Part I

Literature Review

Chapter 1

Overview of the field

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"

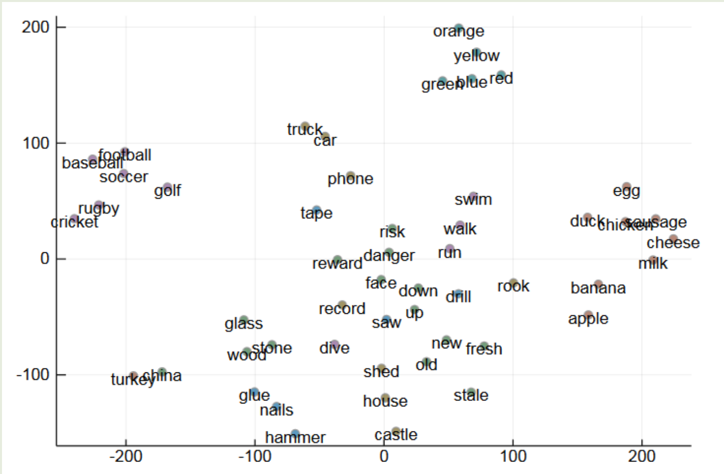


Figure 4.1: Some word embeddings from the FastText project (Bojanowski et al. 2016). They were originally 300 dimensions but have been reduced to 2 using t-SNE (Maaten and Hinton 2008) algorithm. The colors are from 5 manually annotated categories done before this visualisation was produced: `foods`, `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

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?"

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:

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

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" 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"

4.1.1.1 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.

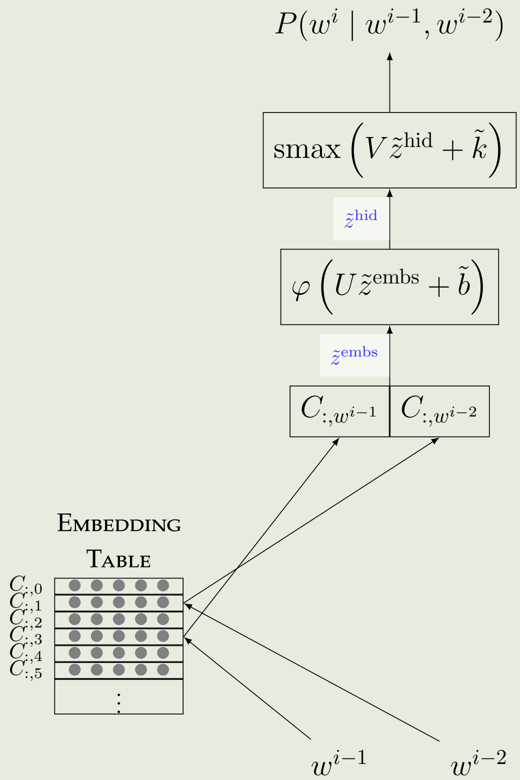


Figure 4.2: The Neural Trigram Language Model

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):

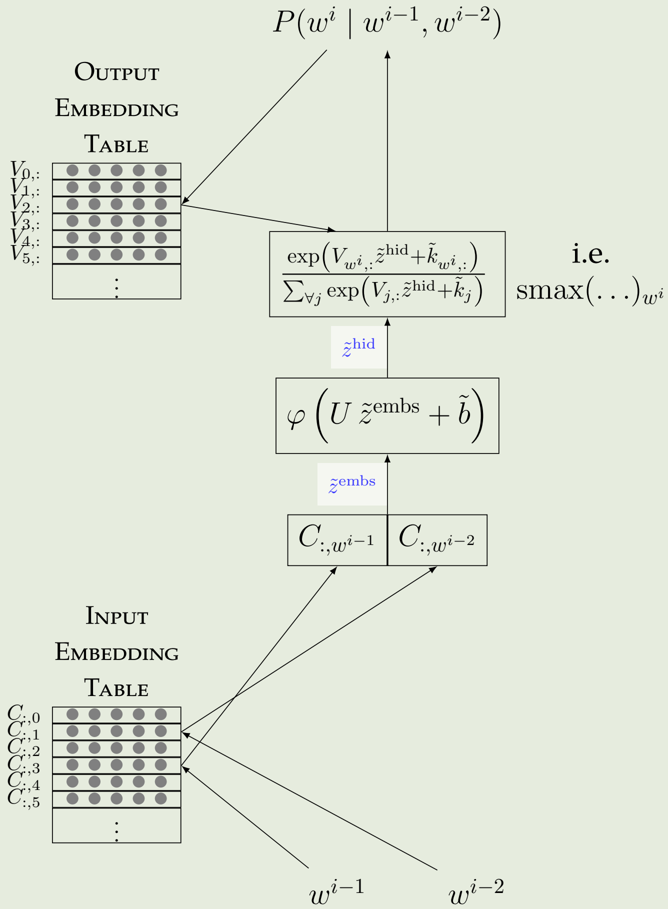
$$\mathrm{smax}(V\tilde{z}^{\mathrm{hid}} + \tilde{k})_{w^i} = \frac{\exp\left(V_{w^i,:}\tilde{z}^{\mathrm{hid}} + \tilde{k}_{w^i}\right)}{\sum_{\forall j} \exp\left(V_{j,:}\tilde{z}^{\mathrm{hid}} + \tilde{k}_j\right)} \tag{4.5}$$

