Tensorized Embedding Layers for Efficient Model Compression

Valentin Khrulkov [*1] **Oleksii Hrinchuk** [*1 2] **Leyla Mirvakhabova** [*1] **Ivan Oseledets** [1 3]

Abstract

The embedding layers transforming input words into real vectors are the key components of deep neural networks used in natural language processing. However, when the vocabulary is large (e.g., 800k unique words in the One-Billion-Word dataset), the corresponding weight matrices can be enormous, which precludes their deployment in a limited resource setting. We introduce a novel way of parametrizing embedding layers based on the Tensor Train (TT) decomposition, which allows compressing the model significantly at the cost of a negligible drop or even a slight gain in performance. Importantly, our method does not take the pre-trained model and compress its weights but rather supplants the standard embedding layers with their TT–based counterparts. The resulting model is then trained end-to-end, however, it can capitalize on larger batches due to the reduced memory requirements. We evaluate our method on a wide range of benchmarks in sentiment analysis, neural machine translation, and language modeling, and analyze the trade-off between performance and compression ratios for a wide range of architectures, from MLPs to LSTMs and Transformers.

1. Introduction

Deep neural networks (DNNs) typically used in natural language processing (NLP) employ large embeddings layers, which map the input words into continuous representations and usually have the form of lookup tables. Despite such simplicity and, arguably because of it, the resulting models are cumbersome, which may cause problems in training and deploying them in a limited resource setting. Thus, the compression of large neural networks and the development of novel lightweight architectures have become essential problems in NLP research.

One way to reduce the number of parameters in the trained model is to imply a specific structure on its weight matrices (e.g., assume that they are low-rank or can be well approximated by low-rank tensor networks). Such approaches are successful at compressing the pre-trained models, but they do not facilitate the training itself. Furthermore, they usually add to overall training time by requiring an additional fine-tuning phase as the compression algorithms usually optimize different objective functions.

In this paper, we introduce a new, parameter efficient embedding layer, termed TT–embedding, which can be plugged in into any model and trained end-to-end. The benefits of our compressed TT–layer are twofold. Firstly, instead of storing huge rectangular embedding matrix, we store a sequence of much smaller 2-dimensional and 3-dimensional tensors, necessary for reconstructing the required embeddings, which allows compressing the model significantly at the cost of a negligible performance drop. Secondly, the number of model parameters can be relatively small (and constant) during the whole training stage, which allows to use larger batches and train efficiently in a case of limited resources.

To validate the efficiency of the proposed approach, we have tested it on a variety of popular NLP tasks, namely sentiment analysis, neural machine translation, and language modeling. In our computational experiments, we have observed that in a majority of tasks, the standard embeddings can be replaced by TT–embeddings with the compression ratio of 2 or 3 orders without any significant drop (and sometimes even with a slight gain) of the metric of interest. Specifically, we report the following compression ratios of the embedding layers: 441 on the IMDB dataset with 0.2% absolute increase in classification accuracy; 57 on the WMT 2014 En–De dataset with 1.2 drop in the BLEU score, and 572 on the WikiText–103 dataset with 0.3 drop in perplexity.

Additionally, we have also evaluated our algorithm on a task of binary classification based on a large number of categorical features. More concretely, we applied TT–embedding to the click through rate (CTR) prediction problem, a crucial task in the field of digital advertising. Neural networks, typically used for solving this problem, while being rather

[*]Equal contribution [1]Skolkovo Institute of Science and Technology, Moscow, Russia [2]Moscow Institute of Physics and Technology, Moscow, Russia [3]Institute of Numerical Mathematics, Russian Academy of Sciences, Moscow, Russia. Correspondence to: Valentin Khrulkov <valentin.khrulkov@skoltech.ru>.

Under review as a conference paper at ICML 2019.

elementary, include a large number of embedding layers of significant size. As a result, a majority of model parameters that represent these layers, usually occupy hundreds of gigabytes of space. We show that TT–embedding not only considerably reduces the number of parameters in such models, but also sometimes improves their accuracy.

2. Related work

A number of prior works have explored different methods for compressing DNNs. (Sainath et al., 2013; Xue et al., 2013; Yu et al., 2017b) proposed to replace weight matrices in fully-connected layers with their low-rank approximations, obtained via truncated SVD. (Jaderberg et al., 2014) showed that using rank-1 decompositions of convolutional filters in the spatial domain led to significant compression and speed up on inference. (Kim et al., 2015; Howard et al., 2017) developed low-rank structural approximation with automatic selection of hyperparameters (e.g., ranks) for a specific purpose of deploying large multilayer neural networks on mobile devices. Other methods for DNNs compression include but not limited to pruning (Han et al., 2015b), quantization (Hubara et al., 2017; Xu et al., 2018), or their combination with Huffman coding (Han et al., 2015a).

