Sequential Sentence Embeddings for Semantic Similarity

Antonio Carta
Dipartimento di Informatica
Università di Pisa
Email: antonio.carta@di.unipi.it

Davide Bacciu
Dipartimento di Informatica
Università di Pisa
Email: bacciu@di.unipi.it

***Abstract*—Sentence embeddings are distributed representations of sentences intended to be general features to be effectively used as input for deep learning models across different natural language processing tasks. State-of-the-art sentence embeddings for semantic similarity are computed with a weighted average of pretrained word embeddings, hence completely ignoring the contribution of word ordering within a sentence in defining its semantics. We propose a novel approach to compute sentence embeddings for semantic similarity that exploits a linear autoencoder for sequences. The method can be trained in closed form and it is easy to fit on unlabeled sentences. Our method provides a grounded approach to identify and subtract common discourse from a sentence and its embedding, to remove associated uninformative features. Unlike similar methods in the literature (e.g. the popular Smooth Inverse Frequency approach), our method is able to account for word order. We show that our estimate of the common discourse vector improves the results on two different semantic similarity benchmarks when compared to related approaches from the literature.**

***Keywords*—sentence embeddings; linear autoencoders; semantic similarity**

I. Introduction

According to the distributional hypothesis words that occur in the same context tend to have the same meaning [1], [2]. Following this hypothesis, several methods have been developed to compute a distributed representation of a word by exploiting the context in which it appears[3], [4]. After training, these representations can be used as input features for a learning model trained to solve specific Natural Language Processing (NLP) tasks.

Most machine learning models accept only fixed-length vectors as input. Therefore, it is interesting to study techniques to project variable-length sentences into vectors of fixed size. In this paper, we focus on unsupervised embedding methods for semantic similarity. Unsupervised methods can exploit large corpora of unlabeled data, which are especially useful for languages where the amount of labeled data is limited and purely supervised methods might generalize poorly to new domains.

A simple and straightforward solution, popularized by [5], is to compute the sentence embedding as the average of the composing words' embeddings. The Smooth Inverse Frequency (SIF) approach [6] improves this method with two modifications. First, each word embedding is weighted with a coefficient dependent on its frequency. Second, a common discourse vector, estimated as the principal direction of the weighted averages of all the sentences, is removed from the final embedding. The common discourse is subtracted to remove shared components which do not contribute to the semantic content of the sentence but are only related to the generic structure of a sentence in the source language. The SIF approach has become widely popular thanks to both its simplicity and efficacy in semantic similarity tasks. On the other hand, it founds on a strong simplifying assumption, which is that of completely ignoring the contribution of word ordering in defining the interpretation of the sentence.

In this paper, we define and empirically assess the efficacy of alternative aggregation methods to compute sentence embeddings that can correctly account for word order. To this end, we put forward the use of a linear autoencoder for sequences [7], which is an autoencoder able to memorize and reconstruct sequential data. We show how the representation learned by an autoencoder trained on unlabeled sentences can be used to compute sentence embeddings. The proposed approach only requires the user to select the size of the hidden state for the autoencoder.

In addition, we study the impact of removing a common discourse vector from a weighted average of word embeddings. In particular, we propose a novel and general approach to disentangle common discourse from the sentence embeddings using the same autoencoder model used to generate it. In particular, we propose to isolate the effect of common discourse by training a linear autoencoder for sequences to reconstruct the original sentences while being restricted to use only a small number of hidden neurons. In our intuition, such constraint should make the autoencoder unable to reconstruct the original sequence except for the common discourse. Note that, unlike SIF, our method is able to account for the sequential structure of the sentence. The experimental results with this approach show an improvement with respect to SIF on two different semantic similarity benchmarks. We argue that this improvement is caused by the ability to capture relevant semantic properties dependent on the word order. Furthermore, while the authors of SIF assume that removing only the principal component is sufficient to estimate the common discourse, our experimental results show that such an assumption does not hold in practice. Our finding is in contrast with previous work[6], [8] and justifies more sophisticated methods to estimate the common discourse.

Summarizing, the main contributions of the paper are:

- we introduce, for the first time, a linear model for sentence embeddings that can effectively account for the effect of word order;
- we define a novel algorithm to extract the common discourse vector from a set of sentences through a robust and general approach;
- we experimentally study the effect of common discourse removal on two semantic similarity datasets.

II. Related Work

Several approaches have been developed to represent words and sentences with continuous vector representations. Language models based on neural networks [9], [10] allow to model the language generation with a shallow feedforward network. These networks learn distributed word representations that can be used as features for other downstream tasks.

The word2vec model [3], [11] provides an efficient method to compute word embeddings based on neural language models. Word embeddings represent several semantic and syntactic relationships with linear relationships [3], i.e. displacement vectors. GloVe [4] embeddings are based on matrix factorizations. Like word2vec, GloVe embeddings represents several word properties as linear relationships. Later, [12] has shown that the two approaches are related since they solve the same matrix factorization problem on the Pointwise Mutual Information (PMI) matrix.

Several approaches in the literature exploit neural networks to compute sentence embeddings, like Skip-Thought vectors [13], recursive neural networks [14], [15], [16], [17], convolutional neural networks [18] and attention models [19]. However, [5] shows that most of these methods are outperformed by a simple average of word embeddings in out-of-domain scenarios, and therefore their representations cannot be used as general features in separate tasks. Subsequent work was able to train an LSTM [20] and a novel architecture, dubbed GRAN[21], which is a combination between an LSTM and an averaging model. By making the network focus less on the word order, the authors are able to outperform both the average model and the LSTM. More recently, [22] trained a variant of an LSTM on the Stanford Natural Language Inference datasets and showed that the learned representations can be used as general features for a wide range of NLP tasks. Unfortunately, this approach requires access to a large high-quality labeled dataset for natural language inference, which is not available for most languages. In [23], it is defined a model inspired by word2vec and based on k-grams representations.

