24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Capturing Word Order in Averaging Based Sentence Embeddings

Jae Hee Lee[1] and **Jose Camacho Collados**[2] and **Luis Espinosa Anke**[3] and **Steven Schockaert**[4]

Abstract. One of the most remarkable findings in the literature on sentence embeddings has been that simple word vector averaging can compete with state-of-the-art models in many tasks. While counter-intuitive, a convincing explanation has been provided by Arora et al., who showed that the bag-of-words representation of a sentence can be recovered from its word vector average with almost perfect accuracy. Beyond word vector averaging, however, most sentence embedding models are essentially black boxes: while there is abundant empirical evidence about their strengths and weaknesses, it is not clear why and how different embedding strategies are able to capture particular properties of sentences. In this paper, we focus in particular on how sentence embedding models are able to capture word order. For instance, it seems intuitively puzzling that simple LSTM autoencoders are able to learn sentence vectors from which the original sentence can be reconstructed almost perfectly. With the aim of elucidating this phenomenon, we show that to capture word order, it is in fact sufficient to supplement standard word vector averages with averages of bigram and trigram vectors. To this end, we first study the problem of reconstructing bags-of-bigrams, focusing in particular on how suitable bigram vectors should be encoded. We then show that LSTMs are capable, in principle, of learning our proposed sentence embeddings. Empirically, we find that our embeddings outperform those learned by LSTM autoencoders on the task of sentence reconstruction, while needing almost no training data.

1 Introduction

Sentence embeddings are vector representations that capture the meaning of a given sentence. Several authors have empirically found that surprisingly high-quality sentence embeddings can be obtained by simply adding up the word vectors from a given sentence [29], sometimes in combination with particular weighting and post-processing strategies [3]. Since word vectors are typically compared in terms of their cosine similarity, summing up word vectors corresponds to a form of averaging (i.e. the direction of the resulting sum averages the direction of the individual word vectors). For this reason, we refer to this class of methods as averaging based sentence embeddings. Recently, [2] has shed some light on the remarkable effectiveness of averaging based embeddings, using insights from the theory of compressed sensing. Essentially the paper showed that from a given sum of word vectors, it is almost always possible to reconstruct the corresponding bag-of-words representation, as long as the dimensionality of the vectors is sufficiently high relative to the length of the sentence. Despite the fact that word vector averages are thus far more expressive than they may intuitively appear, they obviously cannot capture information about word order. While this only affects their ability to capture sentence similarity in a minimal way [3], it is nonetheless an important limitation of such representations.

Beyond averaging based strategies, even relatively standard neural network models are able to learn sentence embeddings which capture word order nearly perfectly. For instance, LSTM autoencoders learn an encoder, which maps a sequence of words onto a sentence vector, together with a decoder, which aims to reconstruct the original sequence from the sentence vector. While empirical results clearly show that such architectures can produce sentence vectors that capture word order, from an intuitive point of view it remains puzzling that such vectors can arise from a relatively simple manipulation of word vectors. The aim of this paper is to develop a better understanding of how order-encoding sentence vectors can arise in such a way. In particular, we want to find the simplest extension of word vector averaging which is sufficient for learning sentence vectors that capture word order. To this end, we follow the following intuition: since averages of word vectors allow us to recover the bag-of-words representation of a sentence [3], averages of bigram vectors may allow us to recover the bag-of-bigrams, and similar for longer n-grams. This paper makes the following three main contributions:

1. We present an in-depth study about how bigrams should be encoded to maximize the probability that bags-of-bigrams can be reconstructed from the corresponding bigram vector averages (Section 3).
2. We empirically show that simply concatenating averaged vector representations of unigrams, bigrams and trigrams gives us sentence embeddings from which the original sentence can be recovered more faithfully than from sentence embeddings learned by LSTM autoencoders (Section 4).
3. We show that LSTM architectures are capable of constructing such concatenations of unigram, bigram and trigram averages. Even though LSTM autoencoders in practice are unlikely to follow this exact strategy, this clarifies why simple manipulations of word vectors are sufficient for encoding word order (Section 5).[5]

2 Related Work

Sentence embedding is a widely-studied topic in the field of representation learning. Similarly to the predictive objective of word embedding models such as Skip-gram [18], recent unsupervised sentence embedding models have based their architecture on predicting

[1] Cardiff University, UK, email: leejh3@cardiff.ac.uk
[2] Cardiff University, UK, email: camachocolladosj@cardiff.ac.uk
[3] Cardiff University, UK, email: espinosa-ankel@cardiff.ac.uk
[4] Cardiff University, UK, email: schockaerts1@cardiff.ac.uk

[5] The code for reproducing the results in the paper can be downloaded from `https://github.com/dschaehi/capturing-word-order`

the following sentence, given a target sentence. A popular example of this kind of model is Skip-Thought [15]. In this model a recurrent neural network is employed as part of a standard sequence to sequence architecture for encoding and decoding. While using the next sentence as a supervision signal has proved very powerful, one disadvantage of this model is that it is computationally demanding. For this reason, variations have been proposed which frame the objective as a classification task, instead of prediction, which allows for more efficient implementations. A representative example of this strategy is the quick thoughts model from [16]. As another line of work, large pre-trained language models such as BERT have shown strong performance in many downstream tasks [10], and are also able to extract high-quality sentence embeddings [25]. In spite of these advancements, [3] showed that a simple average of word embeddings can lead to competitive performance, while being considerably more transparent, computationally efficient, and lower-dimensional than most other sentence embeddings. In this paper we focus on the latter kind of approach.

To the best of our knowledge, the idea of capturing word order using averaging based strategies has not previously been considered. However, there is a considerable amount of work on averaging based sentence embeddings, which we discuss in Section 2.1. There is also some related work on capturing word order in sentence vectors, as we discuss in Section 2.2.

2.1 Averaging Based Sentence Embeddings

Several extensions to the basic approach of word vector averaging have already been proposed. For example, FastSent [12] learns word vectors in such a way that the resulting averaging based sentence vectors are predictive of the words that occur in adjacent sentences. In this way, the supervision signal that is exploited by models such as Skip-Thought can be exploited by averaging models as well. Similarly, Siamese CBOW [14] also focuses on learning sentence vectors that are good predictors for adjacent sentence vectors. Another line of work has focused on changing the nature of the averaging operation. For instance, [26] uses the family of power means in addition to the standard arithmetic average, the intuition being that by concatenating different averages different semantic properties are captured. Several approaches have also been proposed for computing weighted averages and for post-processing averaging based sentence vectors [3].

