailment task. Thus these simple methods are worth further consideration. SOWE is the basis of the work presented in this paper.

Our method performs the sentence generation in two steps, as shown in Figure 1. It combines the work of White, Togneri, Liu, *et al.* [7] on generating bags of words (BOW) from sums of word embeddings (SOWE); with the work of Horvat and Byrne [8] on ordering BOW into sentences. The overall approach, of word selection followed by word ordering, can be used to generate proper sentences from SOWE vectors.

The rest of the paper is organized into the following sections. Section 2 discusses the prior work on sentence generation. Section 3 explains the problem in detail and how our method is used to solve it. Section 4 describes the settings used for evaluation. Section 5 presents the results of this evaluation. The paper concludes with Section 6 and a discussion of future work on this problem.

2 Related Works

To the best of our knowledge only three prior works exist in the area of sentence generation from embeddings. The first two (Dinu and Baroni [1], Iyyer, Boyd-Graber, and Daumé III [2]) are based on the recursive structures in language, while Bowman, Vilnis, Vinyals, *et al.* [3], uses the sequential structure.

Dinu and Baroni [1] extends the models described by Zanzotto, Korkontzelos, Fallucchi, *et al.* [9] and Guevara [10] for generation. The composition is described as a linear transformation of the input word embeddings to get an output vector, and another linear transformation to reverse the composition reconstructing the input. The linear transformation matrices are solved using least squares regression. This method of composing, can be applied recursively from words to phrases to clauses and so forth. It theoretically generalises to whole sentences, by recursively applying the composition or decomposition functions. However, Dinu and Baroni's work is quantitatively assessed only on direct reconstruction for decomposing Preposition-Noun and Adjective-Noun word phrases. In these cases where the decomposition function was trained directly on vectors generated using the dual composition function they were able to get perfect reconstruction on the word embedding based inputs.

Iyyer, Boyd-Graber, and Daumé III [2] extends the work of Socher, Huang, Pennington, *et al.* [11] defining an unfolding recursive dependency-tree recursive autoencoder (DT-RAE). Recursive neural networks are jointly trained for both composing the sentence's words into a vector, and for decomposing that vector into words. This composition and decomposition is done by reusing a composition neural network at each vertex of the dependency tree structure, with different weight matrices for each dependency relation. The total network is trained based on the accuracy of reproducing its input word embeddings. It can be used to generate sentences, if a dependency tree structure for the output is provided. This method was demonstrated quantitatively on five examples; the generated sentences were shown to be loosely semantically similar to the originals.

Bowman, Vilnis, Vinyals, *et al.* [3] uses a a modification of the variational autoencoder (VAE) [12] with natural language inputs and outputs, to learn the sentence representations. These input and output stages are performed using long short-term memory recurrent neural networks [13]. They demonstrate a number of uses of this technique, one of which is sentence generation, in the sense of this paper. While Bowman et al. do define a generative model, they do not seek to recreate a sentence purely from its vector input, but rather to produce a series of probability distributions on the words in the sentence. These distributions can be evaluated greedily, which the authors used to give three short examples of resynthesis. They found the sentence embeddings created captured largely syntactic and loose topical information.

We note that none of the aforementioned works present any quantitative evaluations on a corpus of full sentences. We suggest that that is due to difficulties in evaluation. As noted in Iyyer, Boyd-Graber, and Daumé III [2] and Bowman, Vilnis, Vinyals, *et al.* [3], they tend to output lose paraphrases, or roughly similar sentences. This itself is a separately useful achievement to pure exact sentence generation; but it is not one that allows ready interpretation of how much information is maintained by the embeddings. Demonstration of our method at generating the example sentences used in those work is available as supplementary material[1]. As our method often can exactly recreate the original sentence from its vector representation evaluation is simpler.

Unlike current sentence generation methods, the non-compositional BOW generation method of White, Togneri, Liu, *et al.* [7] generally outputs a BOW very close to the reference for that sentence – albeit at the cost of losing all word order information. It is because of this accuracy that we base our proposed sentence generation method on it (as detailed in Section 3.1). The word selection step we used is directly based on their greedy BOW generation method. We improve it for sentence generation by composing with a word ordering step to create the sentence generation process.

1. http://white.ucc.asn.au/publications/White2016SOWE2Sent/

3 General Framework

As discussed in Section 1, and shown in Figure 1, the approach taken to generate the sentences from the vectors comes in two steps. First selecting the words used – this is done deterministically, based on a search of the embedding space. Second is to order them, which we solve by finding the most likely sequence according to a stochastic language model. Unlike the existing methods, this is a deterministic approach, rather than a machine learn method. The two subproblems which result from this split resemble more classical NP-Hard computer science problems; thus variations on known techniques can be used to solve them.

3.1 Word Selection

White, Togneri, Liu, *et al.* [7] approaches the BOW generation problem, as task of selecting the vectors that sum to be closest to a given vector. This is related to the knapsack and subset sum problems. They formally define the vector selection problem as:

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

to find the bag of vectors selected from the vocabulary set $\mathcal{V}$ which when summed is closest to the target vector $\tilde{s}$. Closeness is assessed with distance metric d. $\tilde{c}$ is the indicator function for that multi-set of vectors. As there is a one to one correspondence between word embeddings and their words, finding the vectors results in finding the words. White, Togneri, Liu, *et al.* [7] propose a greedy solution to the problem[2].

The key algorithm proposed by White, Togneri, Liu, *et al.* [7] is greedy addition. The idea is to greedily add vectors to a partial solution building towards a complete bag. This starts with an empty bag of word embeddings, and at each step the embedding space is searched for the vector which when added to the current partial solution results in the minimal distance to the target – when compared to other vectors from the vocabulary. This step is repeated until there are no vectors in the vocabulary that can be added without moving away from the solution. Then a fine-tuning step, n-substitution, is used to remove some simpler greedy mistakes.

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

2. We also investigated beam search as a possible improvement over the greedy addition and n-substitution used by White, Togneri, Liu, *et al.* [7], but did not find significant improvement. The additional points considered by the beam tended to be words that would be chosen by the greedy addition in the later steps – thus few alternatives where found.

3.2 The Ordering Problem

After the bag of words has been generated by the previous step, it must be ordered (sometimes called linearized). For example "are how , today hello ? you", is to be ordered into the sentence: "hello , how are you today ?". This problem cannot always be solved to a single correct solution. Mitchell and Lapata [14] gives the example of "It was not the sales manager who hit the bottle that day, but the office worker with the serious drinking problem." which has the same word content (though not punctuation) as "That day the office manager, who was drinking, hit the problem sales worker with a bottle, but it was not serious.". However, while a unique ordering cannot be guaranteed, finding the most likely word ordering is possible. There are several current methods for word ordering

To order the words we use a method based on the work of Horvat and Byrne [8], which uses simple trigrams. More recent works, such as beam-search and LSTM language model and proposed by Schmaltz, Rush, and Shieber [15]; or a syntactic rules based method such as presented in Zhang and Clark [16], could be used. These more powerful ordering methods internalise significant information about the language. The classical trigram language model we present is a clearer baseline for the capacity to regenerate the sentences; which then be improved by using such systems.

