An Exploration of Word Embedding Initialization in Deep-Learning Tasks

Tom Kocmi and **Ondřej Bojar**
Charles University,
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
surname@ufal.mff.cuni.cz

Abstract

Word embeddings are the interface between the world of discrete units of text processing and the continuous, differentiable world of neural networks. In this work, we examine various random and pretrained initialization methods for embeddings used in deep networks and their effect on the performance on four NLP tasks with both recurrent and convolutional architectures. We confirm that pretrained embeddings are a little better than random initialization, especially considering the speed of learning. On the other hand, we do not see any significant difference between various methods of random initialization, as long as the variance is kept reasonably low. High-variance initialization prevents the network to use the space of embeddings and forces it to use other free parameters to accomplish the task. We support this hypothesis by observing the performance in learning lexical relations and by the fact that the network can learn to perform reasonably in its task even with fixed random embeddings.

1 Introduction

Embeddings or lookup tables (Bengio et al., 2003) are used for units of different granularity, from characters (Lee et al., 2016) to subword units (Sennrich et al., 2016; Wu et al., 2016) up to words. In this paper, we focus solely on word embeddings (embeddings attached to individual token types in the text). In their highly dimensional vector space, word embeddings are capable of representing many aspects of similarities between words: semantic relations or morphological properties (Mikolov et al., 2013; Kocmi and Bojar, 2016) in one language or cross-lingually (Luong et al., 2015).

Embeddings are trained *for a task*. In other words, the vectors that embeddings assign to each word type are almost never provided manually but always discovered automatically in a neural network trained to carry out a particular task. The well known embeddings are those by Mikolov et al. (2013), where the task is to predict the word from its neighboring words (CBOW) or the neighbors from the given word (Skip-gram). Trained on a huge corpus, these "Word2Vec" embeddings show an interesting correspondence between lexical relations and arithmetic operations in the vector space. The most famous example is the following:

$$v(\textit{king}) - v(\textit{man}) + v(\textit{woman}) \approx v(\textit{queen})$$

In other words, adding the vectors associated with the words '*king*' and '*woman*' while subtracting '*man*' should be equal to the vector associated with the word '*queen*'. We can also say that the difference vectors $v(\textit{king}) - v(\textit{queen})$ and $v(\textit{man}) - v(\textit{woman})$ are almost identical and describe the gender relationship.

Word2Vec is not trained with a goal of proper representation of relationships, therefore the absolute accuracy scores around 50% do not allow to rely on these relation predictions. Still, it is a rather interesting property observed empirically in the learned space. Another extensive study of embedding space has been conducted by Hill et al. (2017).

Word2Vec embeddings as well as GloVe embeddings (Pennington et al., 2014) became very popular and they were tested in many tasks, also because for English they can be simply downloaded as pretrained on huge corpora. Word2Vec was trained on 100 billion words Google News

Proc. of ICON-2017, Kolkata, India. December 2017 ©*2017 NLPAI*, pages 56–64

dataset[1] and GloVe embeddings were trained on 6 billion words from the Wikipedia. Sometimes, they are used as a fixed mapping for a better robustness of the system (Kenter and De Rijke, 2015), but they are more often used to seed the embeddings in a system and they are further trained in the particular end-to-end application (Collobert et al., 2011; Lample et al., 2016).

In practice, random initialization of embeddings is still more common than using pretrained embeddings and it should be noted that pretrained embeddings are not always better than random initialization (Dhingra et al., 2017).

We are not aware of any study of the effects of various random embeddings initializations on the training performance.

In the first part of the paper, we explore various English word embeddings initializations in four tasks: neural machine translation (denoted MT in the following for short), language modeling (LM), part-of-speech tagging (TAG) and lemmatization (LEM), covering both common styles of neural architectures: the recurrent and convolutional neural networks, RNN and CNN, resp.

In the second part, we explore the obtained embeddings spaces in an attempt to better understand the networks have learned about word relations.