The use of n-grams in averaging based sentence embeddings has previously been considered as well. The DisC [2] model is most similar in spirit to our approach. They compute n-gram embeddings as the component-wise multiplication of the constituent word vectors. To represent sentences, they then concatenate averages of the bigram and trigram vectors that are thus obtained, together with the usual unigram averages. However, experimental results showed that the impact of including these n-gram averages on downstream applications was limited. Other approaches aim to learn semantically meaningful representations of frequent n-grams by learning them directly from co-occurrence statistics. Note that the word vectors of the constituent words are thus not used to compute the n-gram vectors. An example of such an approach is the Sent2Vec [21] model, which learns n-gram vectors that are optimized for predicting which sentence vectors include those embeddings.

2.2 Sentence Embeddings Capturing Word Order

While unigram averages clearly cannot capture any ordering information directly, it should be noted that they can still sometimes allow us to partially reconstruct the order in which words appear in the sentence, by exploiting regularities in natural language, e.g. the fact that certain words are more likely to appear at the start of a sentence than at the end. The extent to which unigram averages can capture information about word order in this way was analyzed in [1]. They found that while unigram averages can to some extend correctly classify whether the order of a word pair in a sentence is switched (70% accuracy), they lead to almost random predictions (51% accuracy) when the task is to classify sentences whose bigrams are switched [9]. The latter result in particular indicates that unigram averages are not helpful for reconstructing sentences.

The possibility of using n-gram averages in sentence embeddings was already considered by Arora et al. [2], who constructed n-gram vectors using component-wise multiplication of standard word vectors (cf. Section 2.2). However, they focused on the usefulness of such n-gram averages for measuring sentence similarity and did not consider the sentence reconstruction task. As an alternative to using n-grams, the ordinally forgetting strategy [30, 27] is also designed to capture word order based on averaging. Their representations are weighted averages of one-hot encodings, where the word at position i in a sentence is weighted by α^i for some constant $\alpha \in]0, 1[$. However, while it is possible to reconstruct the initial sentence from the resulting vector, due to the use of one-hot encodings, the dimensionality of this vector is prohibitively high. In particular, this earlier work therefore does not provide any real insights about how word order can be captured in neural network models such as LSTMs, which rely on dense vectors.

Finally, the popular transformer model [28] encodes positional information by manipulating word vectors based on their position in a sentence. While this manipulation is sufficient for capturing information about the position of words in deep networks based on transformers, it is unclear whether such a strategy could be adapted to work well with averaging based encodings. The main problem with this strategy, in our setting, is that for a sentence with 25 words, the number of candidate basis vectors to consider in the unigram reconstruction step would increase 25-fold. Given that the performance of the compressed sensing method crucially relies on the number of such basis vectors (see Section 4), a straightforward application of position encodings in the style of transformers would not be suitable.

3 Sentence Representation

In this section, we first discuss how sentences can be encoded using averages of unigram, bigram and trigram vectors (Section 3.1). Intuitively, these averages allow us to recover the bag-of-words, bag-of-bigrams and bag-of-trigrams representation of a given sentence. When it comes to capturing word order, the bag-of-bigrams representation plays a central role, with trigrams only needed to resolve some ambiguities. For this reason, in Section 3.2, we focus in particular on different strategies for encoding bigram vectors from the constituent word vectors. In Section 3.3 we then study the properties of these different bigram encodings, focusing on how well we can predict whether a given bigram vector is included in a bigram vector average. In Section 4 we will then show how well these representations allow us to recover the full sentence.

3.1 Averaging Based Sentence Vectors

Consider a sentence $S = w_1 w_2 \dots w_\ell$ and let $\mathbf{w_i}$ be the word vector for w_i. We write $U(S)$, $B(S)$ and $T(S)$ for the bags of unigrams, bigrams and trigrams from S, i.e., $U(S) = \{w_1, \dots, w_\ell\}$, $B(S) =$

$\{(w_1, w_2), (w_2, w_3), \ldots, (w_{\ell-1}, w_\ell)\}$ and $T(S) = \{(w_1, w_2, w_3), (w_2, w_3, w_4), \ldots, (w_{\ell-2}, w_{\ell-1}, w_\ell)\}$. We respectively write $\mathbf{S^n}$ for the n-gram based representation of S:

$$\mathbf{S^1} = \sum_{i=1}^{\ell} \mathbf{w_i} \qquad \mathbf{S^2} = \sum_{i=1}^{\ell-1} f(\mathbf{w_i}, \mathbf{w_{i+1}})$$

$$\mathbf{S^3} = \sum_{i=1}^{\ell-2} f(\mathbf{w_i}, \mathbf{w_{i+1}}, \mathbf{w_{i+2}})$$

for some n-gram encoding function f. Our main aim is to study sentence embeddings of the form $\mathbf{S^1} \oplus \mathbf{S^2} \oplus \mathbf{S^3}$, where we write $\oplus$ for vector concatenation. Since we can usually retrieve the bag-of-words representation of the sentence from $\mathbf{S^1}$, the main question is whether we can construct $\mathbf{S^2}$ and $\mathbf{S^3}$ such that the set of all bigrams and trigrams from S can be retrieved. Note that while including $\mathbf{S^1}$ may seem redundant, to effectively recover the bigrams from $\mathbf{S^2}$ we will first need to recover the bag-of-words representation from $\mathbf{S^1}$ (see Section 4). Intuitively, by recovering the bigrams we can already reconstruct a large part of the sentence. The following result clarifies under what conditions reconstructing the bigrams alone is sufficient.

Proposition 1. *Assume that (i) no word occurs three or more times in S, and (ii) at most one word occurs twice in S. Then it holds that S is uniquely determined by $B(S)$.*

Proof. First suppose that no word occurs more than once. Then we can uniquely determine the first word w_1 from the sentence, because this will be the only word that does not occur as the second argument in any of the elements from $B(S)$. Moreover, we can also uniquely determine the second word, as there will only be one element in $B(S)$ that has w_1 as the first argument. Continuing in this way, we can reconstruct the entire sentence. If no word occurs more than once, it is straightforward to see that $B(S)$ uniquely determines S. Now suppose there is a word a that is repeated, i.e., $S = w_1 \ldots w_{i-1} a w_{i+1} \ldots w_{j-1} a w_{j+1} \ldots w_\ell$. Similar as before, we can uniquely determine the sequence $w_1 \ldots a$. (Also in the case where a is the first word of the sentence). By symmetry, we can also uniquely determine the sequence $a w_{j+1} \ldots w_\ell$. In this way, we can uniquely determine w_{i+1} as the successor of the first occurrence of a, and complete the reconstruction of the sentence. □

As an example where the conditions of the proposition are not satisfied, given $B(S) = \{(a, b), (a, c), (b, a), (c, a)\}$ we could have $abaca$ or $acaba$. Such ambiguities can almost always be avoided by additionally considering trigrams, although in some cases also higher-order n-grams would be needed, i.e. if the same bigram is repeated three or more times in the sentence, or if there is more than one bigram that occur at least twice. However, as the empirical results will show, such situations are rare in practice, which is why we will not consider higher-order n-grams.