Horvat and Byrne [8] formulated the word ordering problem as a generalised asymmetrical travelling salesman problem (GA-TSP). Figure 2 shows an example of the connected graph for ordering five words. We extend beyond the approach of Horvat and Byrne [8] by reformulating the problem as a linear mixed integer programming problem (MIP). This allows us to take advantage of existing efficient solvers for this problem. Beyond the GA-TSP approach, a direct MIP formulation allows for increased descriptive flexibility and opens the way for further enhancement. Some of the constraints of a GA-TSP can be removed, or simplified in the direct MIP formulation for word ordering. For example, word ordering does have distinct and known start and end nodes (as shall be detailed in the next section). To formulate it as a GA-TSP it must be a tour without beginning or end. Horvat and Byrne [8] solve this by simply connecting the start to the end with a zero cost link. This is not needed if formulating this as a MIP problem, the start and end nodes can be treated as special cases. Being able to special case them as nodes known always to occur allows some simplification in the subtour elimination step. The formulation to mixed integer programming is otherwise reasonably standard.

4

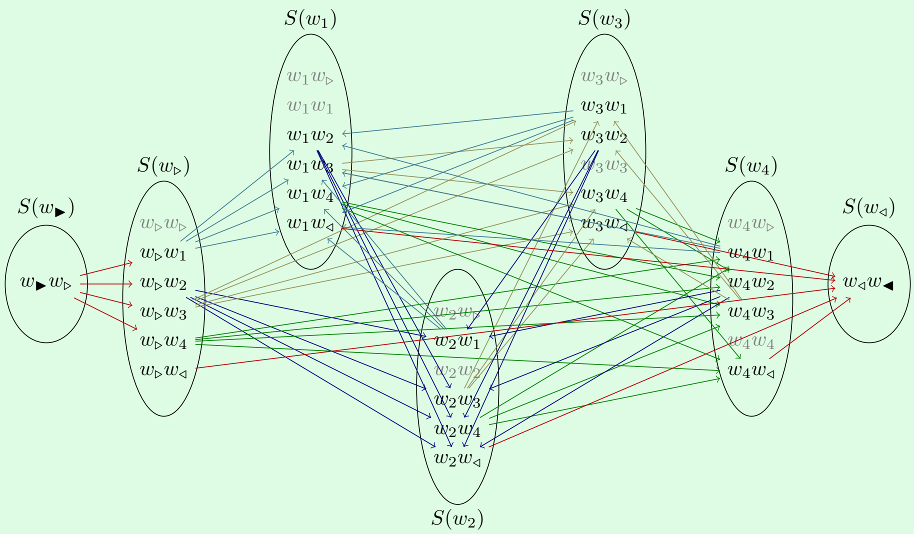


Figure 2. A graph showing the legal transitions between states, when the word-ordering problem is expressed similar to a GA-TSP. Each edge $\langle w_a, w_b\rangle \to \langle w_c, w_d\rangle$ has cost $-\log(P(w_c|w_a w_b))$. The nodes are grouped into districts (words). Nodes for invalid states are greyed out.

3.2.1 Notation

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

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

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

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

3.2.2 Optimization Model

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

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

The table of transitions to be optimized is:

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

The total cost to be minimized, is given by

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

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

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

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

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

3.2.3 Constraints

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

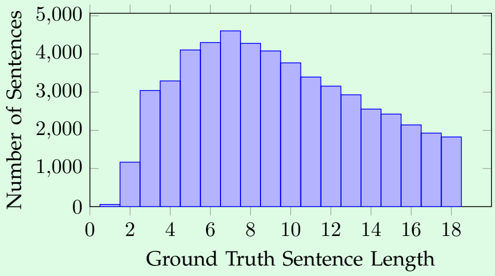


Figure 3. The distribution of the evaluation corpus after preprocessing.

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

4 Experimental Setup and Evaluations

This experimental data used in this evaluation was obtained from the data released with White, Togneri, Liu, *et al.* [7].[3]

4.1 Word Embeddings

GloVe representations of words are used in our evaluations [17]. GloVe was chosen because of the availability of a large pre-trained vocabulary of vectors.[4] The representations used for evaluation were pretrained on the 2014 Wikipedia and Gigaword 5. Other vector representations are presumed to function similarly. White, Togneri, Liu, *et al.* [7] showed that their word selection method significantly improves with higher dimensional embeddings. Due to their findings, we only evaluated 300 dimensional embeddings.

4.2 Corpus and Language Modelling

The evaluation was performed on a subset of the Books Corpus [18]. The corpus was preprocessed as in the work of White, Togneri, Liu, *et al.* [7]. This meant removing any sentences which used words not found in the embedding vocabulary.

After preprocessing, the base corpus, was split 90:10. 90% (59,694,016 sentences) of the corpus was used to fit a trigram model. This trigram language model was smoothed using the Knesler-Ney back-off method [19].

3. Available online at http://white.ucc.asn.au/publications/White2016BOWgen/
4. Available online at http://nlp.stanford.edu/projects/glove/

[6]

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

Table 1
The overall performance of the Sel. BOW+Ord. sentence generation process when evaluated on the Books corpus.

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

Table 2
The performance of the word selection step, on the Books corpus. This table shows a subset of the results reported by White, Togneri, Liu, *et al.* [7].

The remaining 10% of the corpus was kept in reserve. From the 10%, 1% (66,464 sentences) were taken for testing. From this any sentences with length over 18 words were discarded – the time taken to evaluate longer sentences increases exponentially and becomes infeasible. This left a final test set of 53,055 sentences. Figure 3 shows the distribution of the evaluation corpus in terms of sentence length.

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

4.3 Mixed Integer Programming

Gurobi version 6.5.0 was used to solve the MIP problems, invoked though the JuMP library [20]. During preliminary testing we found Gurobi to be significantly faster than the open source GLTK. Particularly for longer sentences, we found two orders of magnitude difference in speed for sentences of length 18. This is inline with the more extensive evaluations of Meindl and Templ [21]. Gurobi was run under default settings, other than being restricted to a single thread. Restricting the solver to a single thread allowed for parallel processing.

Implementation was in the Julia programming language [22]. The implementation, and non-summarised results are available for download.[6]

5 Results and Discussion

The overall results for our method (Sel. BOW+Ord.) sentence generation are shown in Table 1. Also shown are the results for just the ordering step, when the reference bag of words provided as the input (Ref. BOW+Ord.). The Perfect Sentences column shows the portion of the output sentences which exactly reproduce the input. The more forgiving BLEU Score [23] is shown to measure how close the generated sentence is to the original. The portion of cases for which there does exist a solution within the constraints of the MIP ordering problem is

5. http://www.cs.toronto.edu/~mbweb/
6. http://white.ucc.asn.au/publications/White2016SOWE2Sent/

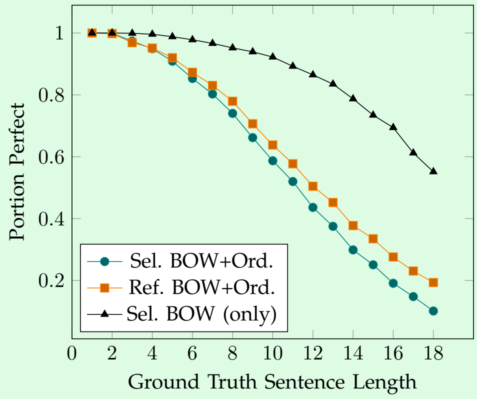


Figure 4. The portion of sentences reconstructed perfectly by the Sel. BOW+Ord. process. Shown also is the results on ordering only (Ref. BOW+Ord.), which orders the reference BOWS; and the portion of BOWs perfect from the word selection step only (Sel. BOW (only)) i.e. the input to the ordering step.

showin in Portion Feasible. In the other cases, where the MIP problem is unsolvable, for calculating the BLEU score, we order the BOW based on the order resulting from the word selection step, or in the reference case randomly.