2 Embeddings initialization

Given a vocabulary V of words, *embeddings* represent each word as a dense vector of size d (as opposed to "one-hot" representation where each word would be represented as a sparse vector of size $|V|$ with all zeros except for one element indicating the given word). Formally, embeddings are stored in a matrix $E \in \mathbb{R}^{|V| \times d}$.

For a given word type $w \in V$, a row is selected from E. Thus, E is often referred to as word lookup table. The size of embeddings d is often set between 100 and 1000 (Bahdanau et al., 2014; Vaswani et al., 2017; Gehring et al., 2017).

2.1 Initialization methods

Many different methods can be used to initialize the values in E at the beginning of neural network training. We distinguish between randomly initialized and pretrained embeddings, where the latter can be further divided into embeddings pretrained on the same task and pretrained on a standard task such as Word2Vec or GloVe.

Random initialization methods common in the literature[2] sample values either uniformly from a fixed interval centered at zero or, more often, from a zero-mean normal distribution with the standard deviation varying from 0.001 to 10.

The parameters of the distribution can be set empirically or calculated based on some assumptions about the training of the network. The second approach has been done for various hidden layer initializations (i.e. not the embedding layer). E.g. Glorot and Bengio (2010) and He et al. (2015) argue that sustaining variance of values thorough the whole network leads to the best results and define the parameters for initialization so that the layer weights W have the same variance of output as is the variance of the input.

Glorot and Bengio (2010) define the "Xavier" initialization method. They suppose a linear neural network for which they derive weights initialization as

$$W \sim \mathcal{U}\big[-\frac{\sqrt{6}}{\sqrt{n_i + n_o}}; \frac{\sqrt{6}}{\sqrt{n_i + n_o}}\big] \qquad (1)$$

where n_i is the size of the input and n_o is the size of the output. The initialization for nonlinear networks using ReLu units has been derived similarly by He et al. (2015) as

$$W \sim \mathcal{N}(0, \frac{2}{n_i}) \qquad (2)$$

The same assumption of sustaining variance cannot be applied to embeddings because there is no input signal whose variance should be sustained to the output. We nevertheless try these initialization as well, denoting them *Xavier* and *He*, respectively.

2.2 Pretrained embeddings

Pretrained embeddings, as opposed to random initialization, could work better, because they already contain some information about word relations.

To obtain pretrained embeddings, we can train a randomly initialized model from the normal distribution with a standard deviation of 0.01, extract embeddings from the final model and use them as pretrained embeddings for the following trainings

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

[2] Aside from related NN task papers such as Bahdanau et al. (2014) or Gehring et al. (2017), we also checked several popular neural network frameworks (TensorFlow, Theano, Torch, ...) to collect various initialization parameters.

on the same task. Such embeddings contain information useful for the task in question and we refer to them as *self-pretrain*.

A more common approach is to download some ready-made "generic" embeddings such as Word2Vec and GloVe, whose are not directly related to the final task but show to contain many morpho-syntactic relations between words (Mikolov et al., 2013; Kocmi and Bojar, 2016). Those embeddings are trained on billions of monolingual examples and can be easily reused in most existing neural architectures.

3 Experimental setup

This section describes the neural models we use for our four tasks and the training and testing datasets.

3.1 Models description

For all our four tasks (MT, LM, TAG, and LEM), we use Neural Monkey (Helcl and Libovický, 2017), an open-source neural machine translation and general sequence-to-sequence learning system built using the TensorFlow machine learning library.

All models use the same vocabulary of 50000 most frequent words from the training corpus. And the size of embedding is set to 300, to match the dimensionality of the available pretrained Word2Vec and GloVe embeddings.

All tasks are trained using the Adam (Kingma and Ba, 2014) optimization algorithm.