SIF [6] computes sentence embeddings with a weighted average followed by a principal component removal. Their approach is justified by a language model [24], [25], improved in [6], where the generation process is guided by a discourse vector. Subsequent work in [8] proposes a different weighting scheme based on a modification of the underlying language model where the word generation is inversely proportional to the angular distance between the discourse and the word vector.

A linear autoencoder for sequences can encode sequences into the hidden state of a linear dynamical system. In [7] it is shown a closed-form expression to compute the optimal linear autoencoder. Other works [26], [27] have shown how to exploit them to pretrain recurrent architectures.

III. SIF Sentence Embeddings

Sentence embeddings obtained by summing the embeddings of their constituent words achieve competitive results, often outperforming more complex models [5]. The SIF embeddings[6] improve this simple averaging scheme by providing a novel way to scale each word vector based on its frequency. Furthermore, SIF removes from each embedding the projection along the first principal component computed by a singular value decomposition on the matrix of sentence embeddings (prior to the removal). Removal of this projection serves to take out common components between sentences which, in the intention of the authors of [6], should be related to grammar or common words and are often not relevant from a semantic perspective. While some methods obtain better results on various benchmarks [21], the advantage of SIF lies in its simplicity coupled with results close to the state-of-the-art. Furthermore, its derivation allows to partially explain why unsupervised embeddings based on the sum and average of word embeddings can be so competitive over recurrent architectures.

The SIF embedding for a sentence represented as a sequence of word embeddings $x_1, \dots, x_l$ is computed as

$$\begin{aligned} \boldsymbol{e}_{avg} &= \frac{1}{l}\sum_{i=1}^{l} \alpha_{x_i} \boldsymbol{x}_i \\ \boldsymbol{e}_{SIF} &= \boldsymbol{e}_{avg} - \mathrm{proj}_{\boldsymbol{c_0}}\, \boldsymbol{e_{avg}}, \end{aligned}$$

where $\boldsymbol{e}_{SIF}$ is the final sentence embedding, α_{x_i} is the coefficient for the word x_i, $\boldsymbol{c}_0$ the common discourse vector, and $\mathrm{proj}_{\boldsymbol{c}}\, \boldsymbol{e} = \frac{\boldsymbol{e}^\top \boldsymbol{c}}{||\boldsymbol{c}||^2} \boldsymbol{e}$ is the projection of $\boldsymbol{e}$ over $\boldsymbol{c}$.

In the remainder of the section, we review the SIF algorithm and the language model on which it is based. The pseudocode for the algorithm is reported in Algorithm 1.

A. Random Walk Language Model

A random walk model can be used to describe the sentence generation process [28]. According to the assumptions of this model, each word $\boldsymbol{x}_t$ is generated by a log-linear model $p(\boldsymbol{x}_t|\boldsymbol{c}_t) \propto \exp(\boldsymbol{x}_t^\top \boldsymbol{c}_t)$, where $\boldsymbol{c}_t$ is the discourse vector at time t. The discourse vector $\boldsymbol{c}_{t+1}$ is obtained by adding a small displacement to the current discourse vector $\boldsymbol{c}_t$. In essence, at each timestep a single word is generated based on the current discourse vector while the discourse vector is performing a slow random walk.

This simple model can be improved by adding two additional generation mechanisms, introduced in the original SIF paper [6]: first, we assume there is a small probability that random words are generated independently from the discourse vector.

Second, we assume that there exists a common discourse vector, dependent on grammar and common language shared by all the sentences. The improved language model can be expressed as

$$\begin{aligned} p(\boldsymbol{x}_t|\boldsymbol{c}_s) &= \alpha p(\boldsymbol{x}_t) + (1-\alpha)\frac{\exp(\boldsymbol{x}_t^\top \tilde{\boldsymbol{c}}^s)}{Z_{\tilde{\boldsymbol{c}}_s}} \\ \tilde{\boldsymbol{c}}_s &= \beta \boldsymbol{c}_0 + (1-\beta)\boldsymbol{c}_s, \qquad \boldsymbol{c}_0 \perp \boldsymbol{c}_s, \end{aligned}$$

where α is the probability to generate a random word, $\boldsymbol{c}_s$ is the discourse vector, assumed constant for the entire sentence, $\boldsymbol{c}_0$ is the common discourse vector, and $Z_{\tilde{\boldsymbol{c}}_s}$ the partition function. This model describes the three different word generation mechanisms discussed above: the random generation, controlled by the hyperparameter α, the common discourse, controlled by the hyperparameter β, and the current discourse, which is equivalent to the original random walk model.