Table 2 shows the results reported by [7] for the Word Selection step only (Sel. BOW (only)). The Perfect BOWs column reports the portion of the generated BOWs which perfectly match the reference BOWs. We also show the Mean Precision, averaged across all cases, this being the number of correct words generated, out of the total number of words generated. Similarly, the Mean Jaccard Index is shown, which is a measure of the similarities of the BOWs, being the size of the intersection of the generated BOW with the reference BOW, divided by the size of their union. We present these results to show how each step's performance impacts the overall system.

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

It can be noted that Ref. BOW+Ord. and Sel. BOW+Ord. were significantly more accurate than the best results reported by Horvat and Byrne [8]. We attribute this to Horvat and Byrne preprocessing the evaluation corpora to remove the easier sentences with 4 or less words. We did not remove short sentences from the corpus. The performance on these sentences was particularly high, thus improving the overall results on ordering.

The overall resynthesis (Sel. BOW+Ord.) degrades as the sentence length increases as shown in Figure 4. It can be seen from the figure that sentence length is a critical factor in the performance. The performance drop is largely from the complexity in the ordering step when faced with long sentences. This is evident in Figure 4, as performance degrades at almost the same rate even when using the perfect BOW (compare Ref. BOW+Ord. vs Sel. BOW+Ord.); rather than being degraded by the failures in the word selection step (Sel. BOW (only)). We can conclude that sentences with word selection failures (Sel. BOW (only)) are also generally sentences which would have word ordering failures even with perfect BOW (Ref. BOW+Ord.). Thus improving word selection, without also improving ordering, would not have improved the overall results significantly.

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

Our method considers the word selection and word ordering as separate steps. This means that unorderable words can be selected if there is an error in the first step. This is not a problem for the existing methods of Iyyer, Boyd-Graber, and Daumé III [2] and of Bowman, Vilnis, Vinyals, *et al.* [3]. Iyyer, Boyd-Graber, and Daumé III [2] guarantees grammatical correctness, as the syntax tree must be provided as an input for resynthesis – thus key ordering information is indirectly provided and it is generated into. Bowman, Vilnis, Vinyals, *et al.* [3] on the other hand integrates the language model with the sentence embedding so that every point in the vector space includes information about word order. In general, it seems clear that incorporating knowledge about order, or at least co-occurrence probabilities, should be certain to improve the selection step. Even so the current simple approach has a strong capacity to get back the input, without such enhancement.

6 Conclusion

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

The first part is the word selection problem, of going from the sum of embeddings to a bag of words. To solve this we utilised the method presented in White, Togneri, Liu, *et al.* [7]. Their greedy algorithm was found to perform well at regenerating a BOW. The second part was word ordering. This was done through a MIP bases reformulation of the work of the graph-based work of Horvat and Byrne [8]. It was demonstrated that a probabilistic language model can be used to order the bag of words output to regenerate the original sentences. While it is certainly impossible to do this perfectly in every case, for many sentences the most likely ordering is correct.

From a theoretical basis the resolvability of the selection problem, presented by White, Togneri, Liu, *et al.* [7], shows that adding up the word embeddings does preserve the information on which words were used; particularly for higher dimensional embeddings. This shows clearly that collisions do not occur (at least with frequency) such that two unrelated sentences do not end up with the same SOWE representation. This work extends that by considering if the order can be recovered based on simple corpus statistics. Its recoverability is dependent, in part, on how frequent sentences with the same words in different order are in the corpus language – if they were very frequent then non-order preserving, non-compositional representations like SOWE would be poor at capturing meaning, and the ordering task would generally fail. As the method we presented generally does succeed, we can conclude that word order ambiguity is not a dominating problem. This supports the use of simple approaches like SOWE as a meaning representation for sentences – at least for sufficiently short sentences.

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

8

6.1 Acknowledgements

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

References

[1] G. Dinu and M. Baroni, "How to make words with vectors: Phrase generation in distributional semantics", in *Proceedings of ACL*, 2014, pp. 624–633.

[2] M. Iyyer, J. Boyd-Graber, and H. Daumé III, "Generating sentences from semantic vector space representations", in *NIPS Workshop on Learning Semantics*, 2014.

[3] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space", *International Conference on Learning Representations (ICLR) Workshop*, 2016.

[4] S. Ritter, C. Long, D. Paperno, M. Baroni, M. Botvinick, and A. Goldberg, "Leveraging preposition ambiguity to assess compositional distributional models of semantics", *The Fourth Joint Conference on Lexical and Computational Semantics*, 2015.

[5] L. White, R. Togneri, W. Liu, and M. Bennamoun, "How well sentence embeddings capture meaning", in *Proceedings of the 20th Australasian Document Computing Symposium*, ser. ADCS '15, Parramatta, NSW, Australia: ACM, 2015, 9:1–9:8, ISBN: 978-1-4503-4040-3.

[6] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Towards universal paraphrastic sentence embeddings", *International Conference on Learning Representations (ICLR)*, 2016.

[7] L. White, R. Togneri, W. Liu, and M. Bennamoun, "Generating bags of words from the sums of their word embeddings", in *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, 2016.

[8] M. Horvat and W. Byrne, "A graph-based approach to string regeneration.", in *EACL*, 2014, pp. 85–95.

[9] F. M. Zanzotto, I. Korkontzelos, F. Fallucchi, and S. Manandhar, "Estimating linear models for compositional distributional semantics", in *Proceedings of the 23rd International Conference on Computational Linguistics*, Association for Computational Linguistics, 2010, pp. 1263–1271.

[10] E. Guevara, "A regression model of adjective-noun compositionality in distributional semantics", in *Proceedings of the 2010 Workshop on Geometrical Models of Natural Language Semantics*, Association for Computational Linguistics, 2010, pp. 33–37.

[11] R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning, "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection", in *Advances in Neural Information Processing Systems 24*, 2011.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes", *ArXiv preprint arXiv:1312.6114*, 2013.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] J. Mitchell and M. Lapata, "Vector-based models of semantic composition.", in *ACL*, 2008, pp. 236–244.

[15] A. Schmaltz, A. M. Rush, and S. M. Shieber, "Word ordering without syntax", *ArXiv e-prints*, Apr. 2016. arXiv: 1604.08633 [cs.CL].

[16] Y. Zhang and S. Clark, "Discriminative syntax-based word ordering for text generation", *Comput. Linguist.*, vol. 41, no. 3, pp. 503–538, Sep. 2015, ISSN: 0891-2017.

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014, pp. 1532–1543.

[18] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books", in *ArXiv preprint arXiv:1506.06724*, 2015.

[19] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling", in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, IEEE, vol. 1, 1995, pp. 181–184.

[20] M. Lubin and I. Dunning, "Computing in operations research using julia", *INFORMS Journal on Computing*, vol. 27, no. 2, pp. 238–248, 2015.

[21] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems", *Eurostat and Statistics Netherlands*, 2012.

[22] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing", 2014. arXiv: 1411.1607 [cs.MS].

[23] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation", in *Proceedings of the 40th annual meeting on association for computational linguistics*, Association for Computational Linguistics, 2002, pp. 311–318.

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

Lyndon White, Roberto Togneri, Wei Liu and **Mohammed Bennamoun**
The University of Western Australia
35 Stirling Highway, Crawley, Western Australia
lyndon.white@research.uwa.edu.au
{roberto.togneri, wei.liu, mohammed.bennamoun}@uwa.edu.au

These supplementary materials show additional examples of the performance of our method against the works of Iyyer, Boyd-Graber, and Daumé III [1] and Bowman, Vilnis, Vinyals, *et al.* [2], as of our well as on sentences with ambiguous order. Bare in mind, exact reproduction is not the goal of either prior work; nor truly is it a goal of out work. Our goal being the regeneration of sentences while preserving meaning – exact reproduction does of course meet that goal. The examples that follow should highlight the differences in the performance of the methods.