To encode trigrams, for simplicity we will only consider the choice $f(\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}) = \mathbf{w_1} \odot \mathbf{w_2} \odot \mathbf{w_3}$, as proposed by [2], where we write $\odot$ for the component-wise product. Note that this encoding does not capture in which order the three words appear, but since we only need the trigram averages to resolve ambiguities, this is sufficient for our purpose. We now turn to the question of how bigrams should be encoded.

3.2 Bigram Representation

We now discuss different strategies for encoding bigram vectors. One possibility is to again use the component-wise product, i.e. $f_\odot(\mathbf{w}, \mathbf{w}') = \mathbf{w} \odot \mathbf{w}' = (x_1 y_1, \ldots, x_d y_d)$, where $\mathbf{w} = (x_1, \ldots, x_d)$ and $\mathbf{w}' = (y_1, \ldots, y_d)$. Note that with this choice we cannot differentiate between a given sentence $w_1 \ldots w_\ell$ and its reverse $w_\ell \ldots w_1$, although this can be addressed by assuming that each sentence starts with a special token.

Another natural choice for representing bigrams is to use the vector difference, i.e. $f_{diff}(\mathbf{x}, \mathbf{y}) = \mathbf{y} - \mathbf{x}$. Clearly, however, such vectors are not suitable for averaging, as adding up the different bigram vectors would cancel most of the word vectors out, i.e. $\mathbf{S^2} = \mathbf{w_n} - \mathbf{w_1}$. To address this, we can apply a non-linear function such as $\tanh$ to the vector difference, i.e. $f_{tanh}(\mathbf{x}, \mathbf{y}) = \tanh(\mathbf{y} - \mathbf{x})$. While this choice does not suffer from the same limitation in theory, our ability to recover bigrams then crucially depends on the order of magnitude of the coordinates. To see why, note that by using the Taylor expansion of $\tanh$, given by $\tanh(x) \approx x - \frac{x^3}{3}$ for values close to 0, we know that the j^{th} coordinate of $\mathbf{S^2}$ is approximately equal to

$$(x_j^\ell - x_j^1) - \sum_{i=1}^{\ell-1} \frac{1}{3} (x_j^{i+1} - x_j^i)^3$$

If the coefficients of the word vectors are sufficiently small, $\mathbf{S^2}$ will thus still be dominated by $\mathbf{w}_\ell - \mathbf{w_1}$. To obtain a better bigram encoding, we can define f as $f_\lambda(\mathbf{w}, \mathbf{w}') = \tanh(\lambda(\mathbf{w}' - \mathbf{w}))$ for λ a sufficiently large constant. In the limit $\lambda \to +\infty$, this leads to $f_\infty(\mathbf{w}, \mathbf{w}') = \text{sgn}(\mathbf{w}' - \mathbf{w})$.

We will go one step further and combine the use of $\tanh$ with a learned linear transformation, i.e., we will use representations of the following form:

$$f_T(\mathbf{w}, \mathbf{w}') := \tanh(\mathbf{T}(\mathbf{w}' - \mathbf{w}) + \mathbf{b})$$

where we learn the matrix $\mathbf{T}$ and bias $\mathbf{b}$ in an unsupervised way from a given text corpus. Intuitively, we want to select $\mathbf{T}$ and $\mathbf{b}$ such that $\cos(\mathbf{S^2}, f_T(\mathbf{w}, \mathbf{w}'))$ is higher for bigrams (w,w') that occur in S than for other bigrams. To implement this, for each sentence S in our given text corpus $\mathcal{S}$ we randomly pick a positive bigram example from $B(S)$ and a negative bigram example from $(U(S) \times U(S)) \setminus B(S)$. Let us write pos_S and neg_S for the resulting bigram vectors obtained by applying f_T. Our objective is then to enforce that $\cos(\mathbf{S^2}, pos_S)$ is greater than $\cos(\mathbf{S^2}, neg_S)$ by some margin $\gamma > 0$. In particular, we learn $\mathbf{T}$ and $\mathbf{b}$ by minimizing the max-margin loss $\max(\cos(\mathbf{S^2}, neg_S) + \gamma - \cos(\mathbf{S^2}, pos_S), 0)$ for each sentence S in the text corpus $\mathcal{S}$.

3.3 Properties of Bigram Representations

Next, we study the properties of the aforementioned bigram encodings. As our main result, we show that the proposed encoding using f_T makes it easier to recover bigrams from a bigram vector average, and we provide some analysis to explain why that is the case.

Experimental Setting To empirically analyze the properties of the different bigram encodings, we use the pre-trained 300-dimensional fastText word vectors [5], with a vocabulary of two million words. These word vectors were trained on Common Crawl with 600 billion tokens [19]. The sentences that we used for training, validation and testing were obtained from an English Wikipedia dump. All sentences used in the evaluation consist of up to 25 words[6] that have corresponding fastText word vectors. The training, validation and test set consist

[6] This choice is based on the consideration that the average and median length of the sentences in the Wikipedia corpus are 25 and 23, respectively.

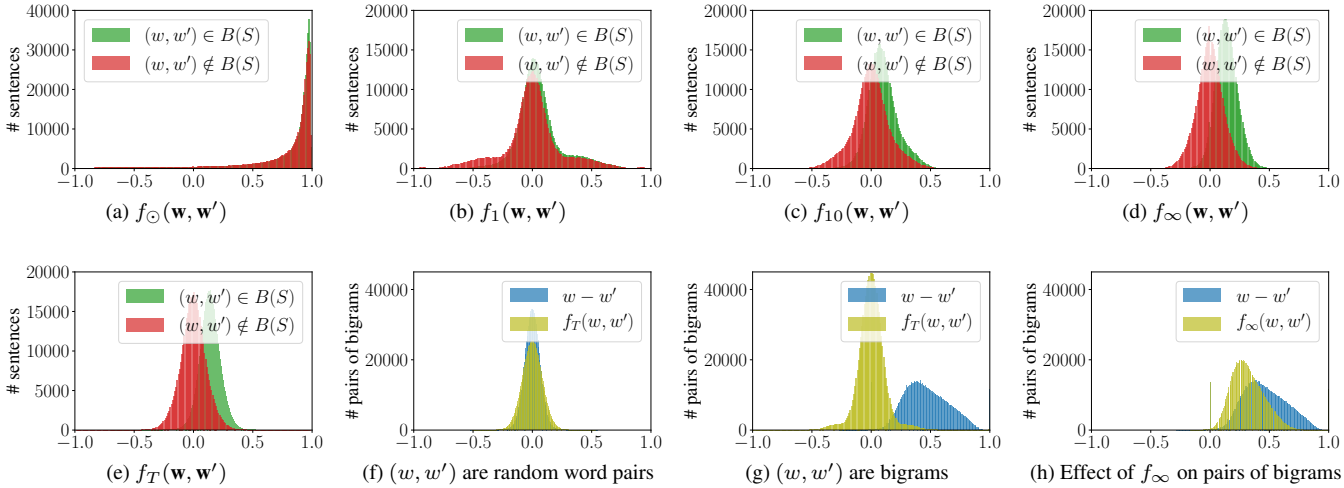


Figure 1. **(a)–(e)** Histograms of the cosine similarity $\cos(f(w, w'), \mathbf{S^2})$ between a sentence and word pair. **(f)–(h)** Histograms of the cosine similarity $\cos(f(w_1, w_2), f(w_3, w_4))$ between bigrams. (Best viewed in color.)