In recent years, a large body of research was devoted to compressing and speeding up various components of neural networks used in NLP tasks. (Joulin et al., 2016) adapted the framework of product quantization to reduce the number of parameters in linear models used for text classification. (See et al., 2016) proposed to compress LSTM-based neural machine translation models with pruning algorithms. (Lobacheva et al., 2017) showed that the recurrent models could be significantly sparsified with the help of variational dropout (Kingma et al., 2015). (Chen et al., 2018b) proposed more compact K-way D-dimensional discrete encoding scheme to replace the "one-hot" encoding of categorical features, such as words in NLP taks. Very recently, (Chen et al., 2018a) and (Variani et al., 2018) introduced GroupReduce and WEST, two very efficient compression methods for the embedding and softmax layers, based on structured low-rank matrix approximation. Concurrently, (Lam, 2018) proposed the quantization algorithm for compressing word vectors and showed the superiority of the obtained embeddings on word similarity, word analogy, and question answering tasks.

Tensor methods have also been already successfully applied to neural networks compression. (Novikov et al., 2015) coined the idea of reshaping weights of fully-connected layers into high-dimensional tensors and representing them in Tensor Train (TT) (Oseledets, 2011) format. This approach was later extended to convolutional (Garipov et al., 2016) and recurrent (Yang et al., 2017; Tjandra et al., 2017; Yu et al., 2017a) neural networks. Furthermore, (Lebedev et al., 2014) showed that convolutional layers could be also compressed with canonical (CP) tensor decomposition (Carroll & Chang, 1970; Harshman, 1970). While all these methods allowed to reduce the number of parameters in the networks dramatically, they mostly capitalized on heavy fully-connected and convolutional layers (present in AlexNet (Krizhevsky et al., 2012) or VGG (Simonyan & Zisserman, 2014) networks), which became outdated in the following years. In this work, we show the benefits of applying tensor machinery to the compression of embedding layers, which are still widely used in NLP.

3. Tensor Train embedding

In this section, we briefly introduce the necessary notation and present the algorithm for constructing and training the TT–embedding layer. Hereinafter, by N-way tensor $\boldsymbol{\mathcal{X}}$ we mean a multidimensional array:

$$\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}.$$

with entries $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N)$, such that $\{0 \leq i_k < I_k\}_{k=1}^{N}$.

3.1. Motivation

Since most of the parameters in the NLP models occupy the embedding layers, we can greatly compress the entire model by compressing these layers, which is the problem we attack in this work. Our goal is to replace the standard embedding layer specified by an embedding matrix with a more compact, yet powerful and trainable, representation which would allow us to efficiently map words into vectors.

The simplest approach to compactly represent a matrix of a large size is to use the low–rank matrix factorization, which treats matrix $\mathbf{E} \in \mathbb{R}^{I \times J}$ as a product of two matrices $\mathbf{E} = \mathbf{U}\mathbf{V}^\top$. Here $\mathbf{U} \in \mathbb{R}^{I \times R}$ and $V \in \mathbb{R}^{J \times R}$ are much "thinner" matrices, and R is the rank hyperparameter. Note that rather than training the model with the standard embedding layer, and then trying to compress the obtained embedding, we can initially seek the embedding matrix in the described low–rank format. Then, for evaluation and training, the individual word embedding $\mathbf{E}[i, :]$ can be computed as a product $\mathbf{U}[i, :]\mathbf{V}^\top$ which does not require materializing the full matrix $\mathbf{E}$. This approach reduces the number of degrees of freedom in the embedding layer from IJ to $(I + J)R$.

However, typically, in the NLP tasks the embedding dimension J is much smaller than the vocabulary size I, and obtaining significant compression ratio using low-rank matrix factorization is problematic. In order to preserve the model performance, the rank R cannot be taken very small, and the compression ratio is bounded by $\frac{IJ}{(I+J)R} \leq \frac{J}{R}$, which is close to 1 for usually full-rank embedding matrix. To overcome this bound and achieve significant compression ratio even for matrices of disproportional dimensionalities,

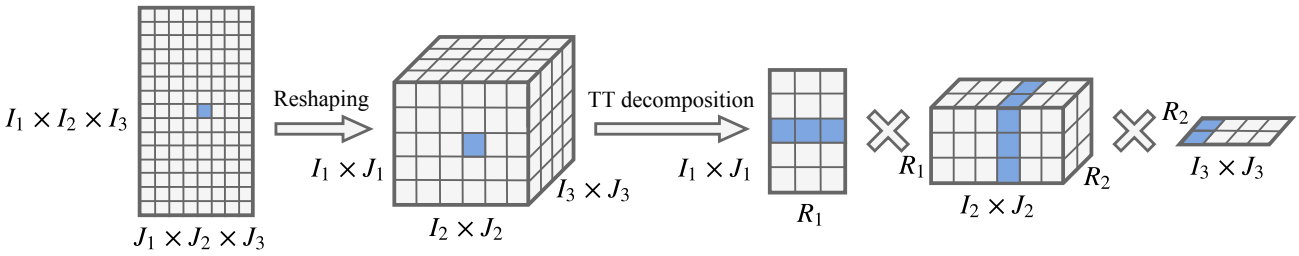

Figure 1. Construction of the TT–matrix from the standard embedding matrix. Blue color depicts how the single element in the initial matrix is transformed into the product of the highlighted vectors and matrices in the TT–cores.