Tables 1 to 3 show quantitative examples; including comparison to the existing works. In these tables ✗ and ✓ are used to show correctness of the output in the selection (Sel.) and in the ordering (Ord.) steps.

The sentences shown in Table 1, are difficult. The table features long complex sentences containing many proper nouns. These examples are sourced from Iyyer, Boyd-Graber, and Daumé III [1]. The output from their DT-RAE method is also shown for contrast. Only 3C is completed perfectly by our method. Of the remainder the MIP word ordering problem has no solutions, except in 3D, where it is wrong, but does produce an ordered sentence. In the others the language model constraints does not return any feasible ($P(\tau) > 0$) ordering solutions. This failure may be attributed in a large part to the proper nouns. Proper nouns are very sparse in any training corpus for language modelling. The Kneser-Ney smoothed trigrams back-off only down to bigrams, so if the words of the bigrams from the training corpus never appear adjacently in the training corpus, ordering fails. This is largely the case for very rare words. The other significant factor is the sentence length.

The sentences in Table 2, are short and use common words – they are easy to resynthesis. These examples come from Bowman, Vilnis, Vinyals, *et al.* [2]. The output of their VAE based approach can be compared to that from our approach. Of the three there were two exact match's, and one failure.

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

The examples shown in Table 3 highlight sentences where the order is ambiguous – where there are multiple reasonable solutions to the word ordering problem. In both cases the word selection performs perfectly, but the ordering is varied. In 5A, the Ref. BOW+Ord. sentence and the overall Sel. BOW+Ord. sentence in word order but not in word content. This is because under the trigram language model both sentences have exactly identical probabilities, so it comes to which solution is found first, which varies on the state of the MIP solver. In 5B the word order is switched – "from Paris to London" vs "to London from Paris", which has the same meaning. But, it could also have switched the place names. In cases like this where two orderings are reasonable, the ordering method is certain to fail consistently for one of the orderings. Though it is possible to output the second (and third etc.) most probable ordering, which does ameliorate the failure somewhat. This is the key limitation which prevents this method from direct practical applications.

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

Table 2

A comparison of the output of the Two Step process proposed in this paper, to the example sentences generated by the VAE method of Bowman, Vilnis, Vinyals, *et al.* [2].

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

Table 3

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

References

[1] M. Iyyer, J. Boyd-Graber, and H. Daumé III, “Generating sentences from semantic vector space representations”, in *NIPS Workshop on Learning Semantics*, 2014.

[2] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, “Generating sentences from a continuous space”, *ArXiv preprint arXiv:1511.06349*, 2015.

[3] L. White, R. Togneri, W. Liu, and M. Bennamoun, “Generating bags of words from the sums of their word embeddings”, in *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, 2016.

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

Table 1

A comparison our method, to the example sentences generated by the DT-RAE method of Iyyer, Boyd-Graber, and Daumé III [1]. Ref. BOW+Ord. shows the word ordering step on the reference BOW. the Sel. and Ord. columns indicate if the output had the correct words selected, and ordered respectively. With ✓ indicating correct and ✗ indicating incorrect. ✖ indicates not only that ordering was not correct, but that the MIP problem had no feasible solutions at all. DT-RAE Ref. shows the result of the method of Iyyer, Boyd-Graber, and Daumé III [1], when the dependency tree of the output is provided to the generating process, whereas in DT-RAE Para. an arbitrary dependency tree is provided to the generating process. Note that the reference used as input to Sel. BOW+Ord. and Ref. BOW+Ord. sentence was varied slightly from that used in Iyyer, Boyd-Graber, and Daumé III [1] and White, Togneri, Liu, *et al.* [3], in that terminating punctuation was not removed, and nor were multiword entity references grouped into single tokens.

Finding
Word Sense Embeddings Of Known Meaning

Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun

The University of Western Australia,
35 Stirling Highway, Crawley, Western Australia
lyndon.white@research.uwa.edu.au, roberto.togneri@uwa.edu.au,
wei.liu@uwa.edu.au, mohammed.bennamoun@uwa.edu.au

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

1 Introduction

Popular word embedding vectors, such as Word2Vec, represent a word's semantic meaning and its syntactic role as a point in a vector space [1, 2]. As each word is only given one embedding, such methods are restricted to the representation of only a single combined sense, or meaning, of the word. *Word sense embeddings* generalise word embeddings to handle polysemous and homonymous words. Often these sense embeddings are learnt through unsupervised Word Sense Induction (WSI) [3–6]. The induced sense embeddings are unlikely to directly coincide with any set of human defined meaning at all, i.e. they will not match lexical senses such as those defined in a lexical dictionary, e.g. WordNet [7]. These induced senses may be more specific, more broad, or include the meanings of jargon not in common use.

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

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

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

Refitting can also be used to fit to a user provided example, giving a specific sense vector for that use. This has strong applications in information retrieval. The user can provide an example of a use of the word they are interested in. For example, searching for documents about "`banks`" as in "the river banks were very muddy". By generating an embedding for that specific sense, and by comparing with the generated embeddings in the indexed documents, we can not only pick up on suitable uses of other-words for example "`beach`" and "`shore`", but also exclude different usages, for example of a financial bank. The method we propose, using our refitted embeddings, has lower time complexity than AvgSimC [3], the current standard method for evaluating the similarity of words in context. This is detailed in Section 5.1.

We noted during refitting, that a single induced sense would often dominate the refitted representation. It is rare in natural language for the meaning to be so unequivocal. Generally, a significant overlap exists between the meaning of different lexical senses, and there is often a high level of disagreement when humans are asked to annotate a corpus [8]. We would expect that during refitting there would likewise be contention over the most likely induced sense. Towards this end, we develop a smoothing method, which we call *geometric smoothing* that de-emphasises the sharp decisions made by the (unsmoothed) refitting method. We found that this significantly improves the results. This suggests that the sharpness of sense decisions is an issue with the language model, which smoothing can correct. The geometric smoothing method is presented in Section 3.2.

We demonstrate the refitting method on sense embedding vectors induced using Adaptive Skip-Grams (AdaGram) [6], as well as our own simple greedy word sense embeddings. The method is applicable to any skip-gram-like language model that can take a sense vector as its input, and can output the probability of a word appearing in that sense's context.

The rest of the paper is organised as follows: Section 2 briefly discusses two areas of related works. Section 3 presents our refitting method, as well as our proposed geometric smoothing method. Section 4 describes the WSI embedding models used in the evaluations. Section 5 defines the RefittedSim measure for word similarity in context, and presents its results. Section 6 shows how the refitted sense vectors can be used for lexical WSD. Finally, the paper concludes in Section 7.

2 Related Works

2.1 Directly Learning Lexical Sense Embeddings

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

Iacobacci et al. [9] use a Continuous Bag of Word language model [1], using word senses as the labels rather than words. This is a direct application of word embedding techniques. To overcome the lack of a large sense labelled corpus, Iacobacci et al. use a 3rd party WSD tool, BabelFly [10], to add sense annotations to a previously unlabelled corpus.

Chen et al. [11] use a supervised approach to train sense vectors, with an unsupervised WSD labelling step. They partially disambiguate their training corpus, using word sense vectors based on WordNet; and use these labels to train their embeddings. This relabelled data is then used as training data, for finding sense embeddings using skip-grams.