B. Maximum Likelihood Estimate of the Discourse Vector

Given that the discourse vector is the main variable controlling the sentence generation process, it is natural to consider using it as an embedding corresponding to the entire sentence. In order to do this, we can compute a maximum likelihood estimate (MLE) for $\boldsymbol{c}_t$ given the word embeddings $\boldsymbol{x}_1, \dots, \boldsymbol{x}_l$. For simplicity, we assume that the discourse vector remains constant for the whole sentence, which is approximately true for small sentences since the random walk cannot change the discourse vector too much. Furthermore, we assume that the word vectors are uniformly dispersed and therefore the partition function is constant $Z_{\boldsymbol{c}_s} = Z$. This is a key assumption of [28]. A complete derivation of the MLE is given in [6]. The MLE of the discourse vector $\tilde{\boldsymbol{c}}_s = \boldsymbol{c}_s + \boldsymbol{c}_0$ for a given sentence is a weighted sum of the word embeddings. More precisely, given a sequence of word embeddings $\boldsymbol{x}_1, \dots, \boldsymbol{x}_l$, the MLE for the discourse vector $\boldsymbol{c}_s$ is

$$\begin{aligned} a &= \frac{1-\alpha}{\alpha Z} \\ \tilde{\boldsymbol{c}}_s &= \frac{1}{l}\sum_{i=1}^{l} \frac{a}{p(\boldsymbol{x}_i)+a}\boldsymbol{x}_i \\ \boldsymbol{c}_s &= \tilde{\boldsymbol{c}}_s - \operatorname{proj}_{\boldsymbol{c}_0} \tilde{\boldsymbol{c}}_s. \end{aligned}$$

The coefficients $\alpha_j = \frac{a}{p(\boldsymbol{w}_j)+a}$ are computed, for each word embedding, given the word frequencies $p(\boldsymbol{w})$ and the hyperparameter a. In [6] it is claimed that the method is relatively insensitive to the choice of this hyperparameter, setting it to $a = 10^{-3}$ for all the experiments. To provide a fair comparison we follow the same guideline in our experimental setup and do not tune a. The common discourse vector $\boldsymbol{c}_0$ is computed as the principal component of the matrix with rows $\{\tilde{c}_s | s \in S\}$, where S is our dataset of sentences, computed using its singular value decomposition (SVD).

Algorithm 1 SIF embeddings

```
**Input** set of sentences $S$, word frequencies $p(\boldsymbol{x})$, hyperparameter $a$
**for all** $s \in S$ **do**
    $\boldsymbol{e}_{avg} \leftarrow \frac{1}{|s|}\sum_{\boldsymbol{x}\in s}\frac{a}{p(\boldsymbol{x})+a}\boldsymbol{x}$
$\mathbf{X} \leftarrow$ matrix where each row is a sentence embedding $\boldsymbol{e}_{avg}$ computed for a sentence $s \in S$
$\boldsymbol{c}_0 \leftarrow$ principal direction of $\mathbf{X}$ computed by SVD
$\mathbf{X} \leftarrow \mathbf{X} - \operatorname{proj}_{\boldsymbol{c}_0}\mathbf{X}$
**return** $\mathbf{X}$
```

IV. Linear Autoencoder for Sequences

A Linear Autoencoder for Sequences (LAES) [7] can be used to define a fixed size representation for variable length sequences. Here, we put forward the idea of exploiting it to obtain a fixed length representation of a sentence represented as a sequence of its constituent words' embeddings.

A LAES is made of two components: an encoder, which is defined as a linear dynamical system and allows to encode a sequence into a hidden state, and a decoder, which is a linear function used to reconstruct the original sequence. The entire system is defined by the following equations:

$$\boldsymbol{h}_t = \mathbf{A}\boldsymbol{x}_t + \mathbf{B}\boldsymbol{h}_{t-1} \tag{1}$$

$$\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} = \mathbf{C}\boldsymbol{h}_t, \tag{2}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is an element of the input sequence, $\boldsymbol{h}_t \in \mathbb{R}^p$ the hidden state of the autoencoder, $\mathbf{A} \in \mathbb{R}^{p\times n}$ and $\mathbf{B} \in \mathbb{R}^{p\times p}$ the parameters of the encoder, and $\mathbf{C} \in \mathbb{R}^{(n+p)\times p}$ the parameters of the decoder. Equation (1) allows to compute the state of the autoencoder and, therefore, the sequence encoding, while equation (2) can be used to decode the state and can be applied iteratively to reconstruct the original sequence.

A. Training Algorithm

In this section, we report the closed-form solution that can be used to find the optimal autoencoder realizing the system defined by equations (1) and (2). Let us consider a single sequence $\boldsymbol{x}_1, \dots, \boldsymbol{x}_l$, where $\boldsymbol{x}_i \in \mathbb{R}^n$. We can find the optimal parameters $\mathbf{A}$ and $\mathbf{B}$ of the linear autoencoder by exploiting the data matrix $\boldsymbol{\Xi} \in \mathbb{R}^{l\times ln}$ defined as

$$\underbrace{\begin{bmatrix} \boldsymbol{h}_1^\top \\ \boldsymbol{h}_2^\top \\ \boldsymbol{h}_3^\top \\ \vdots \\ \boldsymbol{h}_l^\top \end{bmatrix}}_{\mathbf{H}} = \underbrace{\begin{bmatrix} \boldsymbol{x}_1^\top & 0 & \dots & 0 \\ \boldsymbol{x}_2^\top & \boldsymbol{x}_1^\top & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_l^\top & \boldsymbol{x}_{l-1}^\top & \dots & \boldsymbol{x}_1^\top \end{bmatrix}}_{\boldsymbol{\Xi}} \underbrace{\begin{bmatrix} \mathbf{A}^\top \\ \mathbf{A}^\top\mathbf{B}^\top \\ \vdots \\ \mathbf{A}^\top\mathbf{B}^{l-1\top} \end{bmatrix}}_{\boldsymbol{\Omega}}.$$