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:

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

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 $\mathbb{E}(V \tilde{z}^{\text{hid}}) = 0$. Given that V is normally[1] initialized as a zero mean Gaussian, this condition is at least initially true. This suggests, interestingly, that we can predetermine good initial values for the output bias $\tilde{k}$ using the log of the unigram probabilities. In practice this is not required, as it is learnt rapidly by the network during training.

[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 ??.

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

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

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

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

4.1 Representations for Language Modeling

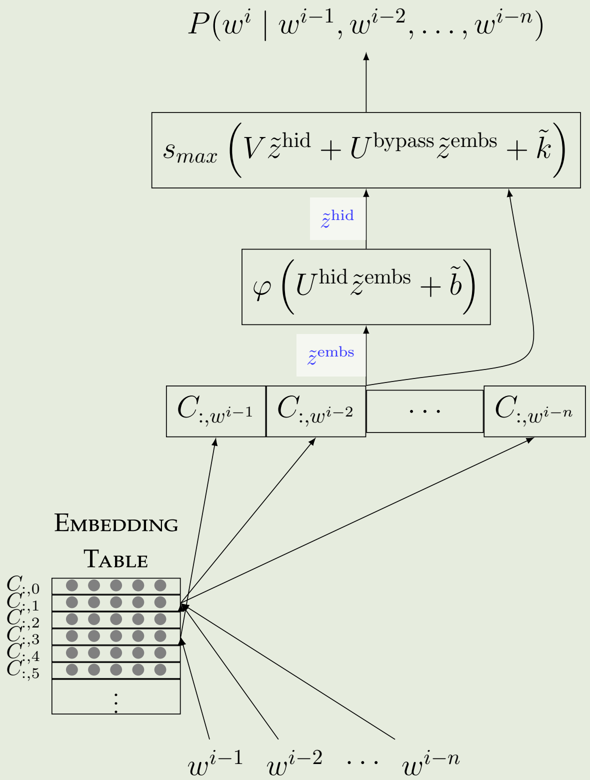


Figure 4.4: Neural Probabilistic Language Model

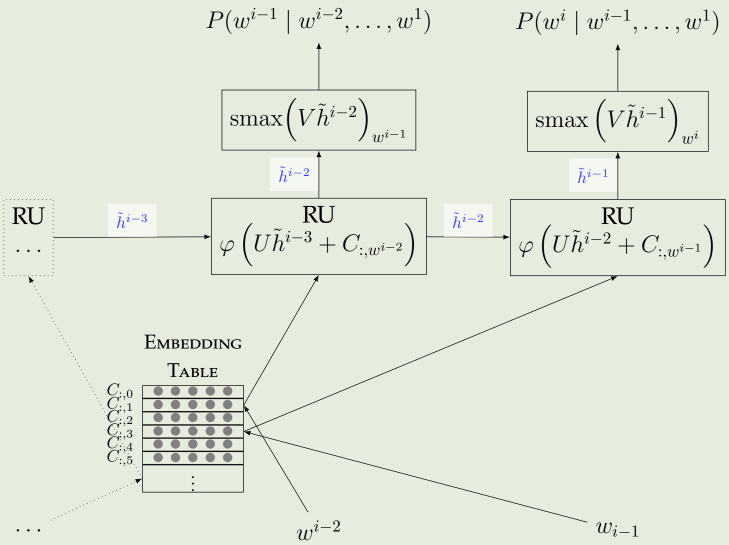


Figure 4.5: RNN Language Model. The RU equation shown is the basic RU used in Mikolov, Karafiát, et al. (2010). It can be substituted for a LSTM RU or an GRU as was done in Sundermeyer, Schlüter, and Ney (2012) and Jozefowicz, Zaremba, and Sutskever (2015), with appropriate changes.