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

2.2 Mapping induced senses to lexical senses

By defining a stochastic map between the induced and lexical senses, Agirre et al. [12], propose a general method for allowing WSI systems to be used for WSD. Their work was used in SemEval-2007 Task 02 [13] to evaluate all entries. Agirre et al. use a mapping corpus to find the probability of a lexical sense, given the induced sense according to the WSI system. This is more general than the approach we propose here, which only works for sense embedding based WSI. By exploiting the particular properties of sense embedding based WSI systems we propose a system that can better facilitate the use of this subset of WSI systems for WSD.

3 Proposed Refitting Framework

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

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

We initially attempted to directly optimise the sense vector to predict the example. We applied the L-BFGS [14] optimisation algorithm with the sense vector being the parameter being optimised over, and the objective being to maximise the probability of the example sentence according to the language model. This was found to generalise poorly, due to over-fitting, and to be very slow. Rather than a direct approach, we instead take inspiration from the locally linear relationship between meaning and vector position that has been demonstrated for word embeddings [1, 15, 16].

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

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

$$l(\mathbf{u}\,|\,\mathbf{c})=\sum_{\forall u_i\in\mathbf{u}} u_i P(u_i\,|\,\mathbf{c}) \tag{1}$$

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

Bengio et al. [17] describe a similar method to Equation (1) for finding (single sense) word embeddings for words not found in their vocabulary. The formula they give is as per Equation (1), but summing over the entire vocabulary of words (rather than just $\mathbf{u}$).

3.1 A General WSD method

Using the language model and application of Bayes' theorem, we define a general word sense disambiguation method that can be used for refitting (Equation (1)), and for lexical word sense disambiguation (see Section 6). This is a standard approach of using Bayes' theorem [5, 6]. We present it here for completeness.

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

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

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

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

$$P(\mathbf{c} \mid s_i) = \prod_{\forall w_j \in \mathbf{c}} P(w_j \mid s_i) \tag{2}$$

The correctness of the conditional independence assumption depends on the quality of the representation – the ideal sense representation would fully capture all information about the contexts it can appear in – thus the other contexts elements would not present any additional information, and so $P(w_a \mid w_b, s_i) = P(w_a \mid s_i)$. Given this, we have an estimate of $P(\mathbf{c} \mid s_i)$ which can be used to find $P(s_i \mid \mathbf{c})$. However, a false assumption of independence contributes towards overly sharp estimates of the posterior distribution [18], which we seek to address in Section 3.2 with geometric smoothing.

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

$$P(s_i \mid \mathbf{c}) = \frac{P(\mathbf{c} \mid s_i) P(s_i)}{\sum_{s_j \in \mathbf{s}} P(\mathbf{c} \mid s_j) P(s_j)} \tag{3}$$

This is the probability of the sense given the context.

3.2 Geometric Smoothing for General WSD

During refitting, we note that often one induced sense would be calculated as having much higher probability of occurring than the others (according to Equation (3)). This level of certainty is not expected to occur in natural languages, ambiguity is almost always possible. To resolve such dominance problems, we propose a new *geometric smoothing* method. This is suitable for smoothing posterior probability estimates derived from products of conditionally independent likelihoods. It smooths the resulting distribution, by shifting all probabilities to be closer to the uniform distribution.

We hypothesize that the sharpness of probability estimates from Equation (3) is a result of data sparsity, and of a false independence assumption in Equation (2). This is well known to occur for n-gram language models [18]. Word-embeddings language models largely overcome the data sparsity problem due to weight sharing effects [17]. We suggest that the problem remains for word sense embeddings, where there are many more classes. Thus the training data must be split further between each sense than it was when split for each word. The power law distribution of word use [19] is compounded by word senses within those used also following the a power law distribution [20]. Rare senses are liable to over-fit to the few contexts they do occur in, and so give disproportionately high likelihoods to contexts that those are similar to. We propose to handle these issues through additional smoothing.

We consider replacing the unnormalised posterior with its n_c-th root, where n_c is the length of the context. We replace the likelihood of Equation (2) with

$P_S(\mathbf{c} \,|\, s_i) = \prod_{\forall w_j \in \mathbf{c}} \sqrt[n_c]{P(w_j \,|\, s_i)}$. Similarly, we replace the prior with: $P_S(s_i) = \sqrt[n_c]{P(w_j \,|\, s_i)}$ When this is substituted into Equation (3), it becomes a smoothed version of $P(s_i \,|\, \mathbf{c})$.

$$P_S(s_i \,|\, \mathbf{c}) = \frac{\sqrt[n_c]{P(\mathbf{c} \,|\, s_i) P(s_i)}}{\sum_{s_j \in \mathbf{s}} \sqrt[n_c]{P(\mathbf{c} \,|\, s_j) P(s_j)}} \tag{4}$$

The motivation for taking the n_c-th root comes from considering the case of the uniform prior. In this case $P_S(\mathbf{c} \,|\, s_i)$ is the geometric mean of the individual word probabilities $P_S(w_j \,|\, s_i)$. Consider, if one has two context sentences, $\mathbf{c} = \{w_1, ..., w_{n_c}\}$ and $\mathbf{c}' = \{w'_1, ..., w'_{n_{c'}}\}$, such that $n'_c > n'_c$ then using Equation (2) to calculate $P(\mathbf{c} \,|\, s_i)$ and $P(\mathbf{c}' \,|\, s_i)$ will result in incomparable results as additional number of probability terms will dominate – often significantly more than the relative values of the probabilities themselves. The number of words that can occur in the context of any given sense is very large – a large portion of the vocabulary. We would expect, averaging across all words, that each addition word in the context would decrease the probability by a factor of $\frac{1}{V}$, where V is the vocabulary size. The expected probabilities for $P(\mathbf{c} \,|\, s_i)$ is $\frac{1}{V^{n_c}}$ and for $P(\mathbf{c}' \,|\, s_i)$ is $\frac{1}{V^{n_{c'}}}$. As $n_{c'} > n_c$, thus we expect $P(\mathbf{c}' \,|\, s_i) \ll P(\mathbf{c} \,|\, s_i)$. Taking the n_c-th and $n_{c'}$-th roots of $P(\mathbf{c} \,|\, s_i)$ and $P(\mathbf{c} \,|\, s_i)$ normalises these probabilities so that they have the same expected value; thus making a context-length independent comparison possible. When this normalisation is applied to Equation (3), we get the smoothing effect.

4 Experimental Sense Embedding Models

We trained two sense embedding models, AdaGram [6] and our own Greedy Sense Embedding method. During training we use the Wikipedia dataset as used by Huang et al. [4]. However, we do not perform the extensive preprocessing used in that work.

Most of our evaluations are carried out on Adaptive SkipGrams (AdaGram) [6]. AdaGram is a non-parametric Bayesian extension of Skip-gram. It learns a number of different word senses, as are required to properly model the language.

We use the implementation[2] provided by the authors with minor adjustments for Julia [21] v0.5 compatibility.

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

To confirm that our techniques are not merely a quirk of the AdaGram method or its implementation, we implemented a new simple baseline word sense embedding method. This method starts with a fixed number of randomly initialised embeddings, then greedily assigns each training case to the sense which predicts it with the highest probability (using Equation (3)). The task remains the same: using skip-grams with hierarchical softmax to predict the context words for the input

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

word sense. This is similar to [22], however it is using collocation probability, rather than distance in vector-space as the sense assignment measure. Our implementation is based on a heavily modified version of Word2Vec.jl[3].

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

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

5 Similarity of Words in Context

Estimating word similarity with context is the task of determining how similar words are, when presented with the context they occur in. The goal of this task is to match human judgements of word similarity. For each of the target words and contexts; we use refitting on the target word to create a word sense embedding specialised for the meaning in the context provided. Then the similarity of the refitted vectors can be measured using cosine distance (or similar). By measuring similarity this way, we are defining a new similarity measure.

