LEARNING SIMILARITY PRESERVING REPRESENTATIONS WITH NEURAL SIMILARITY AND CONTEXT ENCODERS

Franziska Horn & Klaus-Robert Müller
Machine Learning Group
Technische Universität Berlin
Berlin, Germany
franziska.horn@campus.tu-berlin.de
klaus-robert.mueller@tu-berlin.de

ABSTRACT

We introduce similarity encoders (SimEc), which learn similarity preserving representations by using a feed-forward neural network to map data into an embedding space where the original similarities can be approximated linearly. The model can easily compute representations for novel (out-of-sample) data points, even if the original pairwise similarities of the training set were generated by an unknown process such as human ratings. This is demonstrated by creating embeddings of both image and text data. Furthermore, the idea behind similarity encoders gives an intuitive explanation of the optimization strategy used by the continuous bag-of-words (CBOW) word2vec model trained with negative sampling. Based on this insight, we define context encoders (ConEc), which can improve the word embeddings created with word2vec by using the local context of words to create out-of-vocabulary embeddings and representations for words with multiple meanings. The benefit of this is illustrated by using these word embeddings as features in the CoNLL 2003 named entity recognition task.

1 INTRODUCTION

Many dimensionality reduction or manifold learning algorithms optimize for retaining the pairwise similarities, distances, or local neighborhoods of data points. Classical scaling (Cox & Cox, 2000), kernel PCA (Schölkopf et al., 1998), isomap (Tenenbaum et al., 2000), and LLE (Roweis & Saul, 2000) achieve this by performing an eigendecomposition of some similarity matrix to obtain a low dimensional representation of the original data. However, this is computationally expensive if a lot of training examples are available. Additionally, out-of-sample representations can only be created when the similarities to the original training examples can be computed (Bengio et al., 2004).

For some methods such as t-SNE (van der Maaten & Hinton, 2008), great effort was put into extending the algorithm to work with large datasets (van der Maaten, 2013) or to provide an explicit mapping function which can be applied to new data points (van der Maaten, 2009). Current attempts at finding a more general solution to these issues are complex and require the development of specific cost functions and constraints when used in place of existing algorithms (Bunte et al., 2012), which limits their applicability to new objectives.

In this paper we introduce a new neural network architecture, that we will denote as *similarity encoder* (SimEc), which is able to learn representations that can retain arbitrary pairwise relations present in the input space, even those obtained from unknown similarity functions such as human ratings. A SimEc can learn a linear or non-linear mapping function to project new data points into a lower dimensional embedding space. Furthermore, it can take advantage of large datasets since the objective function is optimized iteratively using stochastic mini-batch gradient descent. We show on both image and text datasets that SimEcs can, on the one hand, recreate solutions found by traditional methods such as kPCA or isomap, and, on the other hand, obtain meaningful embeddings from similarities based on human labels.

Additionally, we propose the new *context encoder* (ConEc) model, a variation of similarity encoders for learning word embeddings, which extends word2vec (Mikolov et al., 2013b) by using the local context of words as input to the neural network to create representations for out-of-vocabulary words and to distinguish between multiple meanings of words. This is shown to be advantageous, for example, if the word embeddings are used as features in a named entity recognition task as demonstrated on the CoNLL 2003 challenge.

2 SIMILARITY ENCODERS

We propose a novel dimensionality reduction framework termed similarity encoder (SimEc), which can be used to learn a linear or non-linear mapping function for computing low dimensional representations of data points such that the original pairwise similarities between the data points in the input space are preserved in the embedding space. For this, we borrow the "bottleneck" neural network (NN) architecture idea from autoencoders (Tishby et al., 2000; Hinton & Salakhutdinov, 2006). Autoencoders aim to transform the high dimensional data points into low dimensional embeddings such that most of the data's variance is retained. Their network architecture has two parts: The first part of the network maps the data points from the original feature space to the low dimensional embedding (at the bottleneck). The second part of the NN mirrors the first part and projects the embedding back to a high dimensional output. This output is then compared to the original input to compute the reconstruction error of the training samples, which is used in the backpropagation procedure to tune the network's parameters. After the training is complete, i.e. the low dimensional embeddings encode enough information about the original input samples to allow for their reconstruction, the second part of the network is discarded and only the first part is used to project data points into the low dimensional embedding space. Similarity encoders have a similar two fold architecture, where in the first part of the network, the data is mapped to a low dimensional embedding, and then in the second part (which is again only used during training), the embedding is transformed such that the error of the representation can be computed. However, since here the objective is to retain the (non-linear) pairwise similarities instead of the data's variance, the second part of the NN does not mirror the first like it does in the autoencoder architecture.

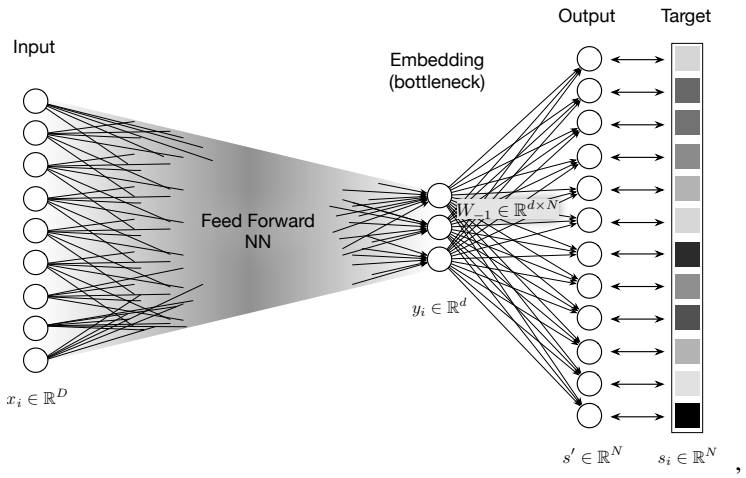

Figure 1: Similarity encoder (SimEc) architecture.