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.

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

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

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

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-

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

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.

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).

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.

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 in-fact 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

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"

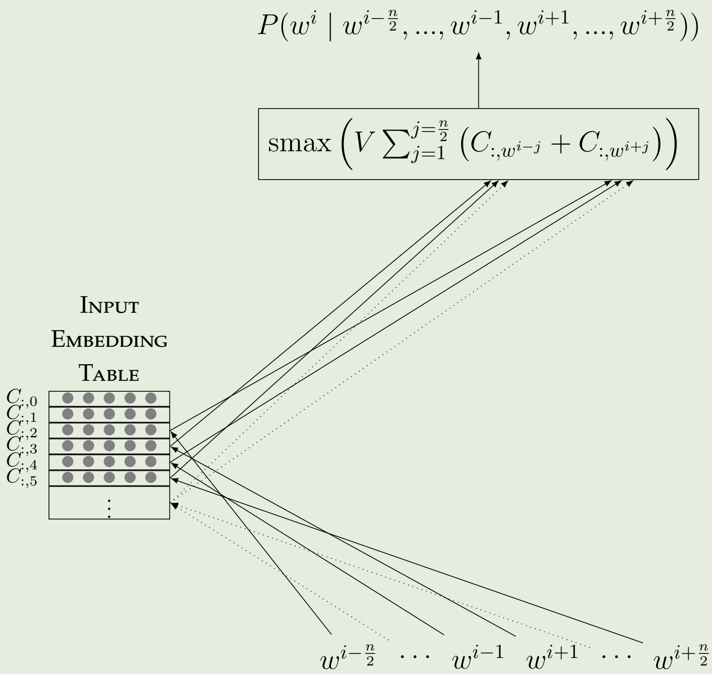


Figure 4.6: CBOW Language Model

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:

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

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.

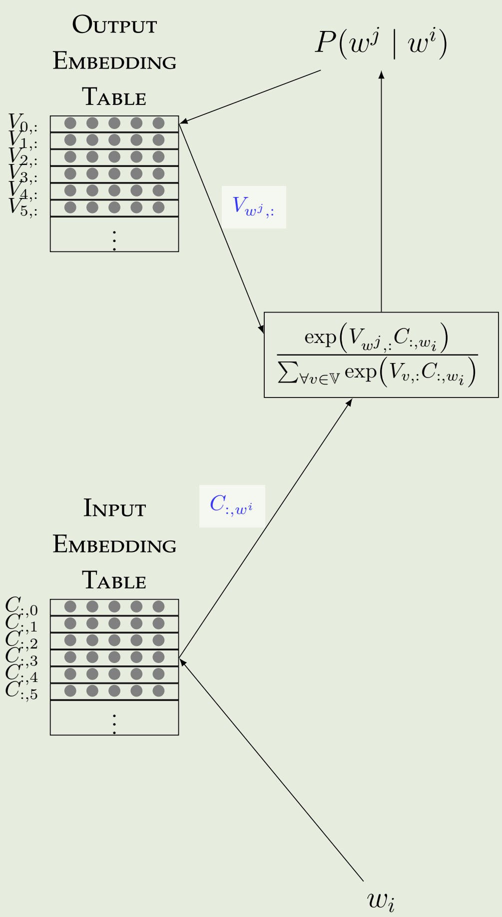


Figure 4.7: Skip-gram language Language Model. Note that the probability $P(w^j \mid w^i)$ is optimised during training for every w^j in a window around the central word w^i. Note that the final layer in this diagram is just a softmax layer, written in in output embedding form.