We are using 4GB machine translation setup (MT) as described in Bojar et al. (2017) with increased encoder and decoder RNN sizes. The setup is the encoder-decoder architecture with attention mechanism as proposed by Bahdanau et al. (2014). We use encoder RNN with 500 GRU cells for each direction (forward and backward), decoder RNN with 450 conditional GRU cells, maximal length of 50 words and no dropout. We evaluate the performance using BLEU (Papineni et al., 2002). Because our aim is not to surpass the state-of-the-art MT performance, we omit common extensions like beam search or ensembling. Pretrained embeddings also prevent us from using subword units (Sennrich et al., 2016) or a larger embedding size, as customary in NMT. We experiment only with English-to-Czech MT and when using pretrained embeddings we modify only the source-side (encoder) embeddings, because there are no pretrained embeddings available for Czech.

The goal of the language model (LM) is to predict the next word based on the history of previous words. Language modeling can be thus seen as (neural) machine translation without the encoder part: no source sentence is given to translate and we only predict words conditioned on the previous word and the state computed from predicted words. Therefore the parameters of the neural network are the same as for the MT decoder. The only difference is that we use dropout with keep probability of 0.7 (Srivastava et al., 2014). The generated sentence is evaluated as the perplexity against the gold output words (English in our case).

The third task is the POS tagging (TAG). We use our custom network architecture: The model starts with a bidirectional encoder as in MT. For each encoder state, a fully connected linear layer then predicts a tag. The parameters are set to be equal to the encoder in MT, the predicting layer have a size equal to the number of tags. TAG is evaluated by the accuracy of predicting the correct POS tag.

The last task examined in this paper is the lemmatization of words in a given sentence (LEM). For this task we have decided to use the convolutional neural network, which is second most used architecture in neural language processing next to the recurrent neural networks. We use the convolutional encoder as defined by Gehring et al. (2017) and for each state of the encoder, we predict the lemma with a fully connected linear layer. The parameters are identical to the cited work. LEM is evaluated by a accuracy of predicting the correct lemma.

When using pretrained Word2Vec and GloVe embeddings, we face the problem of different vocabularies not compatible with ours. Therefore for words in our vocabulary not covered by the pretrained embeddings, we sample the embeddings from the zero-mean normal distribution with a standard deviation of 0.01.

3.2 Training and testing datasets

We use CzEng 1.6 (Bojar et al., 2016), a parallel Czech-English corpus containing over 62.5 million sentence pairs. This dataset already contains automatic word lemmas and POS tags.[3]

[3]We are aware that training and evaluating a POS tagger and lemmatizer on automatically annotated data is a little questionable because the data may exhibit artificial regularities and cannot lead to the best performance, but we assume

Initialization	MT en-cs (25M)	LM (25M)	RNN TAG (3M)	CNN LEM (3M)
$\mathcal{N}(0, 10)$	6.93 BLEU	76.95	85.2 %	48.4 %
$\mathcal{N}(0, 1)$	9.81 BLEU	61.36	87.9 %	94.4 %
$\mathcal{N}(0, 0.1)$	11.77 BLEU	56.61	90.7 %	95.7 %
$\mathcal{N}(0, 0.01)$	11.77 BLEU	56.37	**90.8 %**	**95.9 %**
$\mathcal{N}(0, 0.001)$	**11.88 BLEU**	**55.66**	90.5 %	**95.9 %**
Zeros	11.65 BLEU	56.34	90.7 %	**95.9 %**
Ones	10.63 BLEU	62.04	90.2 %	95.7 %
He init.	11.74 BLEU	56.40	90.7 %	95.7 %
Xavier init.	11.67 BLEU	55.95	**90.8 %**	**95.9 %**
Word2Vec	12.37 BLEU	**54.43**	90.9 %	95.7 %
Word2Vec on trainset	11.74 BLEU	54.63	90.8 %	95.6 %
GloVe	11.90 BLEU	55.56	90.6 %	95.5 %
Self pretrain	**12.61 BLEU**	54.56	**91.1 %**	**95.9 %**

Table 1: Task performance with various embedding initializations. Except for LM, higher is better. The best results for random (upper part) and pretrained (lower part) embedding initializations are in bold.