Reisinger and Mooney [3] define a number of measures for word similarity suitable for use with sense embeddings. The most successful was AvgSimC, which has become the gold standard method for use on similarity tasks. It has been used with great success in many works [4, 11, 5].

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

$$\mathrm{AvgSimC}((\mathbf{u},\mathbf{c}),(\mathbf{u}',\mathbf{c}')) = \frac{1}{n \times n'} \sum_{u_i \in \mathbf{u}} \sum_{u'_j \in \mathbf{u}'} P(u_i \mid \mathbf{c}) P(u'_j \mid \mathbf{c}') d(u_i, u'_j) \tag{5}$$

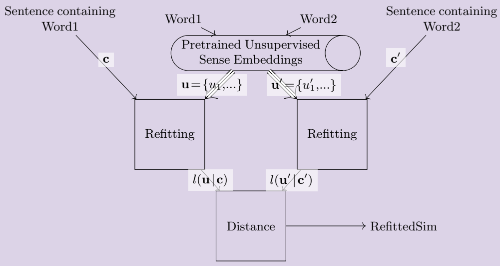
for contexts $\mathbf{c}$ and $\mathbf{c}'$, the contexts of the two words to be compared, and for $\mathbf{u} = \{u_1, ..., u_n\}$ and $\mathbf{u}' = \{u'_1, ..., u\prime_{n'}\}$ the respective sets of induced senses of the two words.

5.1 A New Similarity Measure: RefittedSim

We define a new similarity measure, RefittedSim, as the distance between the refitted sense embeddings. As shown in Figure 1 the example contexts are used to refit the induced sense embeddings of each word. This is a direct application of Equation (1).

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

Fig. 1: Block diagram for RefittedSim similarity measure



Using the same definitions as in Equation (5), RefittedSim is defined as:

$$\mathrm{RefittedSim}((\mathbf{u},\mathbf{c}),(\mathbf{u}',\mathbf{c}')) = d(l(\mathbf{u}\mid\mathbf{c}),l(\mathbf{u}'\mid\mathbf{c}') = d\left(\sum_{u_i\in\mathbf{u}} u_i P(u_i\mid\mathbf{c}), \sum_{u'_j\in\mathbf{u}'} u_i P(u'_j\mid\mathbf{c}')\right) \quad (6)$$

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

There is a notable difference in time complexity between AvgSimC and RefittedSim. AvgSimC has time complexity $O(n\|\mathbf{c}\|+n'\|\mathbf{c}'\|+n\times n')$, while RefittedSim has $O(n\|\mathbf{c}\|+n'\|\mathbf{c}'\|)$. The product of the number of senses of each word $n\times n'$, may be small for dictionary senses, but it is often large for induced senses. Dictionaries tend to define only a few senses per word – the average[4] number of senses per word in WordNet is less than three [7]. For induced senses, however, it is often desirable to train many more senses, to get better results using the more fine-grained information. Reisinger and Mooney [3] found optimal results in several evaluations near 50 senses. In such cases the $O(n\times n')$ is significant, avoiding it with RefittedSim makes the similarity measure more useful for information retrieval.

5.2 Experimental Setup

We evaluate our refitting method using Stanford's Contextual Word Similarities (SCWS) dataset [4]. During evaluation, each context paragraph is limited to 5 words to either side of the target word, as in the training.

5.3 Results

Table 1a shows the results of our evaluations on the SCWS similarity task. A significant improvement can be seen by applying our techniques.

[4] It should be noted, though, that the number of meanings is not normally distributed [23].

Table 1: Spearman rank correlation $\rho \times 100$ when evaluated on the SCWS task.

(a) For varying hyper-parameters.

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

(b) Compared to other methods . RefittedSim-S is with smoothing, and RefittedSim-SU is with uniform prior

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

The RefittedSim method consistently outperforms AvgSimC across all configurations. Similarly geometric smoothing consistently improves performance both for AvgSimC and for RefittedSim. The improvement is significantly more for RefittedSim than for AvgSimC results. In general using the unsupervised sense prior estimate from the AdaGram model, improves performance – particularly for AvgSimC. The exception to this is with RefittedSim with smoothing, where it makes very little difference. Unsurprisingly, given its low quality, the Greedy embeddings are always outperformed by AdaGram. It is not clear if these improvements will transfer to clustering based methods due to the differences in how the sense probability is estimated, compared to the language model based method evaluated on in Table 1a.

Table 1b compares our results with those reported in the literature using other methods. These results are not directly comparable, as each method uses a different training corpus, with different preprocessing steps, which can have significant effects on performance. It can been seen that by applying our techniques we bring the results of our AdaGram model from very poor ($\rho \times 100 = 43.8$) when using normal AvgSimC without smoothing, up to being competitive with other models, when using RefittedSim with smoothing. The method of Chen et al. [11], has a significant lead on the other results presented. This can be attributed to its very effective semi-supervised fine-tuning method. This suggests a possible avenue for future development in using refitted sense vectors to relabel a corpus, and then performing fine-tuning similar to that done by Chen et al.

6 Word Sense Disambiguation

6.1 Refitting for Word Sense Disambiguation

Once refitting has been used to create sense vectors for lexical word senses, an obvious used of them is to perform word sense disambiguation. In this section we refer to the lexical word sense disambiguation problem, i.e. to take a word and find its dictionary sense; whereas the methods discussed in Equations (3)

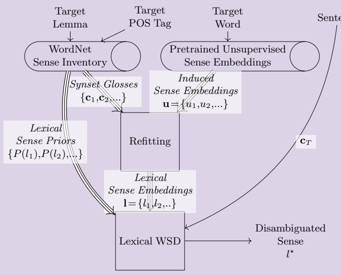


Fig. 2: Block diagram for performing WSD using refitting.

and (4) consider the more general problem, as applicable to disambiguating lexical or induced word senses depending on the inputs. Our overall process shown in Figure 2 uses both: first disambiguating the induced senses as part of refitting, then using the refitted sense vectors to find the most likely dictionary sense.

First, refitting is used to transform the induced sense vectors into lexical sense vectors. We use the targeted word's lemma (i.e. base form), and part of speech (POS) tag to retrieve all possible definitions of the word (Glosses) from WordNet; there is one gloss per sense. These glosses are used as the example sentence to perform refitting (see Section 3). We find embeddings, $\mathbf{l}=\{l_1,...,l_{n_l}\}$ for each of the lexical word senses using Equation (1). These lexical word senses are still supported by the language model, which means one can apply the general WSD method to determine the posterior probability of a word sense, given an observed context.

When given a sentence $\mathbf{c}_T$, containing a target word to be disambiguated, the probability of each lexical word sense $P(l_i\,|\,\mathbf{c}_T)$, can be found using Equation (3) (or the smoothed version Equation (4)), over the lexically refitted sense embeddings. Then, selecting the correct sense is simply selecting the most likely sense:

$$l^\star(\mathbf{l},\mathbf{c}_T)=\underset{\forall l_i\in\mathbf{l}}{\operatorname{argmax}}:P(l_i|\mathbf{c}_T)=\underset{\forall l_i\in\mathbf{l}}{\operatorname{argmax}}:\frac{P(\mathbf{c}_T\,|\,l_i)P(l_i)}{\sum_{\forall l_j\in\mathbf{l}}P(\mathbf{c}_T\,|\,l_j)P(l_j)} \tag{7}$$

6.2 Lexical Sense Prior

WordNet includes frequency counts for each word sense based on Semcor [24]. These form a prior for $P(l_i)$. The comparatively small size of Semcor means that many word senses do not occur at all. We apply add-one smoothing to remove any zero counts. This is in addition to using our proposed geometric smoothing as an optional part of the general WSD. Geometric smoothing serves a different (but related) purpose, of decreasing the sharpness of the likelihood function – not of removing impossibilities from the prior.