4.2.2 Skip-gram

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"

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:

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

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:

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

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

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.

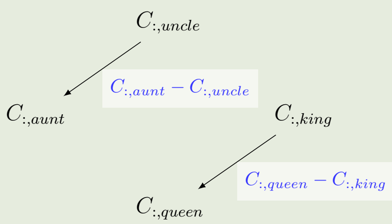


Figure 4.8: Example of analogy algebra

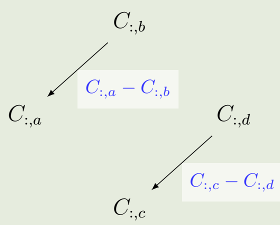


Figure 4.9: Vectors involved in analogy ranking tasks, this may help to understand the math in Equation (4.19)

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.

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

This is done via the minimisation of

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

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.

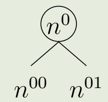


Figure 4.10: Tree for 2 words

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.

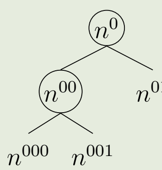


Figure 4.11: Tree for 3 words

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

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

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;

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.

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

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

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).

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.

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

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.

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

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

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

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

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

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"

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

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

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

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

83

4 Word Representations

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

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

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.

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

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

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

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

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

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:

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

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 Co-factorization."

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.

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.

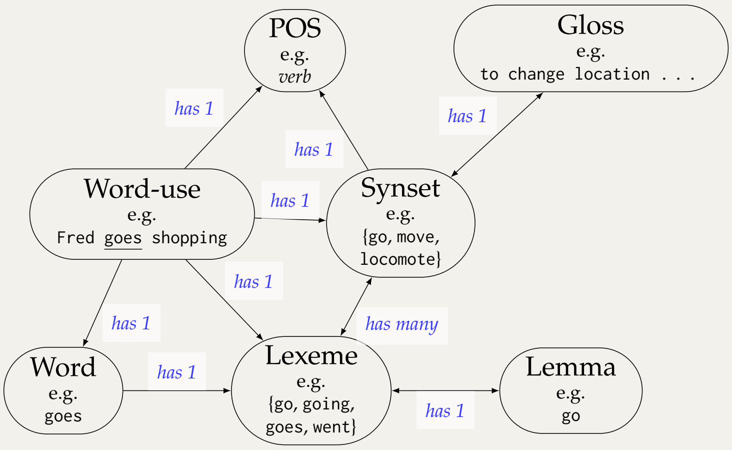


Figure 5.1: The relationship between terms used to discuss various word sense problems. The lemma is used as the representation for the lexeme, for WordNet's purposes when indexing. For many tasks each the word-use is pre-tagged with its lemma and POS tag, as these can be found with high reliability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

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

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,*

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

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)).

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 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: $\mathrm{argmax}_{\forall s^j} P(s^j \mid w^i, p_i)$.

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

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.

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?"

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 Word Sense Representations

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

5.2.1 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.

98

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)

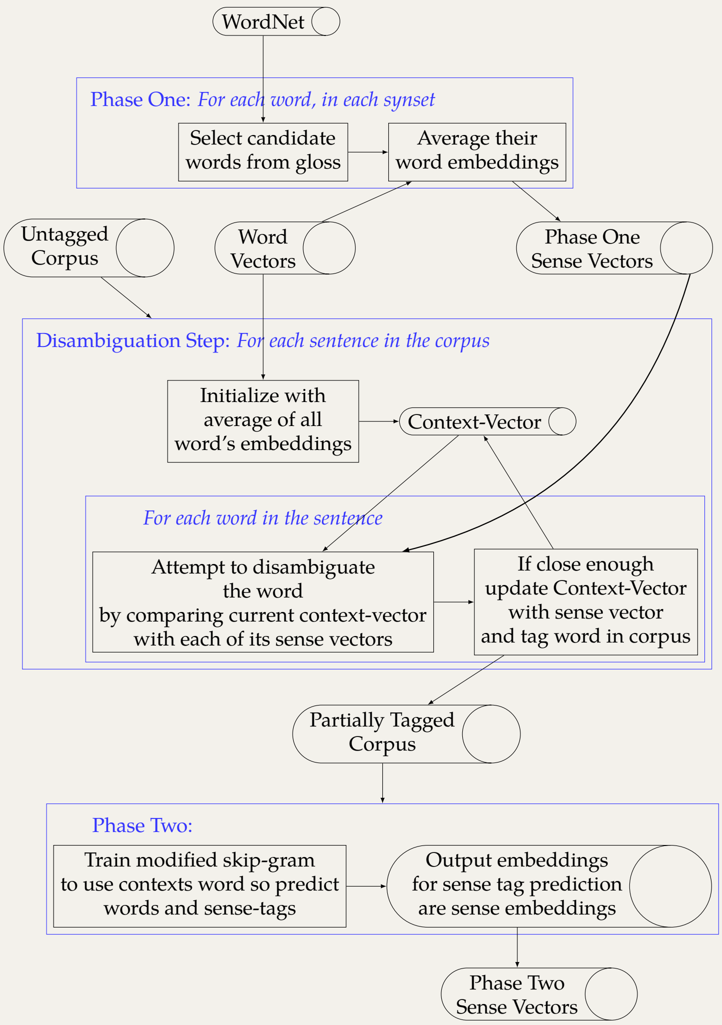
In this section we will discuss methods for finding a word sense without reference to a standard set of senses. Such systems must discover the word senses at the same