The similarity encoder architecture (Figure 1) uses as the first part of the network a flexible non-linear feed-forward neural network to map the high dimensional input data points $x_i \in \mathbb{R}^D$ to a low dimensional embedding $y_i \in \mathbb{R}^d$ (at the bottleneck). As we make no assumptions on the range of values the embedding can take, the last layer of the first part of the NN (i.e. the one resulting in the embedding) is always linear. For example, with two additional non-linear hidden layers, the embedding would be computed as

$$y_i = \sigma_1(\sigma_0(x_i W_0)W_1)W_2,$$

where σ_0 and σ_1 denote your choice of non-linear activation functions (e.g. tanh, sigmoid, or relu), but there is no non-linearity applied after multiplying with W_2. The second part of the network then

consists of a single additional layer with the weight matrix $W_{-1} \in \mathbb{R}^{d \times N}$ to project the embedding to the output, the approximated similarities $s' \in \mathbb{R}^N$:

$$s' = \sigma_{-1}(y_i W_{-1}).$$

These approximated similarities are then compared to the target similarities (for one data point this is the corresponding row $s_i \in \mathbb{R}^N$ of the similarity matrix $S \in \mathbb{R}^{N \times N}$ of the N training samples) and the computed error is used to tune the network's parameters with backpropagation.

For the model to learn most efficiently, the exact form of the cost function to optimize as well as the type of non-linearity σ_{-1} applied when computing the network's output should be chosen with respect to the type of target similarities that the model is supposed to preserve. In the experimental section of the paper we are considering two application scenarios of SimEcs: a) to obtain the same low dimensional embedding as found by spectral methods such as kPCA, and b) to embed data points such that binary similarity relations obtained from human labels are preserved.
In the first case (further discussed in the next section), we omit the non-linearity when computing the output of the network, i.e. $s' = y_i W_{-1}$, since the target similarities, computed by some kernel function, are not necessarily constrained to lie in a specific interval. As the cost function to minimize we choose the mean squared error between the output (approximated similarities) and the original (target) similarities. A regularization term is added to encourage the weights of the last layer (W_{-1}) to be orthogonal.[1] The model's objective function optimized during training is therefore:

$$\min \frac{1}{N} \sum_{i=1}^{N} \|s_i - s'\|_2^2 + \lambda \frac{1}{d^2 - d} \left\| W_{-1} W_{-1}^\top - \text{diag}(W_{-1} W_{-1}^\top) \right\|_1$$

where $\| \cdot \|_p$ denotes the respective p-norms for vectors and matrices and λ is a hyperparameter to control the strength of the regularization.
In the second case, the target similarities are binary and it therefore makes sense to use a non-linear activation function in the final layer when computing the output of the network to ensure the approximated similarities are between 0 and 1 as well:[2]

$$s' = \sigma_{-1}(y_i W_{-1}) \quad \text{with} \quad \sigma_{-1}(z) = \frac{1}{1 + e^{-10(z-0.5)}}.$$

While the mean squared error between the target and approximated similarities would still be a natural choice of cost function to optimize, with the additional non-linearity in the output layer, learning might be slow due to small gradients and we therefore instead optimize the cross-entropy:

$$\min \; -\frac{1}{N} \sum \left[s_i \ln(s') + (1 - s_i) \ln(1 - s') \right].$$

For a different application scenario, yet another setup might lead to the best results. When using SimEcs in practice, we recommend to first try the first setup, i.e. keeping the output layer linear and minimizing the mean squared error, as this often already gives quite good results.

After the training is completed, only the first part of the neural network, which maps the input to the embedding, is used to create the representations of new data points. Depending on the complexity of the feed-forward NN, the mapping function learned by similarity encoders can be linear or non-linear, and because of the iterative optimization using stochastic mini-batch gradient descent, large amounts of data can be utilized to learn optimal representations.[3]

2.1 Relation to Kernel PCA

Kernel PCA (kPCA) is a popular non-linear dimensionality reduction algorithm, which performs the eigendecomposition of a kernel matrix to obtain low dimensional representations of the data points

[1] To get embeddings similar to those obtained by kPCA, orthogonal weights in the last layer of the NN help as they correspond to the orthogonal eigenvectors of the kernel matrix found by kPCA.

[2] This scaled and shifted sigmoid function maps values between 0 and 1 almost linearly while thresholding values outside this interval.

[3] To speed up the training procedure and limit memory requirements for large datasets, the columns of the similarity matrix can also be subsampled (yielding $S \in \mathbb{R}^{N \times n}$), i.e. the number of target similarities (and the dimensionality of the output layer) is $n < N$, however all N training examples can still be used as input to train the network.

(Schölkopf et al., 1998). However, if the kernel matrix is very large this becomes computationally very expensive. Additionally, there are constraints on possible kernel functions (should be positive semi-definite) and new data points can only be embedded in the lower dimensional space if their kernel map (i.e. the similarities to the original training points) can be computed. As we show below, SimEc can optimize the same objective as kPCA but addresses these shortcomings.

The general idea is that both kPCA and SimEc embed the N data points in a feature space where the given target similarities can be approximated linearly (i.e. with the scalar product of the embedding vectors). When the error between the approximated (S') and the target similarities (S) is computed as the mean squared error, kPCA finds the optimal approximation by performing the eigendecomposition of the (centered) target similarity matrix, i.e.

$$S' = YY^\top,$$

where $Y \in \mathbb{R}^{N \times d}$ is the low dimensional embedding of the data based on the eigenvectors belonging to the d largest eigenvalues of S.

In addition to the embedding itself, it is often desired to have a parametrized mapping function, which can be used to project new (out-of-sample) data points into the embedding space. If the target similarity matrix is the linear kernel, i.e. $S = XX^\top$ where $X \in \mathbb{R}^{N \times D}$ is the given input data, this can easily be accomplished with traditional PCA. Here, the covariance matrix of the centered input data, i.e. $C = X^\top X$ is decomposed to obtain a matrix with parameters, $\tilde{W} \in \mathbb{R}^{D \times d}$, based on the eigenvectors belonging to the d largest eigenvalues of the covariance matrix. Then the optimal embedding (i.e. the same solution obtained by linear kPCA) can be computed as

$$Y = X\tilde{W}.$$

This serves as a mapping function, with which new data points can be easily projected into the lower dimensional embedding space.