Each row of $\mathbf{H} \in \mathbb{R}^{l\times p}$ represents a state of the autoencoder, while the rows of $\boldsymbol{\Xi}$ represent the reversed input subsequences. We can factorize $\boldsymbol{\Xi}$ using the SVD to obtain $\boldsymbol{\Xi} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^\top$, where $p = rank(\boldsymbol{\Xi})$, $\mathbf{V} \in \mathbb{R}^{l\times p}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{p\times p}$, $\mathbf{U}^\top \in \mathbb{R}^{p\times ln}$. Imposing the constraint $\mathbf{U} = \boldsymbol{\Omega}$ we find $\mathbf{H} = \boldsymbol{\Xi}\boldsymbol{\Omega} = \mathbf{V}\boldsymbol{\Sigma}$, since the columns of $\mathbf{U}^\top$ are orthogonal by construction. The constraint is used to find a state space where the coordinates are uncorrelated. We can find a solution by exploiting the block structure of Ξ and $\mathbf{\Omega}$. First, we must notice that we can divide $\mathbf{\Omega}$ into the following blocks:

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{A}^\top \\ \mathbf{A}^\top \mathbf{B}^\top \\ \mathbf{A}^\top \mathbf{B}^{2^\top} \\ \vdots \\ \mathbf{A}^\top \mathbf{B}^{l-1^\top} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \\ \mathbf{U}_3 \\ \vdots \\ \mathbf{U}_l \end{bmatrix} = \mathbf{U}. \tag{3}$$

From the above expression we can conclude that $\mathbf{A} = \mathbf{U}_1^\top$. Given $\Xi = [\Xi_1, \dots, \Xi_l]$, we notice each $\Xi_i \in \mathbb{R}^{l \times n}$ can be expressed in terms of the previous blocks $\Xi_j, j < i$ with the following expression

$$\Xi_{i+1} = \mathbf{R}\Xi_i = \begin{bmatrix} 0_{1\times(l-1)} & 0_{1\times 1} \\ \mathbb{I}_{(n-1)\times(n-1)} & 0_{(n-1)\times 1} \end{bmatrix} \Xi_i \tag{4}$$

$$\mathbf{R}\Xi_l = 0. \tag{5}$$

From the singular value decomposition, it follows $\Xi_i = \mathbf{V}\mathbf{\Sigma}\mathbf{U}_i^\top$, which combined with (4) gives $\mathbf{U}_{i+t} = \mathbf{U}_i \mathbf{Q}^t$, where $\mathbf{Q} = \mathbf{\Sigma}\mathbf{V}^\top \mathbf{R}^\top \mathbf{V}\mathbf{\Sigma}^{-1}$. It is also easy to verify that $\mathbf{U}_l \mathbf{Q} = 0$. Therefore, Eq. (3) can be satisfied by choosing $\mathbf{B} = \mathbf{Q}^\top$. Furthermore, since Ξ can be obtained by $\mathbf{H}\mathbf{U}^\top = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top = \Xi$ we define the matrix $\mathbf{C} = \begin{bmatrix} \mathbf{A}^\top \\ \mathbf{B}^\top \end{bmatrix}$ which constitutes the decoder of our model and can be used to reconstruct the original sequence iteratively:

$$\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^\top \\ \mathbf{B}^\top \end{bmatrix} \boldsymbol{h}_t = \mathbf{C}\boldsymbol{h}_t.$$

This algorithm can be easily generalized to a dataset S with multiple sequential samples by modifying the matrices Ξ and $\mathbf{R}$ to account for the presence of multiple sequences. The data matrix Ξ can be obtained by stacking each data matrix Ξ_{s_i} corresponding to the sequence $s_i \in S$ on top of each other, padding with zeroes the last columns in case of different lengths. As an example, let us assume a dataset with two sequences s_1 and s_2 of length l_1, l_2, with $l_1 > l_2$. The matrices Ξ and $\mathbf{R}$ become:

$$\Xi = \begin{bmatrix} \Xi_{s_1} & \\ \Xi_{s_2} & 0_{l_2 \times (l_2 - l_1)} \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} \mathbf{R}_{s_1} & 0 \\ 0 & \mathbf{R}_{s_2} \end{bmatrix}.$$

The terms $\Xi_{s_1}, \Xi_{s_2}, \mathbf{R}_{s_1}, \mathbf{R}_{s_2}$ are defined like the corresponding matrices for the single sequence case.

The computational cost of the algorithm is dominated by the SVD of the matrix Ξ. For an efficient approximated solution we implement the algorithm proposed in [26], which exploits the structure of Ξ to approximate the SVD.

Note that often, in practical problems, we have $p < rank(\Xi)$. This implies that the autoencoder will incur in errors during the reconstruction, while for $p = rank(\Xi)$ the autoencoder is able to reconstruct the sequences without errors and with a minimal hidden state size.

V. Linear Autoencoder Sentence Embeddings

In this section, we discuss several alternatives to compute sentence embeddings from word embeddings using the LAES model. Consider a dataset of sequences $S = \{s_1, \dots, s_D\}$, where each sequence is represented by a list of word embeddings $\boldsymbol{x}_1, \dots, \boldsymbol{x}_l$, such that $\boldsymbol{x}_i \in \mathbb{R}^{n_x}$. We would like to define its corresponding sentence embedding $\boldsymbol{e} \in \mathbb{R}^{n_e}$. We can train a linear autoencoder with parameters $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, and with hidden space size n_h. The hidden states of the autoencoders can be computed by applying Eq. (1)

$$\begin{aligned} \boldsymbol{h}_0 &= 0 \\ \boldsymbol{h}_i &= \mathbf{A}x_i + \mathbf{B}\boldsymbol{h}_{i-1}, \qquad i > 0. \end{aligned}$$

The approaches discussed in the following require only a small amount of supervised data to select the optimal hidden state size n_h for the final task.