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.

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.

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).

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/

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

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:

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

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:

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

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

Equation (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

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

5 Word Sense Representations

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.

110

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 proto-

Bibliography

types”. 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*.

112

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

6.1 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-

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

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"

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.

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"

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

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

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"

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 (Rehůrek and Sojka 2010),

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

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

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"

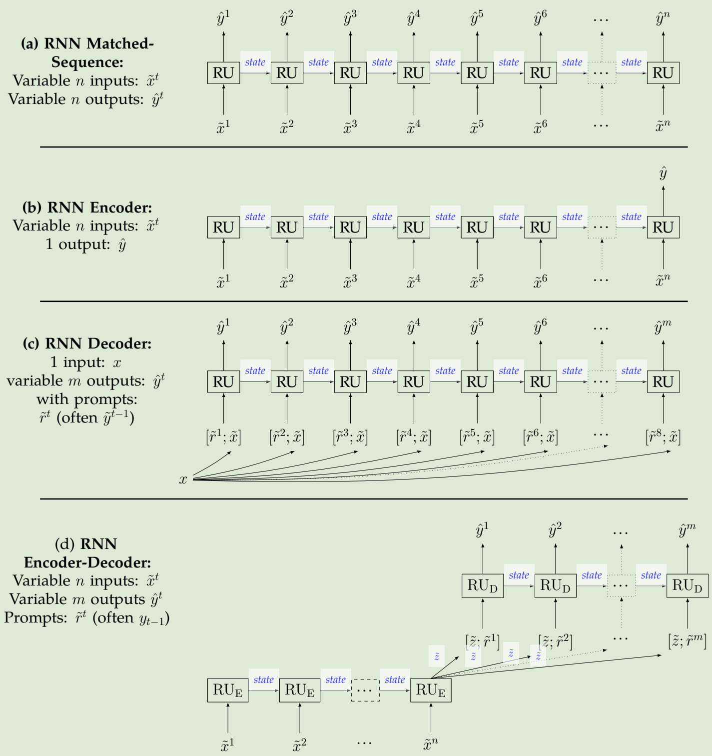


Figure 6.1: The unrolled structure of an RNN for use in (a) Matched-sequence (b) Encoding, (c) Decoding and (d) Encoding-Decoding (sequence-to-sequence) problems. RU is the recurrent unit – the neural network which reoccurs at each time step. (Repeated from Figure 3.1)

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"