We use the `newstest2016` dataset from WMT 2016[4] as the testset for MT, LM and LEM. The size of the testset is 2999 sentence pairs containing 57 thousands Czech and 67 thousands English running words.

For TAG, we use manually annotated English tags from PCEDT[5] (Hajič et al., 2012). From this dataset, we drop all sentences containing the tag "-NONE-" which is not part of the standard tags. This leads to the testset of 13060 sentences of 228k running words.

4 Experiments

In this section, we experimentally evaluate embedding initialization methods across four different tasks and two architectures: the recurrent and convolutional neural networks.

The experiments are performed on the NVidia GeForce 1080 graphic card. Note that each run of MT takes a week of training, LM takes a day and a half and TAG and LEM need several hours each. We run the training for one epoch and evaluate the performance regularly throughout the training on the described test set. For MT and LM, the epoch amounts to 25M sentence pairs and for TAG and LEM to 3M sentences. The epoch size is set empirically so that the models already reach a stable level of performance and further improvement does not increase the performance too much.

MT and LM exhibit performance fluctuation throughout the training. Therefore, we average the results over five consecutive evaluation scores spread across 500k training examples to avoid local fluctuations. This can be seen as a simple smoothing method.[6]

4.1 Final performance

In this section, we compare various initialization methods based on the final performance reached in the individual tasks. Intuitively, one would expect the best performance with self-pretrained embeddings, followed by Word2Vec and GloVe. The random embeddings should perform worse.

Table 1 shows the influence of the embedding initialization on various tasks and architectures.

The rows $ones$ and $zeros$ specify the initialization with a single fixed value.

The "Word2Vec on trainset" are pretrained embeddings which we created by running Gensim (Řehůřek and Sojka, 2010) on our training set. This setup serves as a baseline for the embeddings pretrained on huge monolingual corpora and we can notice a small loss in performance compared to both Word2Vec and GloVe.

We can notice several interesting results. As expected, the self-pretrained embeddings slightly outperform pretrained Word2Vec and GloVe, which are generally slightly better than random initializations.

A more important finding is that there is generally no significant difference in the performance between different random initialization methods, except $ones$ and setups with the standard deviation of 1 and higher, all of which perform considerably worse.

that this difference will have no effect on the comparison of embeddings initializations and we prefer to use the same training dataset for all our tasks.

[4] http://www.statmt.org/wmt16/translation-task.html

[5] https://ufal.mff.cuni.cz/pcedt2.0/en/index.html

[6] See, e.g. `http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc42.htm` from Natrella (2010) justifying the use of the simple average, provided that the series has leveled off, which holds in our case.

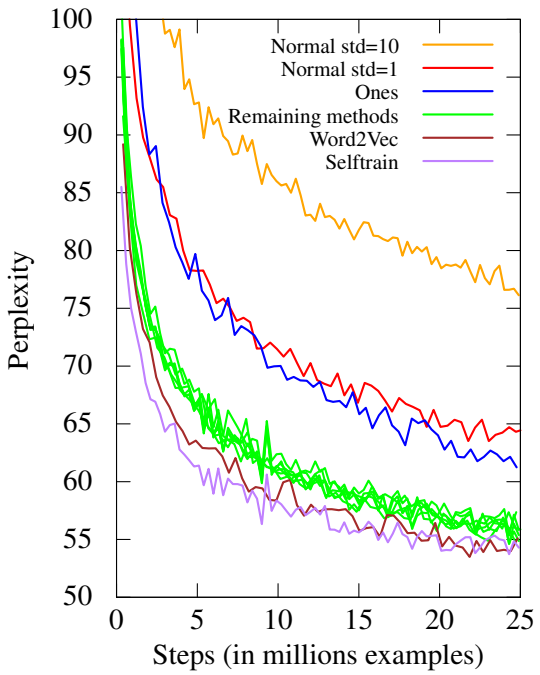

Figure 1: Learning curves for language modeling. The testing perplexity is computed every 300k training examples. Label "Remaining methods" represents all learning curves for the methods from Table 1 not mentioned otherwise.