When using a similarity encoder to embed data in a low dimensional space where the linear similarities are preserved, the SimEc's architecture would consist of a neural network with a single linear layer, i.e. the parameter matrix W_0, to project the input data X to the embedding $Y = XW_0$, and another matrix $W_{-1} \in \mathbb{R}^{d \times N}$ used to approximate the similarities as

$$S' = YW_{-1}.$$

From these formulas one can immediately see the link between linear similarity encoders and PCA / linear kPCA: once the parameters of the neural network are tuned correctly, W_0 would correspond to the mapping matrix $\tilde{W}$ found by PCA and W_{-1} could be interpreted as $Y^\top$, i.e. Y would be the same eigenvector based embedding as found with linear kPCA.

Finding the corresponding function to map new data points into the embedding space is trivial for linear kPCA, but this is not the case for other kernel functions. While it is still possible to find the optimal embedding with kPCA for non-linear kernel functions, the mapping function remains unknown and new data points can only be projected into the embedding space if we can compute their kernel map, i.e. the similarities to the original training examples (Bengio et al., 2004). Some attempts were made to manually define an explicit mapping function to represent data points in the kernel feature space, however this only works for specific kernels and there exists no general solution (Rahimi & Recht, 2007). As neural networks are universal function approximators, with the right architecture similarity encoders could instead learn arbitrary mapping functions for unknown similarities to arrive at data driven kernel learning solutions.

2.2 Model overview

The properties of similarity encoders are summarized in the following. The objective of this dimensionality reduction approach is to retain pairwise similarities between data points in the embedding space. This is achieved by tuning the parameters of a neural network to obtain a linear or non-linear mapping (depending on the network's architecture) from the high dimensional input to the low dimensional embedding. Since the cost function is optimized using stochastic mini-batch gradient descent, we can take advantage of large datasets for training. The embedding for new test points can be easily computed with the explicit mapping function in the form of the tuned neural network. And since there is no need to compute the similarity of new test examples to the original training data for out-of-sample solutions (like with kPCA), the target similarities can be generated by an unknown process such as human similarity judgments.

2.3 EXPERIMENTS

In the following experiments we demonstrate that similarity encoders can, on the one hand, reach the same solution as kPCA, and, on the other hand, generate meaningful embeddings from human labels. To illustrate that this is independent of the type of data, we present results obtained both on the well known MNIST handwritten digits dataset as well as the 20 newsgroups text corpus. Further details as well as the code to replicate these experiments and more is available online.[4]

We compare the embedding found with linear kPCA to that created with a linear similarity encoder (consisting of one linear layer mapping the input to the embedding and a second linear layer to project the embedding to the output, i.e. computing the approximated similarities). Additionally, we show that a non-linear SimEc can approximate the solution found with isomap (i.e. the eigendecomposition of the geodesic distance matrix). We found that for optimal results the kernel matrix used as the target similarity matrix for the SimEc should first be centered (as it is being done for kPCA as well (Müller et al., 2001)).

In a second step, we show that SimEcs can learn the mapping to a low dimensional embedding for arbitrary similarity functions and reliably create representations for new test samples without the need to compute their similarities to the original training examples, thereby going beyond the capabilities of kPCA. For both datasets we illustrate this by using the class labels assigned to the samples by human annotators to create the target similarity matrix for the training fold of the data, i.e. S is 1 for data points belonging to the same class and 0 everywhere else. We compare the solutions found by SimEc architectures with a varying number of additional non-linear hidden layers in the first part of the network (while keeping the embedding layer linear as before) to show how a more complex network improves the ability to map the data into an embedding space in which the class-based similarities are retained.

MNIST The MNIST dataset contains 28×28 pixel images depicting handwritten digits. For our experiments we randomly subsampled 10k images from all classes, of which 80% are assigned to the training fold and the remaining 20% to the test fold (in the following plots, data points belonging to the training set are displayed transparently while the test points are opaque). As shown in Figure 2, the embeddings of the MNIST dataset created with linear kPCA and a linear similarity encoder, which uses as target similarities the linear kernel matrix, are almost identical (up to a rotation). The same holds true for the isomap embedding, which is well approximated by a non-linear SimEc with two hidden layers using the geodesic distances between the data points as targets (Figure 8 in the Appendix). When optimizing SimEcs to retain the class-based similarities (Figure 3), additional

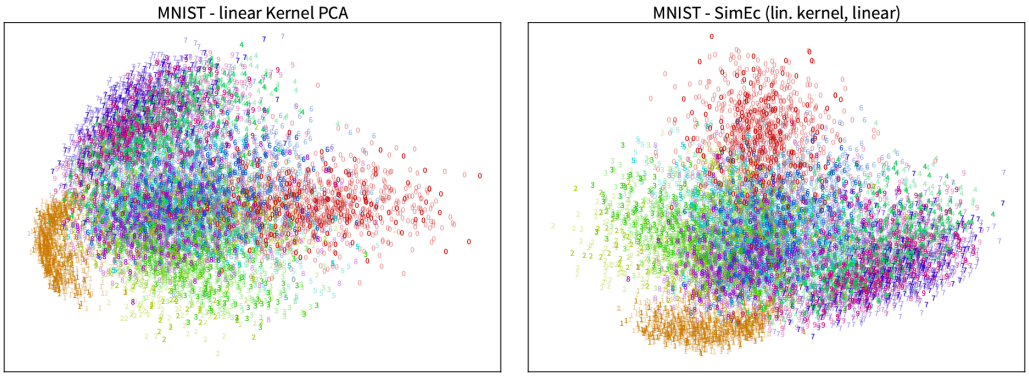

Figure 2: MNIST digits visualized in two dimensions by linear kPCA and a linear SimEc.

non-linear hidden layers in the feed-forward NN can improve the embedding by further separating data points belonging to different classes in tight clusters. As it can be seen, the test points (opaque) are nicely mapped into the same locations as the corresponding training points (transparent), i.e. the model learns to associate the input pixels with the class clusters only based on the imposed similarities between the training data points.