Figure 1 shows a schematic view of the proposed approaches.

A. Hidden State Embeddings

The first approach uses the final hidden state of the encoder to represent the entire sentence:

$$\boldsymbol{e}_h = \boldsymbol{h}_l \tag{6}$$

The method is fully unsupervised since it is based only on the reconstruction error that is implicitly used to find the optimal autoencoder. Previous results in the literature [5] tried similar approaches using RNN and LSTM architectures, however they did not show good results due to overfitting. These models require large datasets in order to be trained effectively. By comparison, the optimal linear autoencoder for sequences can be found with a closed-form expression and therefore it is more interesting in situations where only a small amount of training data is available. Since the last hidden state contains sufficient features to decode the original sequence, it should also contain features useful to solve several NLP tasks.

B. Reconstruction Embeddings

The reconstruction embeddings approach consists of an average of the word embeddings reconstructed by the autoencoder. The linear autoencoder for sequences can be used to compute an approximated reconstruction of the original sequence, which is equal to the original sequence only if $n_h = rank(\Xi)$. Using the reconstructed sequence in place of the original could, in principle, allow discarding noise encoded in the word embeddings. For example, it is known that low-frequency words often have poor quality embeddings [29], [30] and therefore it could be beneficial to partially discard them.

To obtain a single vector we can average the reconstructed sequence as follows

$$\begin{aligned} \tilde{\boldsymbol{h}}_l &= \boldsymbol{h}_l \\ \tilde{\boldsymbol{x}}_i &= \mathbf{A}^\top \tilde{\boldsymbol{h}}_i \\ \boldsymbol{e}_{reco} &= \frac{1}{l}\sum_{i=1}^{l} \tilde{\boldsymbol{x}}_i. \end{aligned}$$

The method is similar to the averaged embeddings, but it can account for the word order and discard the information which is not relevant for the reconstruction.

C. Residual Embeddings

Residual embeddings are inspired by the idea of common discourse removal proposed in SIF. Consider the language model discussed in Section III-A. We have seen that the random walk model generates new words conditioned on the orthogonal discourse vectors $\boldsymbol{c}_0$ and $\boldsymbol{c}_s$. We can estimate $\tilde{\boldsymbol{c}}_s = \boldsymbol{c}_o + \boldsymbol{c}_s$ with a weighted average of word embeddings. Since $\boldsymbol{c}_0$ is shared by all sentences, we would like to remove it from the final sentence embeddings since it does not contains any sentence-specific information.

In SIF the common discourse vector $\boldsymbol{c}_0$ is estimated as the principal component of the embedding matrix computed using the SVD. This approach ignores the word order since it estimates $\boldsymbol{c}_0$ from weighted sums of word embeddings. We propose to use a linear autoencoder for sequences to estimate the common discourse vector. If $n_h < rank(\boldsymbol{\Xi})$ the autoencoder will not reconstruct the sequence exactly. Using a small hidden state size n_h allows reconstructing only the common discourse shared by the majority of the sentences in the training data. Unlike the SVD, the linear autoencoder takes into consideration the order of the words. The sentence embedding $\boldsymbol{e}_{res}$ can be computed as

$$\begin{aligned}\tilde{\boldsymbol{h}}_l &= \boldsymbol{h}_l\\ \tilde{\boldsymbol{x}}_i &= \mathbf{A}^\top \tilde{\boldsymbol{h}}_i\\ \boldsymbol{e}_{res} &= \frac{1}{l}\sum_{i=1}^{l}(\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i).\end{aligned}$$

We must note that the reconstruction and residual embeddings have completely opposite and competing objectives. Reconstruction embeddings try to remove the noise contained in the sentence and each word embedding by reducing the size of the linear autoencoder. Residual embeddings instead try to remove the common discourse by reconstructing the sentence with a linear autoencoder with a small hidden state, and therefore able only to reconstruct a small part of the original sentence. While both approaches have their merits, the experimental results show that the latter approach is much more competitive in practice.

Algorithm 2 shows the pseudocode for residual embeddings. The algorithm is similar to SIF (Algorithm 1) but it changes the common discourse vector estimantion. To provide a fair comparison of the two estimation mechanisms, in our empirical assessment, we have used the same reweighting scheme originally adopted by SIF.

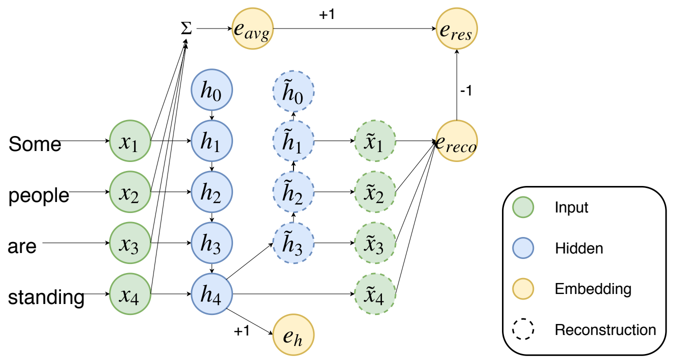

Fig. 1. Graphical representation of the different LAES embeddings. Input nodes are represented in green, hidden states of the LAES in blue, and the embeddings are represented in yellow. Dashed nodes represent reconstructed values.