Any random initialization with standard deviation smaller than 0.1 leads to similar results, including even the *zero* initialization.[7] We attempt to explain this behavior in Section 5.

4.2 Learning speed

While we saw in Table 1 that most of the initialization methods lead to a similar performance, the course of the learning is slightly more varied. In other words, different initializations need different numbers of training steps to arrive at a particular performance. This is illustrated in Figure 1 for LM.

To describe the situation concisely across the tasks, we set a minimal score for each task and we measure how many examples did the training need to reach the score. We set the scores as follows: MT needs to reach 10 BLEU points, LM needs to reach the perplexity of 60, TAG needs to reach the accuracy of 90% and LEM needs to reach the accuracy of 94%.

We use a smoothing window as implemented in TensorBoard with a weight of 0.6 to smooth the testing results throughout the learning. This way, we avoid small fluctuations in training and our estimate when the desired value was reached is more reliable.

[7] It could be seen as a surprise, that zero initialization works at all. But since embeddings behave as weights for bias values, they learn quickly from the random weights available throughout the network.

Initialization	MT en-cs	LM	TAG	LEM
$\mathcal{N}(0, 1)$	25.3M	37.3M	10.6M	2.7M
$\mathcal{N}(0, 0.1)$	9.7M	13.5M	2.0M	1.8M
$\mathcal{N}(0, 0.01)$	9.8M	12.0M	1.4M	1.2M
$\mathcal{N}(0, 0.001)$	9.8M	12.0M	1.0M	0.5M
Zeros	9.4M	12.3M	1.0M	**0.5M**
Ones	18.9M	26.7M	2.9M	0.8M
He init.	9.5M	12.5M	1.0M	**0.5M**
Xavier init.	9.2M	12.3M	1.0M	**0.5M**
Word2Vec	6.9M	7.9M	0.7M	1.2M
GloVe	8.6M	11.4M	1.9M	1.3M
Self pretrain	**5.2M**	**5.7M**	**0.2M**	0.9M

Table 2: The number of training examples needed to reach a desired score.

The results are in Table 2. We can notice that pretrained embeddings converge faster than the randomly initialized ones on recurrent architecture (MT, LM and TAG) but not on the convolutional architecture (LEM).

Self-pretrained embeddings are generally much better. Word2Vec also performs very well but GloVe embeddings are worse than random initializations for TAG.

5 Exploration of embeddings space

We saw above that pretrained embeddings are slightly better than random initialization. We also saw that the differences in performance are not significant when initialized randomly with small values.

In this section, we analyze how specific lexical relations between words are represented in the learned embeddings space. Moreover, based on the observations from the previous section, we propose a hypothesis about the failure of initialization with big numbers (*ones* or high-variance random initialization) and try to justify it.

The hypothesis is as follows:

The more variance the randomly initialized embeddings have, the more effort must the neural network exerts to store information in the embeddings space. Above a certain effort threshold, it becomes easier to store the information in the subsequent hidden layers (at the expense of some capacity loss) and use the random embeddings more or less as a strange "multi-hot" indexing mechanism. And on the other hand, initialization with a small variance or even all zeros leaves the neu-

ral network free choice over the utilization of the embedding space.

We support our hypothesis as follows.

- We examine the embedding space on the performance in lexical relations between words, If our hypothesis is plausible, low-variance embeddings will perform better at representing these relations.

- We run an experiment with non-trainable fixed random initialization to demonstrate the ability of the neural network to overcome broken embeddings and to learn the information about words in its deeper hidden layers.

5.1 Lexical relations

Recent work on word embeddings (Vylomova et al., 2016; Mikolov et al., 2013) has shown that simple vector operations over the embeddings are surprisingly effective at capturing various semantic and morphosyntactic relations, despite lacking explicit supervision in these respects.