[4] https://github.com/cod3licious/simec/examples_simec.ipynb

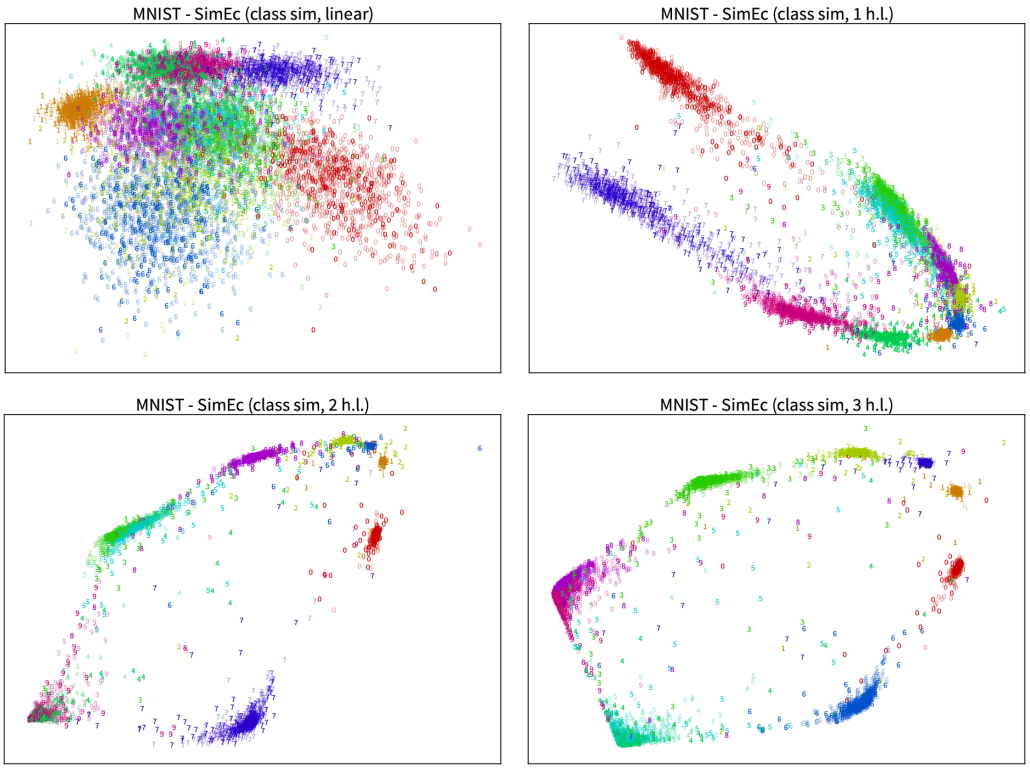

Figure 3: MNIST digits visualized in two dimensions by SimEcs with an increasing number of non-linear hidden layers and the objective to retain similarities based on class membership.

20 newsgroups The 20 newsgroups dataset consists of around 18k newsgroup posts assigned to 20 different topics. We take a subset of seven categories and use the original train/test split ($\sim$4.1k and $\sim$2.7k samples respectively) and remove metadata such as headers to avoid overfitting.[5] All text documents are transformed into 46k dimensional tf-idf feature vectors, which are used as input to the SimEc and to compute the linear kernel matrix of the training fold. The embedding created with linear kPCA is again well approximated by the solution found with a corresponding linear SimEc (Figure 9 in the Appendix). Additionally, this serves as an example where traditional PCA is not an option to obtain the corresponding mapping matrix for the linear kPCA solution, as due to the high dimensionality of the input data and comparatively low number of samples, the empirical covariance matrix would be poorly estimated and too large to decompose into eigenvalues and -vectors. With the objective to retain the class-based similarities, a SimEc with a non-linear hidden layer clusters documents by their topics (Figure 4).

3 Context Encoders

Representation learning is very prominent in the field of natural language processing (NLP). For example, word embeddings learned by neural network language models were shown to improve the performance when used as features for supervised learning tasks such as named entity recognition (NER) (Collobert et al., 2011; Turian et al., 2010). The popular *word2vec* model (Figure 5) learns meaningful word embeddings by considering only the words' local contexts and thanks to its shallow architecture it can be trained very efficiently on large corpora. However, an important limiting factor of current word embedding models is that they only learn the representations for words from a fixed vocabulary. This means, if in a task we encounter a new word which was not present in the texts used for training, we can not create an embedding for this word without repeating the time consuming

[5] `http://scikit-learn.org/stable/datasets/twenty_newsgroups.html`

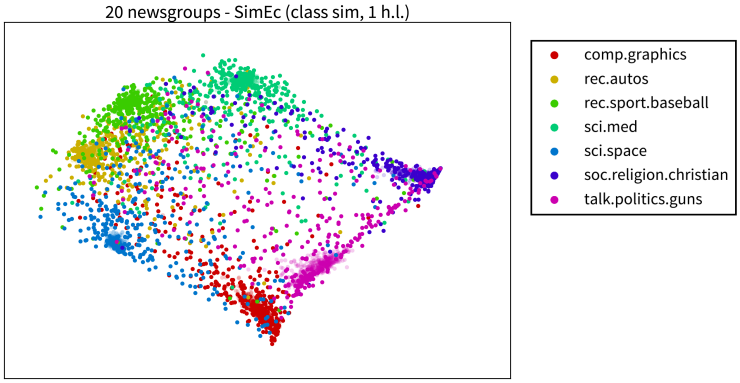

Figure 4: 20 newsgroups texts visualized in two dimensions by a non-linear SimEc with one hidden layer and the objective to preserve the similarities based on class membership in the embedding.

training procedure of the model.[6] Additionally, word2vec, like many other approaches, only learns a single representation for every word. However, it is often the case that a single word can have multiple meanings, e.g. "Washington" is both the name of a US state as well as a former president. It is only the local context in which these words appear that lets humans resolve this ambiguity and identify the proper sense of the word in question. While attempts were made to improve this, they lack flexibility as they require a clustering of word contexts beforehand (Huang et al., 2012), which still does not guarantee that all possible meanings of a word have been identified prior in the training documents. Other approaches require additional labels such part-of-speech tags (Trask et al., 2015) or other lexical resources like WordNet (Rothe & Schütze, 2015) to create word embeddings which distinguish between the different senses of a word.