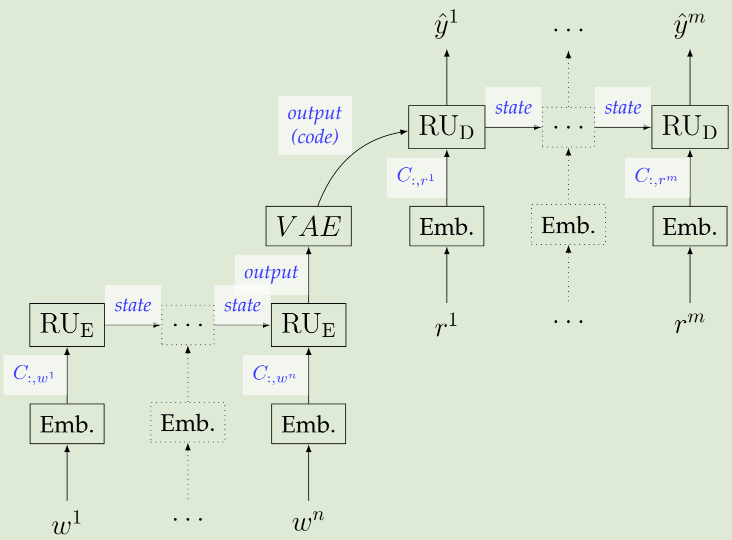


Figure 6.2: The VAE plus encoder-decoder of Bowman, Vilnis, et al. (2016). Note that during training, $\hat{y}^i = w^i$, as it is an autoencoder model. As is normal for encoder-decoders the prompts are the previous output (target during training, predicted during testing): $r^i = y^{i-1}$, with $r^1 = y^0 =$ <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, *6.3 Structured Models*

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.

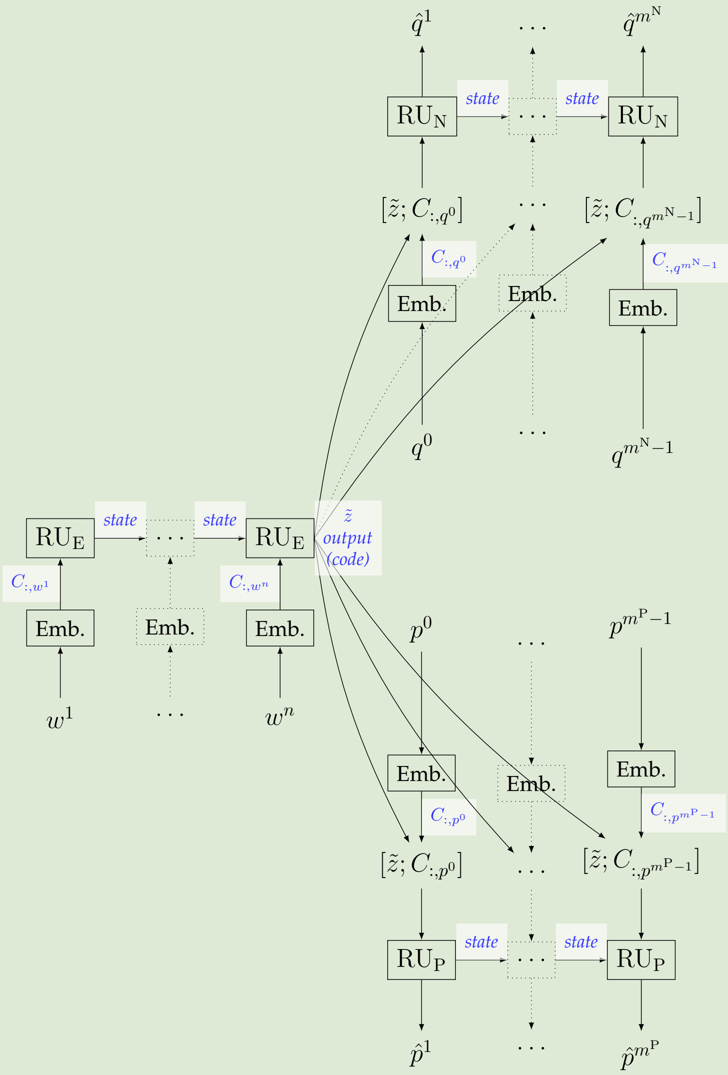


Figure 6.3: The skip-thought model (Kiros et al. 2015). Note that for the next and previous sentences respectively the outputs are $\hat{q}^i$ and $\hat{p}^i$, and the prompts are q^{i-1} and p^{i-1}. As there is no intent to use the decoders after training, there is no need to worry about providing an evaluation-time prompt, so the prompt is always the previous word. $p^0 = p^{m^P} = q^0 = q^{m^q} =$ <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"