The testset by Mikolov et al. (2013) contains "questions" defined as $v(X) - v(Y) + v(A) \sim v(B)$. The well-known example involves predicting a vector for word '*queen*' from the vector combination of $v(\textit{king}) - v(\textit{man}) + v(\textit{woman})$. This example is a part of "semantic relations" in the test set, called opposite-gender. The dataset contain another 4 semantic relations and 9 morphosyntactic relations such as pluralisation $v(\textit{cars}) - v(\textit{car}) + v(\textit{apple}) \sim v(\textit{apples})$.

Kocmi and Bojar (2016) revealed the sparsity of the testset and presented extended testset. Both testsets are compatible and we use them in combination.

Note that the performance on this test set is affected by the vocabulary overlap between the test set and the vocabulary of the embeddings; questions containing out-of-vocabulary words cannot be evaluated. This is the main reason, why we trained all tasks on the same training set and with the same vocabulary, so that their performance in lexical relations can be directly compared.

Another lexical relation benchmark is the word similarity. The idea is that similar words such as '*football*' and '*soccer*' should have vectors close together. There exist many datasets dealing with word similarity. Faruqui and Dyer (2014) have extracted words similarity pairs from 12 different corpora and created an interface for testing the embeddings on the word similarity task.[8]

Initialization	MT en-cs	LM	LEM
$\mathcal{N}(0, 10)$	0.0; 0.3	0.0; 0.3	0.0; 0.3
$\mathcal{N}(0, 1)$	0.0; 0.4	1.4; 3.5	0.0; 0.3
$\mathcal{N}(0, 0.1)$	1.2; 23.5	5.5; 15.2	0.0; 0.8
$\mathcal{N}(0, 0.01)$	2.0; 29.9	6.9; 19.4	0.1; 32.7
$\mathcal{N}(0, 0.001)$	2.1; 31.4	6.7; 18.2	0.3; 33.3
Zeros	1.6; 29.5	6.0; 17.5	0.2; 31.1
Ones	0.5; 16.6	5.3; 9.3	0.1; 31.0
He init.	1.4; 28.9	7.7; 18.3	0.1; 32.6
Xavier init.	1.5; 29.5	7.4; 18.2	0.1; 32.7
Word2Vec on trainset*	22.3; 48.9		
Word2Vec official*	81.3; 70.7		
GloVe official*	12.3; 60.1		

Table 3: The accuracy in percent on the (semantic; morphosyntactic) questions. We do not report TAG since its accuracy was less than 1% on all questions. *For comparison, we present results of Word2Vec trained on our training set and official trained embeddings before applying them on training of particular task.

Initialization	MT en-cs	LM	TAG	LEM
$\mathcal{N}(0, 10)$	3.3	2.2	3.6	2.6
$\mathcal{N}(0, 1)$	15.7	11.8	3.5	2.7
$\mathcal{N}(0, 0.1)$	56.7	32.7	6.9	2.8
$\mathcal{N}(0, 0.01)$	62.5	41.0	12.8	4.7
$\mathcal{N}(0, 0.001)$	59.3	37.4	12.1	2.4
Zeros	57.9	37.4	12.8	3.5
Ones	34.0	19.3	11.4	4.3
He init.	58.2	37.4	12.3	4.2
Xavier init.	58.3	37.5	12.3	2.7

Table 4: Spearman's correlation ρ on word similarities. The results are multiplied by 100.

When evaluating the task, we calculate the similarity between a given pair of words by the cosine similarity between their corresponding vector representation. We then report Spearmans rank correlation coefficient between the rankings produced by the embeddings against human rankings. For convenience, we combine absolute values of Spearman's correlations from all 12 Faruqui and Dyer (2014) testsets together as an average weighted by the number of words in the datasets.

The last type of relation we examine are the nearest neighbors. We illustrate on the TAG task how the embedding space is clustered when various initializations are used. We employ the Principal component analysis (PCA) to convert the embedding space of $|E|$ dimensions into two-dimensional space.