As a further contribution of this paper we provide a link between the successful word2vec natural language model and similarity encoders and thereby propose a new model we call *context encoder* (ConEc), which can efficiently learn word embeddings from huge amounts of training data and additionally make use of local contexts to create representations for out-of-vocabulary words and help distinguish between multiple meanings of words.

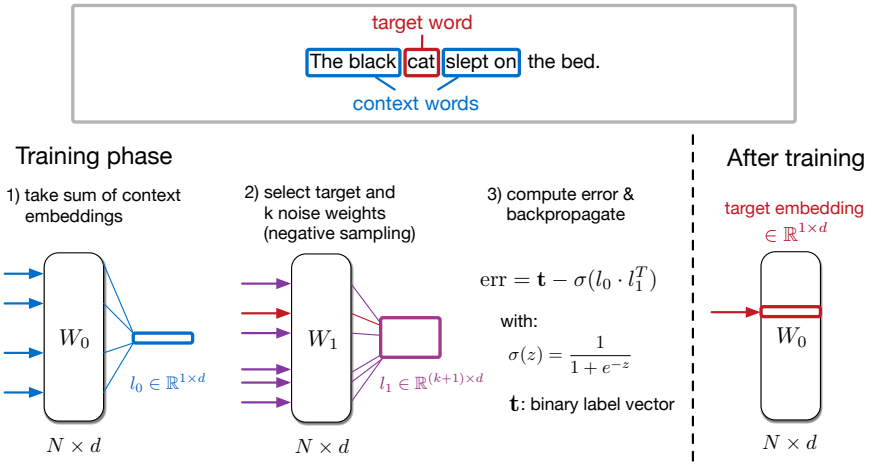

Figure 5: Continuous BOW word2vec model trained using negative sampling (Mikolov et al., 2013a;b; Goldberg & Levy, 2014).

[6]In practice these models are trained on such a large vocabulary that it is rare to encounter a word which does not have an embedding. However, there are still scenarios where this is the case, for example, it is unlikely that the term "W10281545" is encountered in a regular training corpus, but we might still want its embedding to represent a search query like "whirlpool W10281545 ice maker part".

Formally, word embeddings are d-dimensional vector representations learned for all N words in the vocabulary. Word2vec is a shallow model with parameter matrices $W_0, W_1 \in \mathbb{R}^{N \times d}$, which are tuned iteratively by scanning huge amounts of texts sentence by sentence (see Figure 5). Based on some context words the algorithm tries to predict the target word between them. Mathematically this is realized by first computing the sum of the embeddings of the context words by selecting the appropriate rows from W_0. This vector is then multiplied by several rows selected from W_1: one of these rows corresponds to the target word, while the others correspond to k 'noise' words, selected at random (negative sampling). After applying a non-linear activation function, the backpropagation error is computed by comparing this output to a label vector $\mathbf{t} \in \mathbb{R}^{k+1}$, which is 1 at the position of the target word and 0 for all k noise words. After the training of the model is complete, the word embedding for a target word is the corresponding row of W_0.

The main principle utilized when learning word embeddings is that similar words appear in similar contexts (Harris, 1954; Melamud et al., 2015). Therefore, in theory one could compute the similarities between all words by checking how many context words any two words generally have in common (possibly weighted somehow to reduce the influence of frequent words such as 'the' and 'and'). However, such a word similarity matrix would be very large, as typically the vocabulary for which word embeddings are learned comprises several $10,000$ words, making it computationally too expensive to be used with similarity encoders. But this matrix would also be quite sparse, because many words in fact do not occur in similar contexts and most words only have a handful of synonyms which could be used in their place. Therefore, we can view the negative sampling approach used for word2vec (Mikolov et al., 2013b) as an approximation of the words' context based similarities: while the similarity of a word to itself is 1, if for one word we select k random words out of the huge vocabulary, it is very unlikely that they are similar to the target word, i.e. we can approximate their similarities with 0. This is the main insight necessary for adapting similarity encoders to be used for learning (context sensitive) word embeddings.

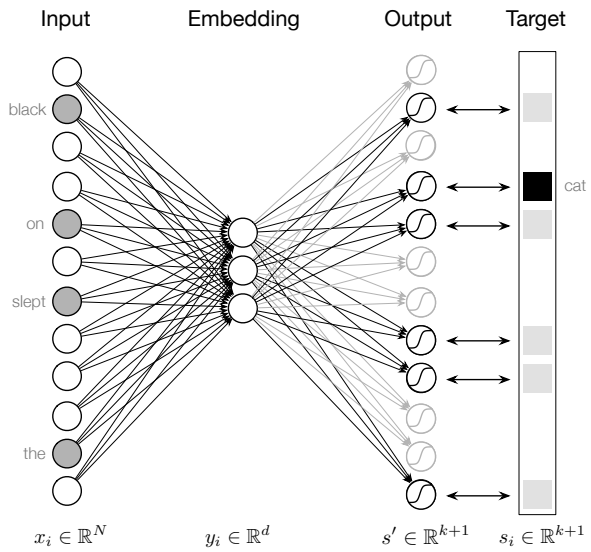

Figure 6: Context encoder (ConEc) architecture. The input consists of a context vector, but instead of comparing the output to a full similarity vector, only the target word and k noise words are considered.

Figure 6 shows the architecture of the context encoder. For the training procedure we stick very closely to the optimization strategy used by word2vec: while parsing a document, we again select a target word and its context words. As input to the context encoder network, we use a vector x_i of length N (i.e. the size of the vocabulary), which indicates the context words by non-zero values (either binary or e.g. giving lower weight to context words further away from the target word). This vector is then multiplied by a first matrix of weights $W_0 \in \mathbb{R}^{N \times d}$ yielding a low dimensional embedding y_i, comparable to the summed context embedding created as a first step when training the word2vec model. This embedding is then multiplied by a second matrix $W_1 \in \mathbb{R}^{d \times N}$ to yield the output. Instead of comparing this output vector to a whole row from a word similarity matrix (as we would with similarity encoders), only $k + 1$ entries are selected, namely those belonging to

the target word as well as k random and unrelated noise words. After applying a non-linearity we compare these entries $s' \in \mathbb{R}^{k+1}$ to the binary target vector exactly as in the word2vec model and use error backpropagation to tune the parameters.