of 1,000,000, 10,000 and 10,000 sentences, respectively.[7] As an intrinsic evaluation of the bigram vectors, we consider the following problem: given the average $\mathbf{S^2}$ of all bigrams from a given sentence S and a bigram vector $f(\mathbf{w}, \mathbf{w}')$, decide whether that vector encodes a bigram which occurs in the sentence or not. The importance of this evaluation task stems from the fact that it is clearly a prerequisite to reconstructing the given sentence, while being independent of any particular sentence reconstruction method.

Results To compare the behavior of the different bigram encoding strategies, Figure 1 shows the histograms of the cosine similarity $\cos(f(\mathbf{w}, \mathbf{w}'), \mathbf{S^2})$ for both positive examples, i.e. pairs (w, w') from S, and negative examples. Figure 1a reveals a clear weakness of $\odot$ as a bigram encoding strategy, as the histograms for positive and negative examples are nearly identical in this case. This means that the similarity between $f(\mathbf{w}, \mathbf{w}')$ and $\mathbf{S^2}$ has very limited predictive value for deciding whether or not $(w, w') \in B(S)$. A similar issue can be observed in Figure 1b, which is due to the fact that the bigram vectors are approximately canceled out when using $\tanh(\mathbf{w}' - \mathbf{w})$, as explained above in Section 3.2. As Figure 1c reveals, multiplying the coefficients with a sufficiently large constant alleviates this issue. In the limit, when using the sign function, the differences between the histograms become even clearer, as can be seen in Figure 1d. Finally, Figure 1e shows that f_T also leads to clearly separated histograms.

To analyze the differences between the bigram encoding strategies quantitatively, in Table 1 we report the reconstruction accuracy, which is here defined as the percentage of sentences for which $\min_{(w,w') \in B(S)} \cos(f(\mathbf{w}, \mathbf{w}'), \mathbf{S^2}) > \cos(f(\mathbf{u}, \mathbf{u}'), \mathbf{S^2})$ where u and u' are randomly chosen words from $U(S)$ such that $(u, u') \notin B(S)$. In other words, we measure how often all of the bigram vectors $f(\mathbf{w}, \mathbf{w}')$ are more similar to $\mathbf{S^2}$ than the vector representation of a randomly sampled non-bigram word pair. The results are in accordance with our observations from the histograms, with $f_\odot$ and f_1 performing poorly (achieving a reconstruction accuracy of only 5% and 12% respectively). Furthermore, we can see that f_T achieves the best results overall.

[7] The model and the training procedure for bigram encoding f_T is implemented using the PyTorch Python library [23].

Table 1. Summary of the performance of different bigram. The accuracy measures how often a bigram in a sentence is closer to the sentence than all other word pairs that are not bigrams in the sentence.

Bigram encoding		Acc.
$f_\odot(\mathbf{w}, \mathbf{w}')$	$= \mathbf{w}^T \odot \mathbf{w}'$	5%
$f_1(\mathbf{w}, \mathbf{w}')$	$= \tanh(\mathbf{w}' - \mathbf{w})$	12%
$f_{10}(\mathbf{w}, \mathbf{w}')$	$= \tanh(10 \cdot (\mathbf{w}' - \mathbf{w}))$	31%
$f_\infty(\mathbf{w}, \mathbf{w}')$	$= \text{sgn}(\mathbf{w}' - \mathbf{w})$	42%
$f_T(\mathbf{w}, \mathbf{w}')$	$= \tanh(\mathbf{T}(\mathbf{w}' - \mathbf{w}) + \mathbf{b})$	46%

Analysis An interesting question is why the bigram vectors obtained with f_T lead to a better recovery accuracy than the other encodings in Figure 1. To analyze this, let $\mathbf{b}_1, ..., \mathbf{b}_{n-1}$ be the vector representations of the bigrams in $B(S)$. Let $i \in \{1, ..., n-1\}$ and let $\mathbf{b}_*$ be the vector representation of a bigram which is not in $B(S)$. Assume for simplicity that the norm $\|\mathbf{b}\|$ of all bigram vectors is approximately equal to some constant η. Then we have

$$\cos(\mathbf{b_i}, \mathbf{S^2}) \propto \eta^2 + \textstyle\sum_{j \neq i} \mathbf{b_i} \cdot \mathbf{b_j}$$
$$\cos(\mathbf{b}_*, \mathbf{S^2}) \propto \mathbf{b}_* \cdot \mathbf{b_i} + \textstyle\sum_{j \neq i} \mathbf{b}_* \cdot \mathbf{b_j}$$

If all bigram vectors are more or less uniformly distributed, we can expect the variance of the dot products $\mathbf{b_i} \cdot \mathbf{b_j}$ and $\mathbf{b}_* \cdot \mathbf{b_j}$ to be relatively low, and in particular we can expect $\sum_{j \neq i} \mathbf{b_i} \cdot \mathbf{b_j} \approx \sum_{j \neq i} \mathbf{b}_* \cdot \mathbf{b_j}$. Since $\mathbf{b}_* \cdot \mathbf{b_i} < \eta^2$ if $\mathbf{b}_* \neq \mathbf{b_i}$, we can thus expect $\cos(\mathbf{b_i}, \mathbf{S^2}) > \cos(\mathbf{b}_*, \mathbf{S^2})$.