6.3 Experimental Setup

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

We use the weighted mapping method of Agirre et al. [12], (see Section 2.2) as a baseline alternative method for using WSI senses for WSD. We use Semcor as the mapping corpus, to derive the mapping weights.

The second baseline we use is the Most Frequent Sense (MFS). This method always disambiguates any word as having its most common meaning. Due to the power law distribution of word senses, this is a very effective heuristic [20]. We also consider the results when using a backoff to MSF when a method is unable to determine the word sense the method can report the MFS instead of returning no result (a non-attempt).

6.4 Word Sense Disambiguation Results

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

Table 2: Results on SemEval 2007 Task 7 – course-all-words disambiguation. The -*S* marks results using geometric smoothing. The ∗ marks results with MSF backoff.

The results of employing our method for WSD , are shown in Table 2. Our results using smoothed refitting, both with AdaGram and Greed Embeddings with backoff, outperform the MSF baseline [25] – noted as a surprisingly hard baseline to beat [11].

The mapping method [12] was not up to the task of mapping unsupervised senses to supervised senses, on this large scale task. The Refitting method works better. Though refitting is only usable for language-model embedding WSI, the mapping method is suitable for all WSI systems.

While not directly comparable due to the difference in training data, we note that our Refitted results, are similar in performance, as measured by F1 score, to the results reported by Chen et al. [11]. AdaGram with smoothing, and Greedy embeddings with backoff have close to the same result as reported for L2R with backoff – with the AdaGram slightly better and the Greedy embeddings slightly worse. They are exceeded by the best method reported in that paper: S2C method with backoff. Comparison to non-embedding based methods is not discussed here for brevity. Historically state of the art systems have functioned very differently; normally by approaching the WSD task by more direct means.

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

7 Conclusion

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

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

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

References

1. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv:1301.3781 (2013)
2. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014). (2014) 1532–1543
3. Reisinger, J., Mooney, R.J.: 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 (2010) 109–117
4. Huang, E.H., Socher, R., Manning, C.D., Ng, A.Y.: 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 (2012) 873–882
5. Tian, F., Dai, H., Bian, J., Gao, B., Zhang, R., Chen, E., Liu, T.Y.: A probabilistic model for learning multi-prototype word embeddings. In: COLING. (2014) 151–160

6. Bartunov, S., Kondrashkin, D., Osokin, A., Vetrov, D.P.: Breaking sticks and ambiguities with adaptive skip-gram. CoRR **abs/1502.07257** (2015)
7. Miller, G.A.: Wordnet: a lexical database for english. Communications of the ACM **38** (1995) 39–41
8. Véronis, J.: A study of polysemy judgements and inter-annotator agreement. In: Programme and advanced papers of the Senseval workshop. (1998) 2–4
9. Iacobacci, I., Pilehvar, M.T., Navigli, R.: Sensembed: learning sense embeddings for word and relational similarity. In: Proceedings of ACL. (2015) 95–105
10. Moro, A., Raganato, A., Navigli, R.: Entity Linking meets Word Sense Disambiguation: a Unified Approach. Transactions of the Association for Computational Linguistics (TACL) **2** (2014) 231–244
11. Chen, X., Liu, Z., Sun, M.: A unified model for word sense representation and disambiguation. In: EMNLP, Citeseer (2014) 1025–1035
12. Agirre, E., Martínez, D., De Lacalle, O.L., Soroa, A.: 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 (2006) 89–96
13. Agirre, E., Soroa, A.: Semeval-2007 task 02: Evaluating word sense induction and discrimination systems. In: Proceedings of the 4th International Workshop on Semantic Evaluations. SemEval '07, Stroudsburg, PA, USA, Association for Computational Linguistics (2007) 7–12
14. Nocedal, J.: Updating quasi-newton matrices with limited storage. Mathematics of computation **35** (1980) 773–782
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. (2013) 3111–3119
16. Mikolov, T., Yih, W.t., Zweig, G.: Linguistic regularities in continuous space word representations. In: HLT-NAACL. (2013) 746–751
17. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. The Journal of Machine Learning Research (2003) 137–186
18. Rosenfeld, R.: Two decades of statistical language modeling: Where do we go from here? Proceedings of the IEEE **88** (2000) 1270–1278
19. Zipf, G.: Human behavior and the principle of least effort: an introduction to human ecology. Addison-Wesley Press (1949)
20. Kilgarriff, A. In: How Dominant Is the Commonest Sense of a Word? Springer Berlin Heidelberg, Berlin, Heidelberg (2004) 103–111
21. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. (2014)
22. Neelakantan, A., Shankar, J., Passos, A., McCallum, A.: Efficient non-parametric estimation of multiple embeddings per word in vector space. arXiv preprint arXiv:1504.06654 (2015)
23. Zipf, G.K.: The meaning-frequency relationship of words. The Journal of general psychology **33** (1945) 251–256
24. Tengi, R.I.: Design and implementation of the WordNet lexical database and searching software. In: WordNet: an electronic lexical database, The MIT Press, Cambridge, Massachusetts. (1998) 105
25. Navigli, R., Litkowski, K.C., Hargraves, O.: Semeval-2007 task 07: Coarse-grained english all-words task. In: Proceedings of the 4th International Workshop on Semantic Evaluations. SemEval '07, Stroudsburg, PA, USA, Association for Computational Linguistics (2007) 30–35

Part IV

Tooling

DataDeps.jl: Repeatable Data Setup for Reproducible Data Science

Editor:

Abstract

We present a framework DataDeps.jl for the reproducible handling of static datasets to enhance the repeatability of software scripts used in the data and computational sciences. DataDeps.jl is a library for the Julia programming language. 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 via traditional means of a software dependency, as the extension does not have to worry about ensuring the data is setup for its dependency. DataDeps.jl makes it easier to rerun another authors code, thus enhancing the reproducibility of data science research.

Keywords: Julia; data; data management; data dependencies; reproducible science; downloading; computational environment setup; continuous integration; software practices.

1. Introduction

In the movement for reproducible data and computational sciences there have been two key additional requests from authors: 1. Make your research code public, 2. Make your data public (Goodman et al., 2014). In practice however, this alone is not enough to ensure that even the purely computational results can be replicated. To get another author's code running on a new machine is often non-trivial. One is more likely to see errors preventing the code from running, than in the results. One aspect of this is the data setup, how to acquire the data, and how to connect it to the code.

DataDeps.jl simplifies the automation of the data setup step for Julia (Bezanson et al., 2014) packages and research software. It allows the code to depend on data, and have that data automatically downloaded as required. DataDeps.jl is a compact tool with a single job, following the unix philosophy of doing one job well. It increases repeatability of any scientific code that uses data. It does this by decreasing the effort to get such a program working on a new system, while also decreasing the effort to write the code in the first place.