Up to now, there are no real differences between the word2vec model and our context encoders, we have merely provided an intuitive interpretation of the training procedure and objective. The main deviation from the word2vec model lies in the computation of the word embedding for a target word after the training is complete. In the case of word2vec, the word embedding is simply the row of the tuned W_0 matrix. However, when considering the idea behind the optimization procedure, we instead propose to compute a target word's representation by multiplying W_0 with the word's average context vector. This is closer to what is being done in the training procedure and additionally it enables us to compute the embeddings for out-of-vocabulary words (assuming at least most of such a new word's context words are in the vocabulary) as well as to place more emphasis on a word's local context (which helps to identify the proper meaning of the word (Melamud et al., 2015)) by creating a weighted sum between the word's average global and local context vectors used as input to the ConEc.

With this new perspective on the model and optimization procedure, another advancement is feasible. Since the context words are merely a sparse feature vector used as input to a neural network, there is no reason why this input vector should not contain other features about the target word as well. For example, the feature vector could be extended to contain information about the word's case, part-of-speech (POS) tag, or other relevant details. While this would increase the dimensionality of the first weight matrix W_0 to include the additional features when mapping the input to the word's embedding, the training objective and therefore also W_1 would remain unchanged. These additional features could be especially helpful if details about the words would otherwise get lost in preprocessing (e.g. by lowercasing) or to retain information about a word's position in the sentence, which is ignored in a BOW approach. These extended ConEcs are expected to create embeddings which distinguish even better between the words' different senses by taking into account, for example, if the word is used as a noun or verb in the current context, similar to the sense2vec algorithm (Trask et al., 2015). However, unlike sense2vec, not multiple embeddings per term are learned, instead the dimensionality of the input vector is increased to include the POS tag of the current word as a feature.

3.1 EXPERIMENTS

The word embeddings learned with word2vec and context encoders are evaluated on a word analogy task (Mikolov et al., 2013a) as well as the CoNLL 2003 NER benchmark task (Tjong et al., 2003). The word2vec model used is a continuous BOW model trained with negative sampling as described above where $k = 13$, the embedding dimensionality d is 200 and we use a context window of 5. The word embeddings created by the context encoders are build directly on top of the word2vec model by multiplying the original embeddings (W_0) with the respective context vectors. Code to replicate the experiments can be found online.[7] The results of the analogy task can be found in the Appendix.[8]

Named Entity Recognition The main advantage of context encoders is that they can use local context to create out-of-vocabulary (OOV) embeddings and distinguish between the different senses of words. The effects of this are most prominent in a task such as named entity recognition (NER) where the local context of a word can make all the difference, e.g. to distinguish between the "Chicago Bears" (an organization) and the city of Chicago (a location). To test this, we used the word embeddings as features in the CoNLL 2003 NER benchmark task (Tjong et al., 2003). The word2vec embeddings were trained on the documents used in the training part of the task.[9] For the context encoders we experimented with different combinations of local and global context vectors. The global context vectors were computed on only the training documents as well, i.e. just as with

[7] `https://github.com/cod3licious/conec`

[8] As it was recently demonstrated that a good performance on intrinsic evaluation tasks such as word similarity or analogy tasks does not necessarily transfer to extrinsic evaluation measures when using the word embeddings as features (Chiu et al., 2016; Linzen, 2016), we consider the performance on the NER challenge as more relevant.

[9] Since this is a very small corpus, we trained word2vec for 25 iterations on these documents (afterwards the performance on the development split stopped improving significantly) while usually the model is trained in a single pass through a much larger corpus.

the word2vec model, when applied to the test documents there are some words which don't have a word embedding available as they did not occur in the training texts. The local context vectors on the other hand can be computed for all words occurring in the current document for which the model should identify the named entities. When combining these local context vectors with the global ones we always use the local context vector as is in case there is no global vector available and otherwise compute a weighted average between the two context vectors as $w_l \cdot \text{CV}_{\text{local}} + (1 - w_l) \cdot \text{CV}_{\text{global}}$.[10] The different word embeddings were used as features with a logistic regression classifier trained on the labels obtained from the training part of the task and the reported F1-scores were computed using the official evaluation script. Please note that we are using this task to show the potential of ConEc word embeddings as features in a real world task and to illustrate their advantages over the regular word2vec embeddings and did not optimize for competitive performance on this NER challenge.

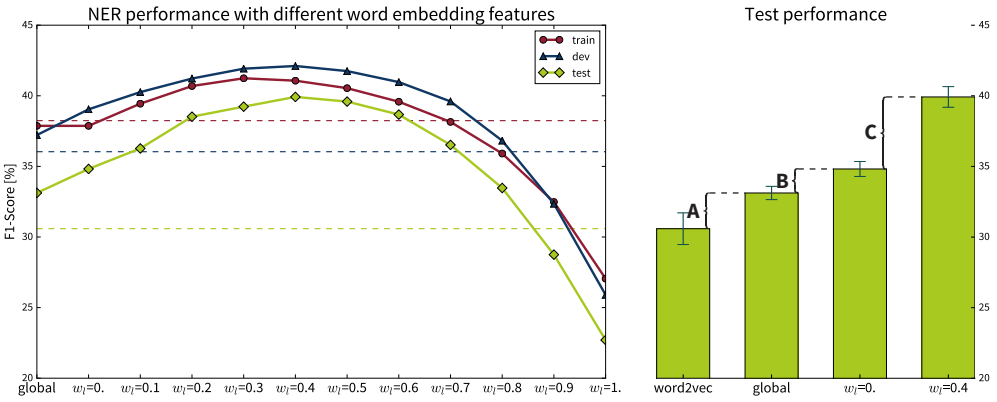

Figure 7: Results of the CoNLL 2003 NER task based on three random initializations of the word2vec model. The overall results are shown on the left, where the mean performance using word2vec embeddings is considered as our baseline indicated by the dashed lines, all other embeddings are computed with context encoders using various combinations of the words' global and local context vectors. On the right, the increased performance (mean and std) on the test fold achieved by using ConEc is highlighted: Enhancing the word2vec embeddings with global context information yields a performance gain of 2.5 percentage points (A). By additionally using local context vectors to create OOV word embeddings ($w_l = 0$) we gain another 1.7 points (B). When using a combination of global and local context vectors ($w_l = 0.4$) to distinguish between the different meanings of words, the F1-score increases by another 5.1 points (C), yielding a F1-score of 39.92%, which marks a significant improvement compared to the 30.59% reached with word2vec features.