This implies that suitable bigram vectors should (i) have approximately the same norm and (ii) be distributed as uniformly as possible. The former condition is satisfied because of our use of $\tanh$, which means that most coordinates will be close to -1 or 1, and thus that the norm of the bigram vectors will be close to square root of their number of dimensions. To investigate the second condition, in Figures 1f and 1g we compare the histograms of the cosine similarities between the vector encodings of randomly chosen bigrams (w_1, w_2) and (w_3, w_4), when using f_T and when using f_{diff}. The results in Figure 1f are for pairs of randomly chosen unigrams, while those in 1g are for actual bigrams from the Wikipedia corpus we used in our experiments. These figures suggest that while word vector differences

Algorithm 1: Sentence reconstruction

Input : Sentence vector $\mathbf{S} = \mathbf{S^1} \oplus \cdots \oplus \mathbf{S^n}$,
Word vector matrix $\mathbf{W}$,
n-gram encodings f^n.
Output : A set of candidate sentences S

$V_1 \leftarrow V$
$M_1 \leftarrow \textsc{Reconstruct}(\mathbf{S^1}, \mathbf{W})$
for $k \leftarrow 2$ **to** n **do**
  $V_k \leftarrow \textsc{Vocab}(M_{k-1})$
  $\mathbf{W}_k \leftarrow \textsc{GetMatrix}(f^k, \mathbf{W}, V_k)$
  $M_k \leftarrow \textsc{Reconstruct}(\mathbf{S^k}, \mathbf{W}_k)$
$\mathcal{S} \leftarrow \textsc{CandidateSentences}(M_n)$
return $\mathcal{S}$

are distributed more or less uniformly for random word pairs, they follow a very different distribution for actual bigrams. In the case of f_T, on the other hand, in both cases the bigram vectors are more or less uniformly distributed. Figure 1h shows the result for f_∞ (for the bigrams from Wikipedia), showing that f_∞ behaves similarly to the word vector differences in this respect. These results suggest that the improved reconstruction accuracy of the f_T encodings comes from the fact that the resulting bigram vectors are distributed more uniformly.

4 Sentence Reconstruction

The aim of this section is to compare our sentence vectors of the form $\mathbf{S^1} \oplus \mathbf{S^2} \oplus \mathbf{S^3}$ with those that are learned by LSTM autoencoders, on the task of sentence reconstruction. In other words, we evaluate both representations in terms of how well they capture the words from the given sentence, and the order in which these words appear. Before we present the empirical evaluation, we first address the question of how we can reconstruct the sentence S from its embedding $\mathbf{S^1} \oplus \mathbf{S^2} \oplus \mathbf{S^3}$. In Section 4.1, we review the compressed sensing strategy that was used by [2] to recover the set of unigrams from the vector $\mathbf{S^1}$. However, the success of this strategy crucially depends on the number of candidate words, which means in particular that it cannot directly be used to recover bigrams and trigrams. In Section 4.2 we discuss a solution to this issue, and explain our overall sentence reconstruction strategy. Finally, in Section 4.3, we present the empirical comparison with LSTM autoencoders. We find that our method compares favorably to LSTMs, showing that the process of averaging n-gram vectors is indeed sufficient, and in fact surprisingly effective, for constructing order-encoding sentence vectors.

4.1 Unigram Reconstruction

In the following, d denotes the dimension of the word vectors and V the vocabulary. Let $\mathbf{W}$ be a $d \times |V|$ matrix having word vectors as its columns and let $\mathbf{e}_w$ be the one-hot vector representation of word w with respect to $\mathbf{W}$, i.e., $\mathbf{w} = \mathbf{W}\mathbf{e}_w$. Then, given a sentence $S = w_1 w_2 \dots w_\ell$ its unigram vector representation $\mathbf{S}$ can be written as $\mathbf{S} = \sum_{i=1}^{\ell} \mathbf{w}_i = \sum_{i=1}^{\ell} \mathbf{W} \cdot \mathbf{e}_{w_i} = \mathbf{W} \cdot (\sum_{i=1}^{\ell} \mathbf{e}_{w_i})$.

The equation $\mathbf{S} = \mathbf{W} \cdot (\sum_{i=1}^{\ell} \mathbf{e}_{w_i})$ lets us interpret the problem of reconstructing unigrams from $\mathbf{S}$ as a *compressed sensing* [6, 11] problem. The goal of compressed sensing is to recover a high-dimensional signal (which is supposed to be sparse) from few linear measurements. In our case, $\mathbf{S}$ is the measurement, $\mathbf{W}$ the measurement matrix, and $\sum_{i=1}^{\ell} \mathbf{e}_{w_i}$ is the signal. Two methods are commonly used to tackle this compressed sensing problem. The first one is called *basis pursuit* (BP) [7], which tries to solve the equation subject to the L_1-norm of the signal and *orthogonal matching pursuit* (OMP) [17], which uses a greedy approach to choose a basis vector, which is in our case the one-hot vector, at each iteration.

4.2 Sentence Reconstruction

To reconstruct n-grams from the n-gram based average $\mathbf{S^n} = \sum_{i=1}^{\ell-n+1} f(\mathbf{w}_i, \dots, \mathbf{w}_{i+n-1})$ of a sentence S with length ℓ, where f is an n-gram encoding function, we can again use compressed sensing. The idea here is to replace the word vector matrix $\mathbf{W}$ with a matrix $\mathbf{W}_n$ that consists of unique column vectors $f(w_{i_1}, \dots, w_{i_n})$ with $(w_{i_1}, \dots, w_{i_n}) \in V^n$. In this way, there is a one-to-one correspondence between the column vectors of $\mathbf{W}_n$ and the n-grams from V^n. We note that the size of V^n grows exponentially with n. Because the reconstruction performance decreases with an increasing number of candidate n-grams, we need prior information that can reduce the number of candidates for successfully recovering n-grams. For this reason, a single n-gram based sentence encoding $\mathbf{S^n}$ is rarely useful for reconstructing the original sentence: if n is small then there are usually many possibilities to form a sentence from the n-grams, and if n is large, then there are too many candidate n-grams, which decreases the performance of compressed sensing. To overcome this, we will rely on a concatenation of n-gram encodings $\mathbf{S^1}, \dots, \mathbf{S^n}$ and reconstruct $\mathbf{S^k}$ from $\mathbf{S^{k-1}}$ in an iterative manner. This strategy is summarized in Algorithm 1.

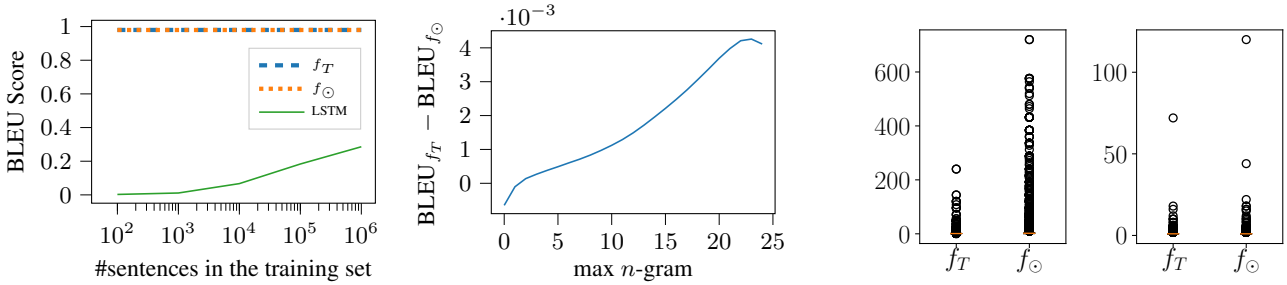
The algorithm first reconstructs a bag (i.e. multiset) M_1 of unigrams from vector $\mathbf{S^1}$ by means of compressed sensing, where we restrict the measurement matrix to word vectors appearing in vocabulary V_1. This step is repeated with k-gram encodings $\mathbf{S^k}$ for $k = 2, \dots, n-1$. To this end, the algorithm first generates a vocabulary V_k consisting of all k-grams that can be constructed by combining two compatible $(k-1)$-grams from the bag M_{k-1}. For example, the k-gram $a_1, ..., a_k$ would be included iff M_{k-1} contains both $a_1, ..., a_{k-1}$ and $a_2, ..., a_k$. Subsequently, a measurement matrix $\mathbf{W}_k$ is constructed, consisting of the vector encodings $f(\mathbf{a_1}, ..., \mathbf{a_k})$ of all the k-grams $a_1, ..., a_k$ from V_k. Finally, once the bag of n-grams M_n has been found, the algorithm generates a set of candidate sentences that can be formed from the n-grams in M_n and returns the result as its output. Note that the output is not always a singleton set, because there is sometimes more than one possibility to form a sentence from the given n-grams.