Vandewalle et al. (2009) distinguishes 6 degrees of reproducibility for scientific code. To achieve either of the 2 highest levels, requires that "The results can be easily reproduced by an independent researcher with at most 15 min of user effort". It is our experience that one can often expend much of that time just on setting up the data. This involves reading the instructions, locating the download link, transferring it to the right location, extracting an archive, and identifying how to inform the script as to where the data is located. These tasks are automatable therefore should be automated; to save user time, and remove the

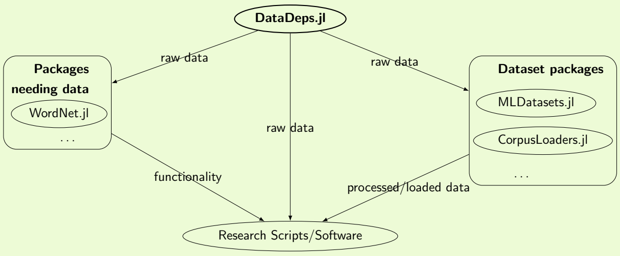


Figure 1: The package ecosystem using DataDeps.jl. DataDeps.jl can be used directly by research code to produce data; or it can be used indirectly by making use of a package which uses DataDeps.jl to manage its data dependencies.

opportunity for mistakes, as per the key practice identified by Wilson et al. (2014) "let the computer do the work".

Automating the data setup is part of achieving full automation of the setup in a new environment. DataDeps.jl handles the setup of data dependencies in Julia, while BinDeps.jl (and other packages) handle the binary software dependency. The automation of the full setup is required to make automated testing possible, for example using Continuous Integration testing services, such as TravisCI or AppVeyor. Automated testing is already ubiquitous in use amongst researchers and developers using Julia, but rarely for parts where data is involved. If the full setup can be automated, such that it can run on a clean CI environment, it is almost certain that any human trying to reproduce your work using your code will be able to do so with little effort.

2. DataDeps.jl

2.1 Ecosystem

As shown in Figure 1, DataDeps.jl exists as a higher level dependency. Some research code employing commonly used datasets will not directly use DataDeps.jl at all; rather it will depend on a dataset loading package such as MLDatasets.jl, or CorpusLoaders.jl, which in turn use DataDeps.jl as their back-end to actually manage their data. These dataset packages provide a more friendly access to data that has been cleanly loaded into Julia data structures. For more obscure, or self-created data sets the research code would employ DataDeps.jl directly, and handle the loading of the data itself. Finally, DataDeps.jl is also used by packages which require data to perform their functionality. Examples include WordNet.jl which provides functions for non-trivial lookups on the WordNet (Miller, 1995) database, or any packages which require a trained machine-learning based model – the data in this case being the serialised model. Packages and research code alike depend on data, and DataDeps.jl exists to fill that need.

DataDeps.jl

2.2 Three common questions about research data

DataDeps.jl is designed around solving common issues researchers have with their file-based data. The three questions 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, I'm going to have to reconfigure things

Redistribution Is it ok to include it with my work? I don't own copyright on this data.

- Am I allowed to redistribute this data?
- What if people getting the data from me as part of my experiments don't realize the original creator?

Replication How can I be sure someone running my code has the same data?

- It is possible for later users to download the wrong data
- It is possible for data to be corrupted, or modified (even maliciously).

These three issues are solved by DataDeps.jl.

2.3 Functionality

When declaring a data dependency the developer needs only to declare a data registration block. For a normal data dependency this is a small piece of code consisting of:

Name used to refer to the dependency in code, as in the described `datadep"NAME"`. This solves the **Storage Location** issues, as will be discussed below data is referred to by name, not by path. The data can be moved to any where on the load path, without changing the code. The load path can be configured using an environment variable, allowing it to differ across environments, by default it includes a very wide range of common locations.

Remote path a path or a list of paths (normally URLs) to remote copies of the data. By fetching the data from a remote location, the issues of **Redistribution** are avoided, as you are not distributing data merely using existing links. This also avoids issues with storing data in version control, which causes problems for many version control systems.

Message the message to be displayed to the user before any download occurs, it is followed by requiring user acceptance to continue. This allows the user to be informed of the data's true origin, and display other key info like papers about the data to be cited. Together with fetching from a remote path, this solves the **Redistribution** issue.

Checksum a checksum (or list of checksums) for the files being fetch. Via external packages a wide variety of checksums can be supported, including MD5 and all versions of SHA. This, together with the automation of the whole process, solves the **Replication** issue, ensuring the data the user receives is as it was when the software was developed. If the checksum is not provided, then one is calculated, together with a warning that this line should be added to the registration block; allowing the researcher to avoid having to use unfamiliar tools to calculate the checksum.

Post fetch method a function (or list of functions) to call on the fetched files. Most commonly used to unpack an archive. This completes the automation of data setup, contribution towards the **Replication** issue.

The registration block is effectively metadata, instructing on how to obtain the data.

3

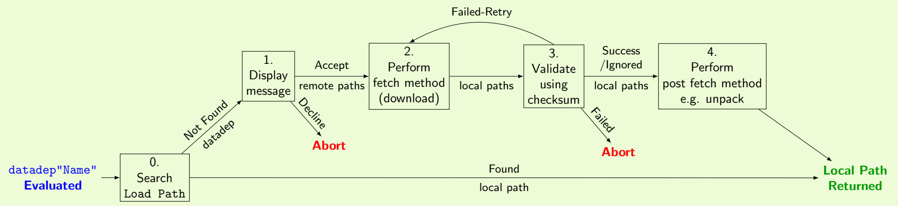


Figure 2: The core process for resolving a data dependency. Each step in the top path is linked to one of the properties declared in the registration block.

Once the data dependency has been declared using the registration block, it can be referred to by name using a datadep string, written `datadep"Name"`. It can be treated just like it were an absolute path to the data, however it is actually a a string macro (related to LISP's reader macro). At compile time it is replaced with a block of code which performs the operation shown in Figure 2. It always resolves to a being a string to an absolute path to the data, even if that means it must download the data first.

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; the registration block here would only include a name and a message. This allows users to manage the data themselves by putting a folder somewhere on the load path. They can still refer to it using the `datadep"Name"`. If it is not found, the message is displayed which should include instructions on fetching it manually. Before that research is submitted for publications, the user can upload their data to a repository such as DataDryad, FigShare or another archival repository, and update the registration.

The use of archival repositories as the remotepath host should combat URL decay (Wren, 2008). Though it remains that URLs in DataDeps.jl are just as vulnerable to URL decay as those in manual instructions. Unlike manual instructions however, periodic automated tests can easily be use to confirm if the URLs remain valid.

3. 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 setting up for good practices now for the still young Julia programming language, better scientific code can be written in the future.

4. Availability and Requirements

DataDeps.jl is verified to work on Windows 7+, Linux, Mac OS X, with Julia 0.6. It depends on HTTP.jl, Reexport.jl, and SHA.jl, which are automatically installed when the package is installed through the normal Julia package manager. The source code, documentation, and issue tracker can be found at `https://github.com/oxinabox/DataDeps.jl`.

References

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. 2014. URL `http://arxiv.org/abs/1411.1607`.

Alyssa Goodman, 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. Ten simple rules for the care and feeding of scientific data. *PLOS Computational Biology*, 10(4):1–5, 04 2014. doi: 10.1371/journal.pcbi.1003542. URL `https://doi.org/10.1371/journal.pcbi.1003542`.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. URL `http://mooo.cf/20140731/Reference/1995%20-%20Miller%20-%20WordNet%20a%20lexical%20database%20for%20English.pdf`.

P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3):37–47, May 2009. ISSN 1053-5888. doi: 10.1109/MSP.2009.932122.

Greg Wilson, 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. Best practices for scientific computing. *PLOS Biology*, 12(1):1–7, 01 2014. doi: 10.1371/journal.pbio.1001745. URL `https://doi.org/10.1371/journal.pbio.1001745`.

Jonathan D Wren. Url decay in medline: a 4-year follow-up study. *Bioinformatics*, 24(11):1381–1385, 2008. URL `https://academic.oup.com/bioinformatics/article/24/11/1381/191025`.