Table 3 reflects several interesting properties

[8] http://wordvectors.org/

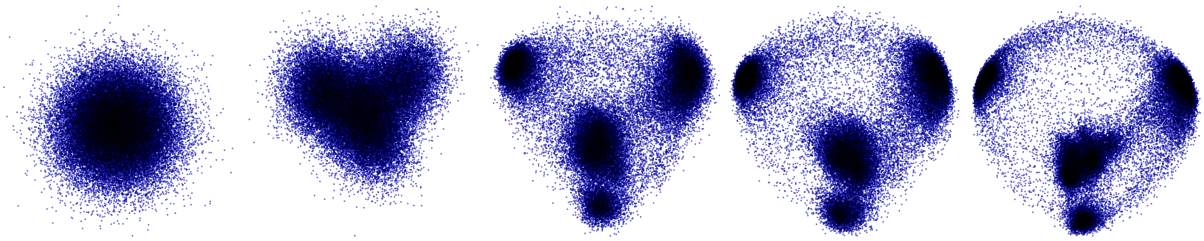

Figure 2: A representation of words in the trained embeddings for TAG task projected by PCA. From left to right it shows trained embeddings for $\mathcal{N}(0, 1)$, $\mathcal{N}(0, 0.1)$, $\mathcal{N}(0, 0.01)$, $\mathcal{N}(0, 0.001)$ and $zeros$. Note that except of the first model all of them reached a similar performance on the TAG task.

about the embedding space. We see task-specific behavior, e.g. TAG not learning any of the tested relationships whatsoever or LM being the only task that learned at least something of semantic relations.

The most interesting property is that when increasing the variance of initial embedding, the performance dramatically drops after some point. LEM reveals this behavior the most: the network initialized by normal distribution with standard deviation of 0.1 does not learn any relations but still performs comparably with other initialization methods as presented in Table 1. We ran the lemmatization experiments once again in order to confirm that it is not only a training fluctuation.

This behavior suggests that the neural network can work around broken embeddings and learn important features within other hidden layers instead of embeddings.

A similar behavior can be traced also in the word similarity evaluation in Table 4, where models are able to learn to solve their tasks and still not learn any information about word similarities in the embeddings.

Finally, when comparing the embedded space of embeddings as trained by TAG in Figure 2, we see a similar behavior. With lower variance in embeddings initialization, the learned embeddings are more clearly separated.

This suggests that when the neural network has enough freedom over the embeddings space, it uses it to store information about the relations between words.

5.2 Non-trainable embeddings

To conclude our hypothesis, we demonstrate the flexibility of a neural network to learn despite a broken embedding layer.

In this experiment, the embeddings are fixed and the neural network cannot modify them during the training process. Therefore, it needs to find a way to learn the representation of words in other hidden layers.

Initialization	MT en-cs	LM	TAG	LEM
$\mathcal{N}(0, 10)$	7.28 BLEU	79.44	47.3 %	85.5 %
$\mathcal{N}(0, 1)$	8.46 BLEU	78.68	87.1 %	94.0 %
$\mathcal{N}(0, 0.01)$	6.84 BLEU	82.84	63.2 %	91.1 %
Word2Vec	8.71 BLEU	60.23	88.4 %	94.1 %

Table 5: The results of the experiment when learned with non-trainable embeddings.

As in Section 4.1, we train models for 3M examples for TAG and LEM and for over 25M examples for MT and LM.

Table 5 confirms that the neural network is flexible enough to partly overcome the problem with fixed embeddings. For example, MT initialized with $\mathcal{N}(0, 1)$ reaches the score of 8.46 BLEU with fixed embeddings compared to 9.81 BLEU for the same but not fixed (trainable) embeddings.

When embeddings are fixed at random values, the effect is very similar to embeddings with high-variance random initialization. The network can distinguish the words through the crippled embeddings but has no way to improve them. It thus proceeds to learn in a similar fashion as with one-hot representation.