We apply this strategy in particular to $\mathbf{S^1} \oplus \mathbf{S^2} \oplus \mathbf{S^3}$, where f_T is used to construct $\mathbf{S^2}$ and $f_\odot$ is used to construct $\mathbf{S^3}$. Note that by encoding trigrams using $f_\odot$, the ordering of the words within a trigram is not captured. However, since the candidate trigrams in V_3 are ordered, we can still recover ordered trigrams from $\mathbf{S^3}$.

4.3 Empirical Comparison of Sentence Reconstruction Methods

We consider the task of sentence reconstruction, where we compare the method from Algorithm 1 with LSTM autoencoders.

Experimental setting Based on the training setting mentioned in Section 3.3, we consider as sentence encoding a concatenation of unigram, bigram and trigram encodings, i.e, $\mathbf{S} = \mathbf{S^1} \oplus \mathbf{S^2} \oplus \mathbf{S^3}$. For comparison purposes, in addition to our proposed bigram encoding f_T, we will also show results for bigram encodings of the form $f_\odot$. To allow the latter strategy to recover the sentence, we prepend and append to each sentence a start and an end token,



(a) Corpus-level BLEU score (b) BLEU$_{f_T}$ − BLEU$_{f_\odot}$

Figure 2. (a) BLEU score of the reconstructed test sentences. (b) BLEU score difference between f_T and $f_\odot$ with varying maximal length of n-grams considered in the BLEU score.

(a) $\mathbf{S^2}$ (b) $\mathbf{S^3}$

Figure 3. Number of candidate sentences reconstructed from $\mathbf{S^2}$ and $\mathbf{S^3}$.

respectively. Algorithm 1 is applied to reconstruct the sentence S from these vector representations. For reconstructing n-grams, we use BP for unigrams and OMP[8] for bi- and trigrams.

We compare the performance of our approach with an LSTM [13] autoencoder architecture with single-layer encoder and decoder. The LSTM encoder reads the input sequence and encodes it as a concatenation of the last hidden and cell states. This state vector is then passed to the decoder, which is trained with a teacher forcing policy. We used a standard implementation of this LSTM autoencoder, which is part of the Keras deep learning library [8]. For inference, this model encodes the input sequence as a state vector, and passes it alongside the special start token to a softmax layer for predicting the next word. This action is performed iteratively until the end token is predicted, or the maximium sentence length is reached. The RMSprop optimizer is used with a learning rate of 0.001 with no decay for a maximum of 50 epochs, and an early stopping strategy based on accuracy on the development set.

Evaluation metrics Our first evaluation metric is simply the percentage of perfectly reconstructed sentences. In order to take into account the magnitude of the error in cases where the reconstruction is not perfect, we also evaluate the reconstruction performance with the BLEU score [22], which is commonly used in the area of machine translation. For our experiments we use the `corpus_bleu` module from the NLTK python library [4]. By default, the BLEU score considers n-grams from $n = 1$ up to $n = 4$, but we will also show results for BLEU scores that consider n-grams with other ranges.

Results Table 2 shows the percentage of sentences which were perfectly reconstructed by each model. Interestingly, both averaging based approaches perform surprisingly well, our proposed f_T encoding leading to the best results (97.9%). The computationally much more expensive LSTM autoencoder could not reconstruct any of the sentences in the test set perfectly. The table also shows the percentage of sentences for which the sets of unigrams, bigrams and trigrams were correctly recovered. Note that when the unigrams cannot be correctly recovered, then the bigrams cannot be recovered either. It is thus interesting to see that with f_T the accuracy for bigrams is approximately the same as the accuracy for unigrams. In contrast, when using $f_\odot$ some additional errors are introduced at the bigram recovery step.[9] Finally, recovering trigrams is clearly harder than recovering bigrams and unigrams.

Table 2. The percentage of correctly reconstructed sentences (resp. n-grams) from the test set. For f_T the bigrams are ordered, whereas the bigrams for $f_\odot$ are unordered.

	Sentence	Tri	Bi	Uni
f_T	**97.9**	**97.9**	**99.8**	**99.8**
$f_\odot$	96.2	97.4	97.7	**99.8**
LSTM	0.0	N/A	N/A	N/A

The evaluation in terms of the standard BLEU score is presented in Figure 2a, where the size of training data is considered as a parameter. As can be seen, the averaging based methods perform well independent of the training data size, whereas the performance of the LSTM autoencoder increases with more data, which, however, does not even go over BLEU score 0.4. The difference between $f_\odot$ and f_T is small, but as shown in Figure 2b, f_T consistently achieves the best score, when the maximal length of n-grams considered for the BLEU score is varied from 1 to 25. One of the main reasons why f_T outperforms $f_\odot$ is because f_T captures the ordering of the two words within a bigram, which is not the case for $f_\odot$. The set of unordered bigrams that is recovered when $f_\odot$ is used thus leaves more flexibility when reconstructing the sentence, which means that the resulting set of candidate sentences is larger. This is shown in Figure 3a. After the trigrams have been recovered as well, some of this ambiguity is eliminated. However, as shown in Figure 3b, the number of compatible sentences remains larger on average for the case where bigrams are encoded using $f_\odot$.

5 LSTM Encoding

In this section, we show that LSTM based architectures are able to construct sentence vectors of the form $\mathbf{S^1} \oplus \mathbf{S^2} \oplus \mathbf{S^3}$. This generalizes a result from [2], where it was shown that LSTMs can construct unigram averages. We will assume that bigrams are encoded using f_T, as this choice leads to the best overall results in the empirical evaluation in Section 4.3. However, it is straightforward to adapt the proposed encoding to the other choices we considered for f, with the

[8] For BP we re-implemented the implementation in `https://github.com/NLPrinceton/sparse_recovery` using the CuPy Python libary [20]. For OMP we use the OrthogonalMatchingPursuit module in the Scikit-learn Python library [24].