Algorithm 2 LAES residual sentence embeddings

```
Input set of sentences $S$, linear autoencoder $LA$ trained on similar sentences
for all $s \in S$ do
    $\boldsymbol{y} \leftarrow LA.encode(s)$
    $\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_{|s|} \leftarrow LA.decode(\boldsymbol{y})$
    $\boldsymbol{e}_{res} \leftarrow \frac{1}{|s|}\sum_{i=1}^{|s|}(\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i)$
$\mathbf{X} \leftarrow$ matrix where each row is a sentence embedding $\boldsymbol{e}_{res}$
return $\mathbf{X}$
```

D. Bidirectional Models

All previous embeddings can also be used with bidirectional models, where two different autoencoders are trained, the first on the original sequences, called the forward model, and the second on the reversed sequences, called the backward model. Bidirectional models are a popular approach to improve the performance of recurrent models[31]. We tested two different approaches, based on the sum of the embeddings and the concatenation

$$\boldsymbol{e}_{sum} = \frac{1}{2}(\boldsymbol{e}_{forward} + \boldsymbol{e}_{backward}) \tag{7}$$

$$\boldsymbol{e}_{cat} = \begin{bmatrix}\boldsymbol{e}_{forward} & \boldsymbol{e}_{backward}\end{bmatrix} \tag{8}$$

where we denote with $\boldsymbol{e}_{forward}$ the embedding computed using the forward model and $\boldsymbol{e}_{backward}$ the embedding computed by the backward model, while $\boldsymbol{e}_{sum}$ and $\boldsymbol{e}_{cat}$ are the embeddings obtained with the sum and concatenation, respectively.

VI. Experimental Results

We evaluate the proposed approaches on two different datasets related to the problem of assessing semantic similarity: SICK[32] and STSBenchmark[33]. These datasets have been created to evaluate distributional semantic models.

SICK [32] (Sentences Involving Compositional Knowledge) consists of 10 000 English sentence pairs randomly extracted from two different datasets: 8K ImageFlickr[34] and SemEval 2012 STS MSR-Video Description[35]. The sentences have been preprocessed to generate new sentences with desirable

TABLE I
EXPERIMENTAL RESULTS ON SICK USING GLOVE EMBEDDINGS. n_h IS THE HIDDEN SIZE OF THE AUTOENCODER FOR LAES OR THE NUMBER OF PRINCIPAL COMPONENTS FOR SIF. BEST RESULT IN BOLD.

	SICK				STSBenchmark			
model	n_h	TRAIN	DEV	TEST	n_h	TRAIN	DEV	TEST
SIF [6]	1	-	-	72.2	15	-	80.1	72.0
uSIF [8]	-	-	-	-	5	-	-	71.5
LAES-HID	2400	41.8	36.7	40.6	2400	30.0	40.5	35.0
LAES-RECO	2400	72.9	70.5	71.9	2400	69.1	71.5	63.2
LAES-RES	2	73.2	71.3	72.5	120	76.2	80.2	71.8
LAES-BRES	1	73.5	71.6	**72.9**	84	76.5	81.4	**72.3**

properties that should be captured by a distributional model. Each pair is manually annotated with a semantic relatedness score ranging from 0 to 5.

STSBenchmark contains a selection of the datasets used for the semantic similarity challenges organized at SemEval between 2012 and 2017[35]. It consists of 8628 sentence pairs separated into train-dev-test splits and from three different genres (news, caption, forum).

The similarity between two sentences is computed in terms of the cosine similarity between their sentence embedding vectors. The performance of the models is evaluated with the Pearson correlation score $\rho = \frac{\sigma_{y\tilde{y}}}{\sigma_y \sigma_{\tilde{y}}}$ between the scores assigned by the model $\tilde{y}$ and the ground truth y computed on the test set, where $\sigma_{y\tilde{y}}$ is the covariance between y and $\tilde{y}$ and $\sigma_y, \sigma_{\tilde{y}}$ are the standard deviations.

We use 300-dimensional GloVe vectors[4] as pretrained word embeddings. In our experimental assessment, we compare the different approaches introduced in Section V against the popular SIF[6] model from the literature and uSIF [8] with GloVe embeddings, an improvement over SIF. We restrict the choice of word embeddings to provide a fair comparison between the models. We choose to use GloVe embeddings because they are trained only using unsupervised data and we are interested in the performance of our embeddings for languages that do not have large corpora of labeled data. In all our tasks, we have rescaled the original GloVe embeddings using SIF reweighting scheme since we found it improved the performance in all our experiments. The linear autoencoder for each model is trained on the sentences from the training set of the benchmarks. For the hidden state (LAES-HID) and reconstruction embeddings (LAES-RECO), we select the hidden state size $n_h \in \{150, 300, 600, 1200, 2400\}$ with a hyperparameter search on the development set. The hidden state size for the residual embeddings (LAES-RES) is selected from the interval $\{1, \dots, 150\}$. The difference between the two hyperparameter spaces is due to the fact that we expect hidden state and reconstruction embeddings to require a larger hidden size. This intuition is confirmed by the experimental results. We report the results of the bidirectional models only with the residual embeddings (LAES-BRES), since they obtained the best results in all the experiments, and select the best configuration between the sum and concatenated embeddings.