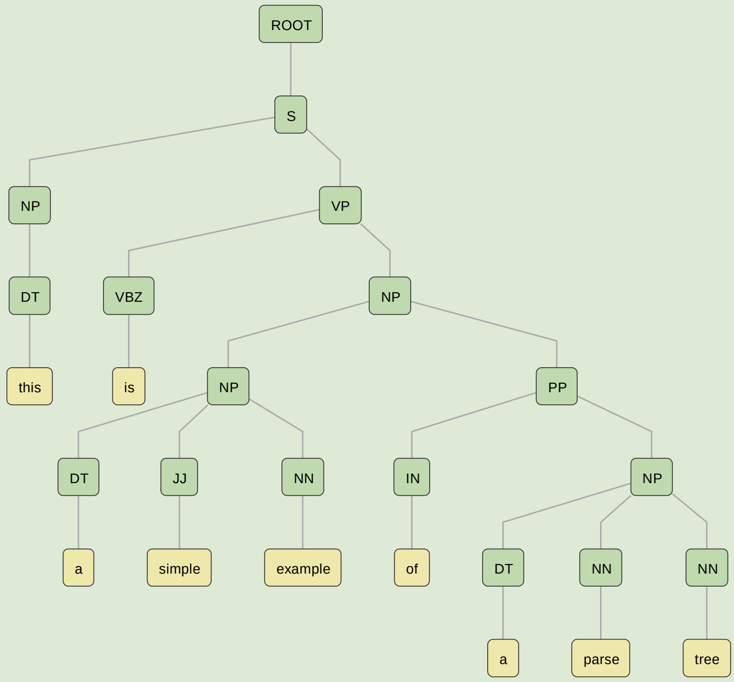


Figure 6.4: A constituency parse tree for the sentence: `This is a simple example of a parse tree.` In this diagram the leaf nodes are the input words, their intimidate parents are their POS tags, and the other nodes with multiple children represent sub-phrases of the sentence, for example NP is a Noun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

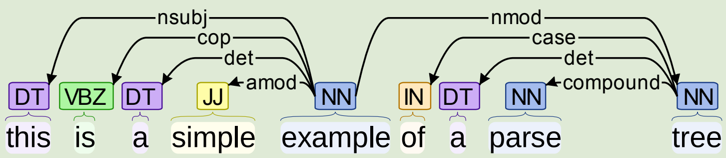


Figure 6.5: A dependency parse tree for the sentence `This is a simple example of a parse tree`, This flattened view may be misleading. `example` is at the peak of the tree, with direct children being: `this`,`is`,`a`,`simple`, and `tree`. `tree` has direct children being: `of`,`a`, and `parse`.

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

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

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 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

6.3 Structured Models

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

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

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

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.

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

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

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

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

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

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.

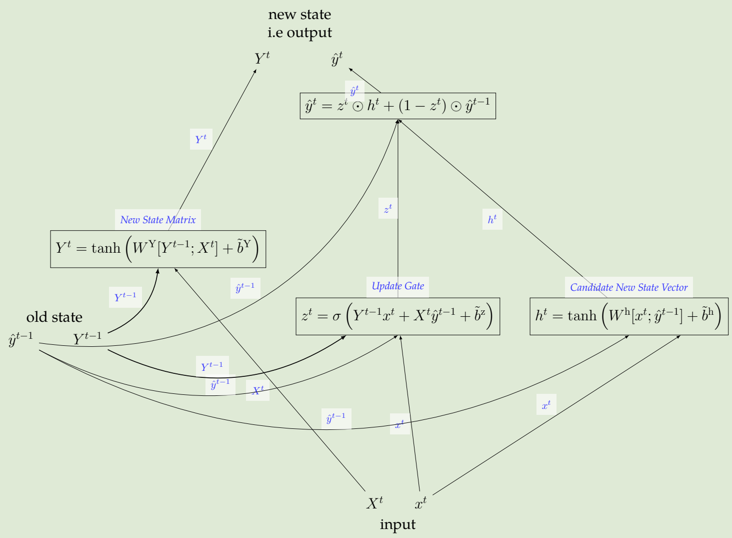


Figure 6.6: A Matrix Vector recurrent unit

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 para-

graphs". 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`.
Rehůřek, 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

Discussion of the state of the field

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.