[9] Note that while $f_\odot$ captures *unordered* bigrams, this does not turn out to be a major limitation, as in most cases we can recover the ordering because of the introduction of the start and end tokens as well as the help of the reconstructed unordered trigrams.

exception of f_∞. We consider an LSTM architecture which processes the sentence from left to right. At each position i in the sentence, we consider LSTM units of the following form:

$$\begin{aligned}
\mathbf{f_i} &= \phi_f(\mathbf{W_f w_i} + \mathbf{U_f h_{i-1}} + \mathbf{b_f}) \\
\mathbf{p_i} &= \phi_p(\mathbf{W_p w_i} + \mathbf{U_p h_{i-1}} + \mathbf{b_p}) \\
\mathbf{o_i} &= \phi_o(\mathbf{W_o w_i} + \mathbf{U_o h_{i-1}} + \mathbf{b_o}) \\
\mathbf{c_i} &= (\mathbf{f_i} \odot \mathbf{c_{i-1}}) + (\mathbf{p_i} \odot \phi_c(\mathbf{W_c w_i} + \mathbf{U_c h_{i-1}} + \mathbf{b_c})) \\
\mathbf{h_i} &= \mathbf{o_i} \odot \phi_h(\mathbf{c_i})
\end{aligned}$$

where $\mathbf{w_i}$ is the vector representation of the word at position i, as before. Let us introduce the following abbreviations:

$$\mathbf{b_i} = f_T(\mathbf{w_{i-1}}, \mathbf{w_i}) \qquad \mathbf{a_i^{\mathit{uni}}} = \sum_{l=0}^{i} \mathbf{w_l}$$

$$\mathbf{t_i^2} = f_\odot(\mathbf{w_i}, \mathbf{w_{i-1}}) \qquad \mathbf{a_i^{\mathit{bi}}} = \sum_{l=1}^{i} f_T(\mathbf{w_{l-1}}, \mathbf{w_l})$$

$$\mathbf{t_i^3} = f_\odot(\mathbf{w_{i-2}}, \mathbf{w_{i-1}}, \mathbf{w_i}) \qquad \mathbf{a_i^{\mathit{tri}}} = \sum_{l=2}^{i} f_\odot(\mathbf{w_{l-2}}, \mathbf{w_{l-1}}, \mathbf{w_l})$$

with $\mathbf{w_0}$ an arbitrary but fixed start token. The following result shows that we can choose the parameters of the LSTM unit such that at each position i it outputs a vector encoding the part of the sentence to the left of i, along with some other vectors which are needed for bookkeeping.

Theorem 1. *We can choose the parameters of the LSTM unit such that*

$$\mathbf{h_i} = \mathbf{w_i} \oplus \mathbf{b_i} \oplus \mathbf{t_i^2} \oplus \mathbf{t_i^3} \oplus \mathbf{a_{i-1}^{\mathit{uni}}} \oplus \mathbf{a_{i-1}^{\mathit{bi}}} \oplus \mathbf{a_{i-1}^{\mathit{tri}}}$$

Proof. We will choose the parameters of the LSTM unit such that at position i it holds that:

$$\mathbf{c_i} = \mathbf{w_i} \oplus \mathbf{b_i} \oplus \mathbf{w_{i-1}} \oplus \mathbf{t_{i-1}^2} \oplus \mathbf{a_{i-1}^{\mathit{uni}}} \oplus \mathbf{a_{i-1}^{\mathit{bi}}} \oplus \mathbf{a_{i-1}^{\mathit{tri}}}$$

and

$$\mathbf{h_i} = \mathbf{w_i} \oplus \mathbf{b_i} \oplus \mathbf{t_i^2} \oplus \mathbf{t_i^3} \oplus \mathbf{a_{i-1}^{\mathit{uni}}} \oplus \mathbf{a_{i-1}^{\mathit{bi}}} \oplus \mathbf{a_{i-1}^{\mathit{tri}}}'$$

where the error terms in these approximations can be made arbitrarily small.

1. We choose ϕ_f as the linear activation function, and choose the parameters $\mathbf{W_f}$, $\mathbf{U_f}$ and $\mathbf{b_f}$ such that:

   $$\mathbf{f_i} = \mathbf{0}_{4d} \oplus \mathbf{1}_{3d}$$

   where we write $\mathbf{0}_n$ for the n-dimensional vector containing a 0 at every position, and $\mathbf{1}_n$ for the n-dimensional vector containing a 1 at every position. Furthermore, we write d for the dimension of the considered word vectors.
2. We choose $\phi_p = \tanh$ and choose $\mathbf{W_p}$, $\mathbf{U_p}$ and $\mathbf{b_p}$ such that:

   $$\mathbf{p_i} = \frac{1}{2} \cdot (\mathbf{1}_d \oplus \mathbf{b_i} \oplus \mathbf{1}_{5d})$$

   The reason why we have to multiply the coordinates by $\frac{1}{2}$ is because the $\tanh$ activation function cannot output 1 (although it can output a value which is arbitrarily close to it).
3. We also choose the linear activation for ϕ_c and choose $\mathbf{W_c}$, $\mathbf{U_c}$ and $\mathbf{b_c}$ such that $\mathbf{W_c w_i} + \mathbf{U_c h_{i-1}} + \mathbf{b_c}$ is given by

   $$2 \cdot (\mathbf{w_i} \oplus \mathbf{1}_d \oplus \mathbf{w_{i-1}} \oplus \mathbf{t_{i-1}^2} \oplus \mathbf{w_{i-1}} \oplus \mathbf{b_{i-1}} \oplus \mathbf{t_{i-1}^3})$$

   Note that we multiply all the coordinates by 2 here to offset the fact that we had to halve the coordinates of $\mathbf{p_i}$.
4. We choose a linear activation for ϕ_o and choose the parameters $\mathbf{W_o}$, $\mathbf{U_o}$ and $\mathbf{b_o}$ such that

   $$\mathbf{o_i} = \mathbf{1}_{2d} \oplus \mathbf{w_i} \oplus \mathbf{w_i} \oplus \mathbf{1}_{3d}$$

5. Finally, we choose a linear activation for ϕ_h. For the initialization we define:

   $$\mathbf{b_0} = 0_d, \quad \mathbf{t_0^2} = \mathbf{0}_d, \quad \mathbf{t_0^3} = \mathbf{0}_d$$

   and set the initial values for $\mathbf{c_0}$ and $\mathbf{h_0}$:

   $$\mathbf{c}_0 := \mathbf{0}_{7d}, \quad \mathbf{h_0} := \mathbf{w_0} \oplus \mathbf{0}_{6d}.$$

Then, one can verify that the parameters can be chosen in these ways, and that these choices lead to the desired behavior. □

6 Conclusions and Future Work

Our motivation in this paper was to elucidate why the relatively simple manipulation of word vectors which is done by e.g. LSTM encoders is sufficient for learning sentence vectors from which the exact sentence can be recovered. To this end, we have provided an analysis of how word ordering can be captured in averaging based sentence embeddings. Specifically, we considered sentence representations which consist of the concatenation of the usual unigram average with bigram and trigram vector averages. To encode bigrams, we rely on a learned transformation of the word vector difference, which we found to substantially outperform common alternatives on a bigram reconstruction task. We then showed that the considered sentence representations compare favorably to LSTM-based sentence autoencoders on the challenging task of reconstructing a given sentence from its vector encoding. While the exact strategy used by LSTMs in practice may be different, the fact that order-encoding embeddings can be obtained by simple averaging shows why we can indeed expect LSTMs to capture word order, in tasks where this is important.