Table I shows the performance of the different models on the SICK benchmark, reporting results on the training, development and test set. The term n_h is the number of components used for the hidden state of the linear autoencoder or the number of principal components removed when using SIF. From the results in Table I, it is clear that the reconstruction and hidden state embeddings do not reach satisfying results. Both approaches improve with a larger hidden size but even with $n_h = 2400$ they are not competitive with the alternative methods. For $n_h = 2400$ reconstruction embeddings obtain a Pearson correlation score close to the weighted embeddings. We conclude that even if the reconstruction error is low, the reconstruction objective alone is not enough to obtain good results on semantic similarity tasks. LAES-RES embeddings improve the results obtained by SIF with a better estimation of the common discourse vector.

Table I shows the results on STSBenchmark. While the performance on the test set is quite similar between the two datasets, the optimal parameter n_h is completely different. On SICK the best models remove only a small number of components, 1 for SIF and at most 2 for linear autoencoder embeddings, while STSBenchmark requires a large number of components, 15 for SIF and up to 120 for linear autoencoder embeddings. The difference between the two datasets could be a consequence of the fact that STSBenchmark aggregates data from different years and therefore could have more heterogeneous data. By contrast, the original authors evaluated their approach separately for each version of the dataset, and therefore in their case a single principal component was sufficient to remove the common discourse. Unfortunately, this does not seem to be a viable approach in general, where the split of training sentences into coherent subgroups is typically not available.

The proposed method works with pretrained embeddings. In the first two experiments, we limited ourselves to GloVe vectors, which are completely unsupervised. Other kinds of words embeddings are built exploiting labeled resources and can greatly improve the performance of our model. Therefore, we decided to test our approach against several methods that exploit large resources of labeled data. Notice that all the models are actually unsupervised since the labeled data comes from different sources. Their main advantage is that by exploiting large dataset of natural language inference, domain translation, and paraphrase, they are able to extract semantic features able to generalize to a wide range of tasks. Table II shows the results of different models on SICK and STSBenchmark (whenever results for unsupervised evaluation are available). Our model is still competitive against neural models based on LSTM like InferSent[22].

TABLE II
EXPERIMENTAL RESULTS ON SICK AND STSBENCHMARK FOR DIFFERENT MODELS TRAINED WITH LARGE LABELED CORPORA. BEST RESULTS IN BOLD.

model	SICK	STSBenchmark
InferSent[1]	—	75.8
sent2vec[1]	70.0	75.5
uSIF (ParaNMT)[1]	73.5	**79.5**
SIF (PSL)	72.9	—
LAES-BRES (cat, $n_h = 4$)	**73.8**	**79.5**

A. Discourse Vector Estimation

In this section, we experimentally assess the validity of the hypothesis that a single component can represent the common discourse. Figure 2 shows the behaviour of the performance for SIF and LAES-RES embeddings for varying values of n_h on the test set of STSBenchmark. Both models have been evaluated using GloVe embeddings, setting $a = 10^{-3}$, and varying $n_h \in \{1, \ldots, 150\}$. The trend is predictable, with a single peak at the optimal value and worse performance for values distant from the optimum. Looking at the two curves it can be noticed that the linear autoencoder requires a larger number of components than SIF to reach the optimal performance. This is expected since in order to be able to reconstruct distant words the autoencoder must have a large enough hidden size. SIF principal components ignore the word order and are not affected by the word order and therefore do not suffer from this phenomenon.

From Figure 2 it is easy to see that a single component is not sufficient to remove the common discourse. This is in contrast with the original proposal in [6]. In fact, on STSBenchmark, even a small deviation from the optimal value can cause a large drop in performance, as clearly highlighted in Fig. 2. This finding further justifies the development of more sophisticated methods to estimate the common discourse, like LAES-RES.

VII. CONCLUSION

Sentence embeddings are used as features in many NLP models. Approaches based on weighted word embeddings reach a competitive performance while completely ignoring the word order when computing sentence embeddings, despite its clear semantic importance. In this paper, we show how to compute effective sentence embeddings using linear autoencoder for sequences. We propose and discuss a number of alternative approaches for sequential sentence embedding. Through an empirical assessment, we show the effectiveness of the LAES-RES model, which is weakly supervised, requiring only a small dataset of labeled data to choose the dimension of the hidden state for the autoencoder. The method is competitive with SIF on two different semantic similarity datasets and provides an alternative to estimate the common discourse vector that can effectively account for word order. Similarly to SIF, the method outperforms several supervised models based on neural architectures[6]. The experimental results show that a good estimate of the discourse vector is crucial to obtain optimal performance, in contrast with previous literature that limited the estimation of the common discourse to a single principal component.

[1]results from http://ixa2.si.ehu.es/stswiki/.index.php/STSbenchmark.

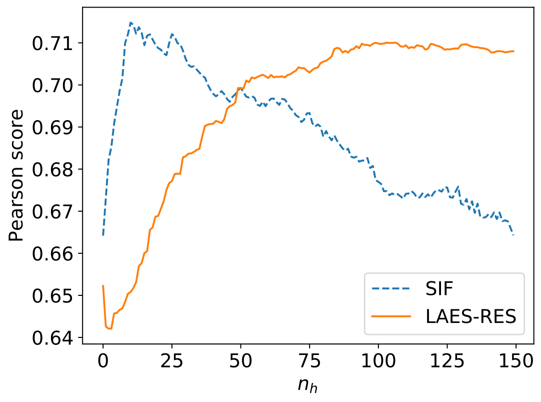

Fig. 2. Test set performance of SIF and LAES-RES on STSBenchmark for different values of n_h.