Figure 7 shows the results achieved with various word embeddings on the training, development and test part of the CoNLL task. As it can be seen there, taking into account the local context can yield large improvements, especially on the dev and test data. Context encoders using only the global context vectors already perform better than word2vec. When using the local context vectors only where the global ones are not available ($w_l = 0$) we can see a jump in the development and test performance, while of course the training performance stays the same as here we have global context vectors for all words. The best performances on all folds are achieved when averaging the global and local context vectors with around $w_l = 0.4$ before multiplying them with the word2vec embeddings. This clearly shows that using ConEcs with local context vectors can be very beneficial as they let us compute word embeddings for out-of-vocabulary words as well as help distinguish between multiple meanings of words.

[10]The global context matrix is computed without taking the word itself into account (i.e. zero on the diagonal) to make the context vectors comparable to the local context vectors of OOV words where we can't count the target word either. Both global and local context vectors are normalized by their respective maximum values, then multiplied with the length normalized word2vec embeddings and again renormalized to have unit length.

4 CONCLUSION

Representing intrinsically complex data is an ubiquitous challenge in data analysis. While kernel methods and manifold learning have made very successful contributions, their ability to scale is somewhat limited. Neural autoencoders offer scalable nonlinear embeddings, but their objective is to minimize the reconstruction error of the input data which does not necessarily preserve important pairwise relations between data points. In this paper we have proposed SimEcs as a neural network framework which bridges this gap by optimizing the same objective as spectral methods, such as kPCA, for creating similarity preserving embeddings while retaining the favorable properties of autoencoders.

Similarity encoders are a novel method to learn similarity preserving embeddings and can be especially useful when it is computationally infeasible to perform the eigendecomposition of a kernel matrix, when the target similarities are obtained through an unknown process such as human similarity judgments, or when an explicit mapping function is required. To accomplish this, a feed-forward neural network is constructed to map the data into an embedding space where the original similarities can be approximated linearly.

As a second contribution we have defined context encoders, a practical extension of SimEcs, that can be readily used to enhance the word2vec model with further local context information and global word statistics. Most importantly, ConEcs allow to easily create word embeddings for out-of-vocabulary words on the spot and distinguish between different meanings of a word based its local context.

Finally, we have demonstrated the usefulness of SimEcs and ConEcs for practical tasks such as the visualization of data from different domains and to create meaningful word embedding features for a NER task, going beyond the capabilities of traditional methods.

Future work will aim to further the theoretical understanding of SimEcs and ConEcs and explore other application scenarios where using this novel neural network architecture can be beneficial. As it is often the case with neural network models, determining the optimal architecture as well as other hyperparameter choices best suited for the task at hand can be difficult. While so far we mainly studied SimEcs based on fairly simple feed-forward networks, it appears promising to consider also deeper neural networks and possibly even more elaborate architectures, such as convolutional networks, for the initial mapping step to the embedding space, as in this manner hierarchical structures in complex data could be reflected. Note furthermore that prior knowledge as well as more general error functions could be employed to tailor the embedding to the desired application target(s).

ACKNOWLEDGMENTS

We would like to thank Antje Relitz, Christoph Hartmann, Ivana Balažević, and other anonymous reviewers for their helpful comments on earlier versions of this manuscript. Additionally, Franziska Horn acknowledges funding from the Elsa-Neumann scholarship from the TU Berlin.

REFERENCES

Yoshua Bengio, Jean-François Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Le Roux, and Marie Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. *Advances in neural information processing systems*, 16:177–184, 2004.

Kerstin Bunte, Michael Biehl, and Barbara Hammer. A general framework for dimensionality-reducing data visualization mapping. *Neural Computation*, 24(3):771–804, 2012.

Billy Chiu, Anna Korhonen, and Sampo Pyysalo. Intrinsic evaluation of word vectors fails to predict extrinsic performance. *ACL 2016*, pp. 1, 2016.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. CRC Press, 2000.

Yoav Goldberg and Omer Levy. word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

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

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

Tal Linzen. Issues in evaluating semantic spaces using word analogies. *arXiv preprint arXiv:1606.07736*, 2016.

Oren Melamud, Ido Dagan, and Jacob Goldberger. Modeling word meaning in context with substitute vectors. In *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, 2015.

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

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013b.

Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *Neural Networks, IEEE Transactions on*, 12(2):181–201, 2001.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *EMNLP*, volume 14, pp. 1532–1543, 2014.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pp. 1177–1184, 2007.

Sascha Rothe and Hinrich Schütze. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. *arXiv preprint arXiv:1507.01127*, 2015.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

EF Tjong, Kim Sang, and F De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In Walter Daelemans and Miles Osborne (eds.), *Proceedings of CoNLL-2003*, pp. 142–147. Edmonton, Canada, 2003.

Andrew Trask, Phil Michalak, and John Liu. sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*, 2015.

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

Laurens van der Maaten. Learning a parametric embedding by preserving local structure. In *International Conference on Artificial Intelligence and Statistics*, pp. 384–391, 2009.