6 Conclusion

In this paper, we compared several initialization methods of embeddings on four different tasks, namely: machine translation (RNN), language modeling (RNN), POS tagging (RNN) and lemmatization (CNN).

The experiments indicate that pretrained embeddings converge faster than random initialization and that they reach a slightly better final performance.

The examined random initialization methods do not lead to significant differences in the performance as long as the initialization is within reasonable variance (i.e. standard deviation smaller than 0.1). Higher variance apparently prevents the network to adapt the embeddings to its needs and the network resorts to learning in its other free parameters. We support this explanation by showing that the network is flexible enough to overcome even non-trainable embeddings.

We also showed a somewhat unintuitive result that when the neural network is presented with embeddings with a small variance or even all-zeros embeddings, it utilizes the space and learns (to some extent) relations between words in a way similar to Word2Vec learning.

Acknowledgement

This work has been in part supported by the European Union's Horizon 2020 research and innovation programme under grant agreements No 644402 (HimL) and 645452 (QT21), by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (projects LM2015071 and OP VVV VI CZ.02.1.01/0.0/0.0/16_013/0001781), by the Charles University Research Programme "Progres" Q18+Q48, by the Charles University SVV project number 260 453 and by the grant GAUK 8502/2016.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *ICLR 2015*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. 2016. Czeng 1.6: Enlarged czech-english parallel corpus with processing tools dockered. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 231–238. Masaryk University, Springer International Publishing.

Ondřej Bojar, Jindřich Helcl, Tom Kocmi, Jindřich Libovický, and Tomáš Musil. 2017. Results of the WMT17 Neural MT Training Task. In *Proceedings of the 2nd Conference on Machine Translation (WMT)*, Copenhagen, Denmark, September.

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

Bhuwan Dhingra, Hanxiao Liu, Ruslan Salakhutdinov, and William W. Cohen. 2017. A comparative study of word embeddings for reading comprehension. *CoRR*, abs/1703.00993.

Manaal Faruqui and Chris Dyer. 2014. Community evaluation and exchange of word vectors at wordvectors.org. In *Proceedings of ACL: System Demonstrations*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing Prague Czech-English Dependency Treebank 2.0. In *Proceedings of the Eighth International Language Resources and Evaluation Conference (LREC'12)*, pages 3153–3160, Istanbul, Turkey, May. ELRA, European Language Resources Association.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Jindřich Helcl and Jindřich Libovický. 2017. Neural monkey: An open-source tool for sequence learning. *The Prague Bulletin of Mathematical Linguistics*, 107:5–17.

Felix Hill, Kyunghyun Cho, Sébastien Jean, and Yoshua Bengio. 2017. The representational geometry of word meanings acquired by neural machine translation models. *Machine Translation*, pages 1–16.

Tom Kenter and Maarten De Rijke. 2015. Short text similarity with word embeddings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1411–1420. ACM.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Tom Kocmi and Ondřej Bojar, 2016. *SubGram: Extending Skip-Gram Word Representation with Substrings*, pages 182–189. Springer International Publishing.

Guillaume Lample, Miguel Ballesteros, Kazuya Kawakami, Sandeep Subramanian, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *In proceedings of NAACL-HLT (NAACL 2016)*., San Diego, US.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2016. Fully character-level neural machine translation without explicit segmentation. *CoRR*, abs/1610.03017.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Bilingual word representations with monolingual quality in mind. In *NAACL Workshop on Vector Space Modeling for NLP*, Denver, United States.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mary Natrella. 2010. Nist/sematech e-handbook of statistical methods.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *ACL 2002, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.

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

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *ArXiv e-prints*, jun.

Ekaterina Vylomova, Laura Rimell, Trevor Cohn, and Timothy Baldwin. 2016. Take and took, gaggle and goose, book and read: Evaluating the utility of vector differences for lexical relation learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1671–1682, Berlin, Germany, August. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.