ACKNOWLEDGMENT

This work has been supported by the Italian Ministry of Education, University, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE), and by an industrial research project funded by Biobeats Group Ltd.

REFERENCES

[1] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.
[2] J. R. Firth, "A synopsis of linguistic theory, 1930-1955," *Studies in linguistic analysis*, 1957.
[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.
[4] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
[5] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Towards Universal Paraphrastic Sentence Embeddings," *arXiv preprint arXiv:1511.08198*, 11 2015. [Online]. Available: http://arxiv.org/abs/1511.08198
[6] S. Arora, Y. Liang, and T. Ma, "A simple but tough to beat baseline for sentence embeddings," *ICLR*, pp. 1–14, 2017.
[7] A. Sperduti, "Linear autoencoder networks for structured data," in *International Workshop on Neural-Symbolic Learning and Reasoning*, 2013.
[8] K. Ethayarajh, "Unsupervised Random Walk Sentence Embeddings: A Strong but Simple Baseline," *ACL*, pp. 1–10, 2018. [Online]. Available: http://www.aclweb.org/anthology/W18-3012
[9] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.
[10] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[11] Q. V. Le and T. Mikolov, “Distributed Representations of Sentences and Documents,” in *ICML*, 5 2014. [Online]. Available: http://arxiv.org/abs/1405.4053

[12] O. Levy and Y. Goldberg, “Neural Word Embedding as Implicit Matrix Factorization,” in *Advances in Neural Information Processing Systems*, 2014, pp. 2177–2185. [Online]. Available: https://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization

[13] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, “Skip-Thought Vectors,” *Advances in Neural Information Processing Systems*, 6 2015. [Online]. Available: http://arxiv.org/abs/1506.06726

[14] P. Frasconi, M. Gori, and A. Sperduti, “A general framework for adaptive processing of data structures,” *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=712151

[15] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, “Recursive Deep Models for Semantic Compositionality,” in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[16] K. S. Tai, R. Socher, and C. D. Manning, “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks,” in *arXiv preprint arXiv:1503.00075*, 2 2015. [Online]. Available: http://arxiv.org/abs/1503.00075

[17] D. Bacciu and A. Bruno, “Deep Tree Transductions - A Short Survey,” in *Proceedings of the 2019 INNS Big Data and Deep Learning (INNSBDDL 2019)*, 2019.

[18] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” *arXiv preprint arXiv:1404.2188*, 2014.

[19] Y. Wang, H. Huang, C. Feng, Q. Zhou, J. Gu, and X. Gao, “Cse: Conceptual sentence embeddings based on attention model,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 505–515.

[20] J. Hochreiter, Sepp; Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1–32, 1997.

[21] J. Wieting and K. Gimpel, “Revisiting Recurrent Networks for Paraphrastic Sentence Embeddings,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 4 2017, pp. 2078–2088. [Online]. Available: http://arxiv.org/abs/1705.00364

[22] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data,” *EMNLP*, 5 2017. [Online]. Available: http://arxiv.org/abs/1705.02364

[23] M. Pagliardini, P. Gupta, and M. Jaggi, “Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features,” 3 2017. [Online]. Available: https://arxiv.org/abs/1703.02507

[24] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, “A latent variable model approach to pmi-based word embeddings,” *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 385–399, 2016.

[25] T. B. Hashimoto, D. Alvarez-Melis, and T. S. Jaakkola, “Word embeddings as metric recovery in semantic spaces,” *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 273–286, 2016.

[26] L. Pasa and A. Sperduti, “Pre-training of Recurrent Neural Networks via Linear Autoencoders,” *Advances in Neural Information Processing Systems 27*, pp. 3572–3580, 2014. [Online]. Available: http://papers.nips.cc/paper/5271-pre-training-of-recurrent-neural-networks-via-linear-autoencoders.pdf

[27] D. Bacciu, A. Carta, and A. Sperduti, “Linear Memory Networks,” *arXiv preprint arXiv:1811.03356*, 2018.

[28] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, “RAND-WALK: A Latent Variable Model Approach to Word Embeddings,” *arXiv preprint arXiv:1502.03520*, 2 2015. [Online]. Available: http://arxiv.org/abs/1502.03520

[29] B. Pierrejean and L. Tanguy, “Predicting Word Embeddings Variability,” in **SEM@NAACL-HLT*, 2018.

[30] O. Levy, Y. Goldberg, and I. Dagan, “Improving Distributional Similarity with Lessons Learned from Word Embeddings,” *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 211–225, 2015.

[31] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM and other neural network architectures,” *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[32] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli, “A SICK cure for the evaluation of compositional distributional semantic models,” in *Proceedings of the 10th Edition of the Language, Resources and Evaluation Conference (LREC 2016)*, 2014, pp. 216–223.

[33] D. M. Cer, M. T. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, “SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation,” in *SemEval@ACL*, 2017.

[34] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier, “Collecting Image Annotations Using Amazon’s Mechanical Turk,” in *Mturk@HLT-NAACL*, 2010.

[35] E. Agirre, D. M. Cer, M. T. Diab, and A. Gonzalez-Agirre, “SemEval-2012 Task 6: A Pilot on Semantic Textual Similarity,” in *SemEval@NAACL-HLT*, 2012.