Laurens van der Maaten. Barnes-Hut-SNE. In *Proceedings of the International Conference on Learning Representations*, 2013.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

APPENDIX

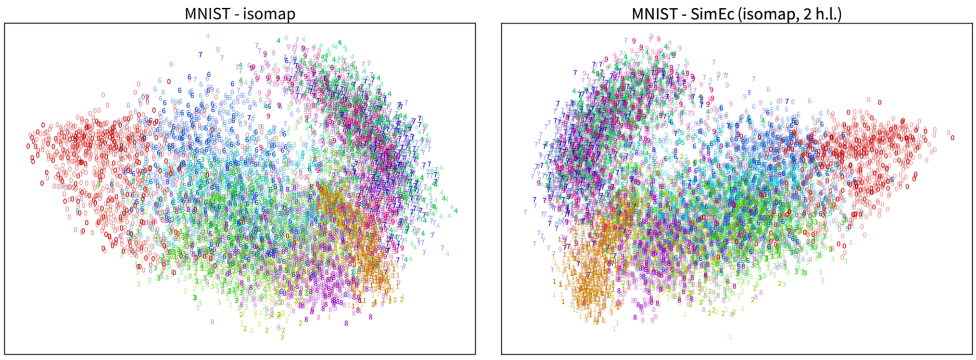

Figure 8: MNIST digits visualized in two dimensions by isomap and a non-linear SimEc.

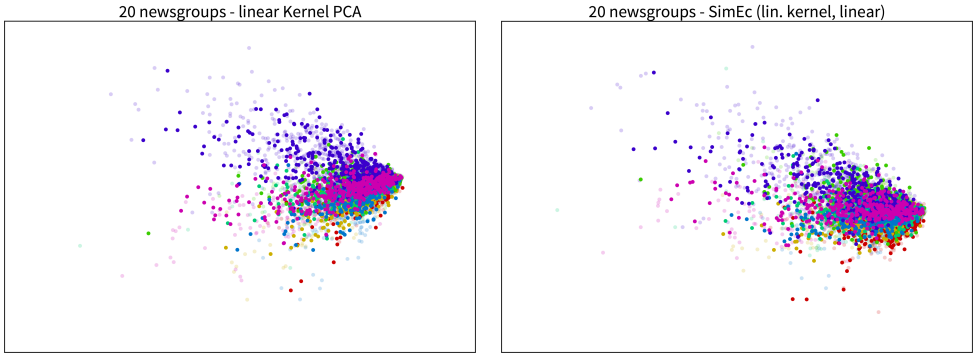

Figure 9: 20 newsgroups dataset embedded with linear kernel PCA and a corresponding linear SimEc.

Analogy task To show that the word embeddings created with context encoders capture meaningful semantic and syntactic relationships between words, we evaluated them on the original analogy task published together with the word2vec model (Mikolov et al., 2013a).[11] This task consists of many questions in the form of "*man* is to *king* as *woman* is to XXX" where the model is supposed to find the correct answer *queen*. This is accomplished by taking the word embedding for *king*, subtracting from it the embedding for *man* and then adding the embedding for *woman*. This new word vector should then be most similar (with respect to the cosine similarity) to the embedding for *queen*.[12] The word2vec and corresponding context encoder model are trained for ten iterations on the `text8` corpus,[13] which contains around 17 million words and a vocabulary of about 70k unique words, and the training part of the `1-billion` benchmark dataset,[14] which contains over 768 million words with a vocabulary of 486k unique words.[15]

The results of the analogy task are shown in Table 1. To capture some of the semantic relations between words (e.g. the first four task categories) it can be advantageous to use context encoders, i.e. to weight the word2vec embeddings with the words' average context vectors - however to achieve the best results we also had to include the target word itself in these context vectors. One reason for the ConEcs' superior performance on some of the task categories but not others might be that the city and country names compared in the first four task categories only have a single sense (referring to the

[11] See also `https://code.google.com/archive/p/word2vec/`.

[12] Readers familiar with Levy et al. (2015) will recognize this as the 3CosAdd method. We have tried 3CosMul as well, but found that the results did not improve significantly and therefore omitted them here.

[13] `http://mattmahoney.net/dc/text8.zip`

[14] `http://code.google.com/p/1-billion-word-language-modeling-benchmark/`

[15] In this experiment we ignore all words which occur less than 5 times in the training corpus.

Table 1: Accuracy on the analogy task with mean and standard deviation computed using three random seeds when initializing the word2vec model. The best results for each category and corpus are in bold.

	text8 (10 iter)		1-billion	
	word2vec	Context Encoder	word2vec	ConEc
capital-common-countries	63.8±4.7	**78.7**±0.2	79.3±2.2	**83.1**±1.2
capital-world	34.0±2.1	**54.7**±1.3	63.8±1.4	**75.9**±0.4
currency	15.4±0.9	**19.3**±0.6	13.3±3.6	**14.8**±0.8
city-in-state	28.6±1.0	**43.6**±0.9	19.6±1.7	**29.6**±1.0
family	**79.6**±1.5	77.2±0.4	78.7±2.2	**79.0**±1.4
gram1-adjective-to-adverb	11.0±0.9	**16.6**±0.7	12.3±0.5	**13.3**±1.1
gram2-opposite	24.3±3.0	24.3±2.0	**27.6**±0.1	21.3±1.1
gram3-comparative	**64.3**±0.5	63.0±1.1	**83.7**±0.9	76.2±1.1
gram4-superlative	**40.3**±2.1	37.6±1.5	**69.4**±0.5	56.2±1.2
gram5-present-participle	30.5±1.0	**31.7**±0.4	**78.4**±1.0	68.0±0.7
gram6-nationality-adjective	**70.6**±1.5	67.2±1.4	83.8±0.6	**83.8**±0.5
gram7-past-tense	30.5±1.8	**33.0**±0.6	**53.9**±0.9	49.2±0.7
gram8-plural	**49.8**±0.3	49.2±1.2	**62.7**±1.9	56.7±1.0
gram9-plural-verbs	**41.0**±2.5	30.1±1.9	**68.7**±0.2	45.0±0.4
total	42.1±0.6	**46.5**±0.1	**57.2**±0.3	55.8±0.3

respective location), while the words asked for in other task categories can have multiple meanings, for example "run" is used as both a verb and a noun and in some contexts refers to the sport activity while other times it is used in a more abstract sense, e.g. in the context of someone running for president. Therefore, the results in the other task categories might improve if the words' context vectors are first clustered and then the ConEc embedding is generated by multiplying with the average of only those context vectors corresponding to the word sense most appropriate for the task category.