In terms of future work, the main unanswered question is how sentence embeddings based on unigram averaging can be enriched in a way that is useful for downstream applications. The simple strategy of concatenating bigram and trigram averages, which we studied in this paper, does not on its own lead to clear improvements in downstream tasks such as measuring sentence similarity, as was already shown in [2]. However, we believe that the bigram encoding strategy that we introduced in this paper would be useful as a building block for generating semantically richer sentence embeddings. In particular, if the aim is to learn semantically meaningful sentence embeddings, rather than order-encoding ones, we believe three changes are needed. First, our bigram encoding network now only considers the task of bigram reconstruction for learning bigram vectors. This could be improved by including a component in the loss function that encourages bigram vectors to be predictive of the next sentence, for instance. Second, clearly not all bigrams add relevant semantic information, which suggests that a weighted averaging strategy could lead to semantically more informative sentence vectors. Finally, in addition to bigrams (and longer n-grams), it may sometimes be more useful to include embeddings of skip-grams, i.e. pairs of non-consecutive words from the sentence. To learn suitable weighted averages of such skipgram vectors, we could again rely on the intuition that sentence vectors should be predictive of adjacent sentences. Interestingly, this strategy of learning weighted skip-gram averages bears a striking resemblance to the popular transformer architecture, which will also be explored in future work.

Acknowledgement This work has been supported by ERC Starting Grant 637277.

REFERENCES

[1] Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg, 'Fine-grained analysis of sentence embeddings using auxiliary prediction tasks', (2017).

[2] Sanjeev Arora, Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli, 'A compressed sensing view of unsupervised text embeddings, bag-of-n-grams, and LSTMs', in *Proceedings of the 6th International Conference on Learning Representations*, (2018).

[3] Sanjeev Arora, Yingyu Liang, and Tengyu Ma, 'A simple but tough-to-beat baseline for sentence embeddings', in *Proceedings of the International Conference on Learning Representations*, (2017).

[4] Steven Bird, Ewan Klein, and Edward Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O'Reilly Media, Beijing ; Cambridge Mass., 1 edition edn., July 2009.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov, 'Enriching word vectors with subword information', *Transactions of the Association for Computational Linguistics*, **5**, (2017).

[6] E. J. Candes and T. Tao, 'Decoding by linear programming', *IEEE Transactions on Information Theory*, **51**(12), 4203–4215, (December 2005).

[7] Shaobing Chen and D. Donoho, 'Basis pursuit', in *Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers*, volume 1, pp. 41–44 vol.1, (October 1994).

[8] François Chollet et al. Keras. `https://keras.io`, 2015.

[9] Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni, 'What you can cram into a single vector: Probing sentence embeddings for linguistic properties', in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2126–2136. Association for Computational Linguistics, (2018).

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 'BERT: Pre-training of deep bidirectional transformers for language understanding', in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, (June 2019). Association for Computational Linguistics.

[11] Simon Foucart and Holger Rauhut, *A Mathematical Introduction to Compressive Sensing*, Applied and Numerical Harmonic Analysis, Birkhäuser Basel, 2013.

[12] Felix Hill, Kyunghyun Cho, and Anna Korhonen, 'Learning distributed representations of sentences from unlabelled data', in *Proceeding of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1367–1377, (2016).

[13] Sepp Hochreiter and Jürgen Schmidhuber, 'Long short-term memory', *Neural computation*, **9**(8), 1735–1780, (1997).

[14] Tom Kenter, Alexey Borisov, and Maarten de Rijke, 'Siamese CBOW: optimizing word embeddings for sentence representations', in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, (2016).

[15] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler, 'Skip-thought vectors', in *Advances in neural information processing systems*, pp. 3294–3302, (2015).

[16] Lajanugen Logeswaran and Honglak Lee, 'An efficient framework for learning sentence representations', in *International Conference on Learning Representations*, (2018).

[17] S.G. Mallat and Zhifeng Zhang, 'Matching pursuits with time-frequency dictionaries', *IEEE Transactions on Signal Processing*, **41**(12), 3397–3415, (December 1993).

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 'Efficient estimation of word representations in vector space', in *Proceedings of the International Conference on Learning Representations*, (2013).

[19] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin, 'Advances in pre-training distributed word representations', in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, (2018).

[20] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis, 'Cupy: A numpy-compatible library for nvidia gpu calculations', in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, (2017).

[21] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi, 'Unsupervised learning of sentence embeddings using compositional n-gram features', in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 528–540, (2018).

[22] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu, 'BLEU: a method for automatic evaluation of machine translation', in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, p. 311, Philadelphia, Pennsylvania, (2001). Association for Computational Linguistics.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, 'Automatic differentiation in PyTorch', in *NIPS Autodiff Workshop*, (2017).

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

[25] Nils Reimers and Iryna Gurevych, 'Sentence-BERT: Sentence embeddings using Siamese BERT-networks', in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3973–3983, Hong Kong, China, (November 2019). Association for Computational Linguistics.

[26] Andreas Rücklé, Steffen Eger, Maxime Peyrard, and Iryna Gurevych, 'Concatenated p-mean word embeddings as universal cross-lingual sentence representations', *CoRR*, **abs/1803.01400**, (2018).

[27] Joseph Sanu, Mingbin Xu, Hui Jiang, and Quan Liu, 'Word Embeddings based on Fixed-Size Ordinally Forgetting Encoding', in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 310–315, Copenhagen, Denmark, (2017). Association for Computational Linguistics.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, 'Attention is all you need', in *Advances in neural information processing systems*, pp. 5998–6008, (2017).

[29] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu, 'Towards universal paraphrastic sentence embeddings', in *Proceedings of the International Conference on Learning Representations*, (2016).

[30] Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, and Lirong Dai, 'The fixed-size ordinally-forgetting encoding method for neural network language models', in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pp. 495–500, (2015).