we reshape them into multidimensional tensors and apply the *Tensor Train* decomposition, which allows for more compact representation, where the number of parameters falls down to logarithmic with respect to I.

3.2. Tensor Train decomposition

A tensor $\boldsymbol{\mathcal{X}}$ is said to be represented in the Tensor Train (TT) format (Oseledets, 2011) if each element of $\boldsymbol{\mathcal{X}}$ can be computed as:

$$\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_d) = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \boldsymbol{\mathcal{G}}^{(1)}(i_1, r_1) \boldsymbol{\mathcal{G}}^{(2)}(r_1, i_2, r_2) \ldots \boldsymbol{\mathcal{G}}^{(N)}(r_{N-1}, i_N),$$

where the tensors $\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ are the so-called *TT–cores* and $R_0 = R_N = 1$ by definition. The minimal values of $\{R_k\}_{k=1}^{N-1}$ for which the TT–decomposition exists are called *TT–ranks*. Note, that the element $\boldsymbol{\mathcal{X}}(i_1, i_2 \ldots i_N)$ is just effectively the product of 2 vectors and $N-2$ matrices:

$$\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N) = \underbrace{\boldsymbol{\mathcal{G}}^{(1)}[i_1, :]}_{1 \times R_1} \underbrace{\boldsymbol{\mathcal{G}}^{(2)}[:, i_2, :]}_{R_1 \times R_2} \ldots \underbrace{\boldsymbol{\mathcal{G}}^{(N-1)}[:, i_{N-1}, :]}_{R_{N-2} \times R_{N-1}} \underbrace{\boldsymbol{\mathcal{G}}^{(N)}[:, i_N]}_{R_{N-1} \times 1},$$

where $\boldsymbol{\mathcal{G}}^{(k)}[:, i_k, :]$ stands for the slice (a subset of a tensor with some indices fixed) of the corresponding TT–core $\boldsymbol{\mathcal{G}}^{(k)}$.

The number of degrees of freedom in such a decomposition can be evaluated to be $\sum_{k=1}^{N} R_k I_k R_{k+1}$. Thus, in the case of small ranks, the total number of parameters required to store a tensor in TT–representation is significantly smaller than $\prod_{k=1}^{N} I_k$ parameters required to store the full tensor of the corresponding size. This observation makes the application of the TT–decomposition appealing in many problems dealing with extremely large tensors.

TT–decomposition exists for any tensor (but it is not unique), however, compressing the tensor by a significant factor is only possible up to some relative error. To make use of this significant parameter reduction for tensors of low TT–ranks, in many practical problems it is common to seek a solution to a problem in the TT–format explicitly (i.e., via TT–cores only, without forming the full tensor), since it allows to perform many operations with low complexity with respect to hyperparameters of the decomposition. These operations include computing slices and performing basic linear operations on tensors.

3.3. TT–matrix

Let $\mathbf{X} \in \mathbb{R}^{I \times J}$ be a matrix of size $I \times J$. Given two arbitrary factorizations of its dimensions into natural numbers, $I = \prod_{k=1}^{N} I_k$ and $J = \prod_{k=1}^{N} J_k$, we can reshape[1] and transpose this matrix into an N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 J_1 \times I_2 J_2 \times \cdots \times I_N J_N}$ and then apply the TT–decomposition to it, resulting in a more compact representation.

More concretely, define the bijections $\boldsymbol{\mathcal{I}}(i) = (i_1, \ldots, i_N)$ and $\boldsymbol{\mathcal{J}}(j) = (j_1, \ldots, j_N)$ that map row and column indices i and j of the matrix X to the N-dimensional vector-indices such that $0 \le i_k < I_k$, $0 \le j_k < J_k$, $\forall k = 1, \ldots, N$. From the matrix $\mathbf{X}$ we can form an N-way tensor $\boldsymbol{\mathcal{X}}$ whose k-th dimension is of length $I_k J_k$ and is indexed by the tuple (i_k, j_k). This tensor is then represented in the TT–format:

$$\boldsymbol{\mathcal{X}}((i_1, j_1) \ldots (i_N, j_N)) = \boldsymbol{\mathcal{G}}^{(1)}[(i_1, j_1), :] \ldots \boldsymbol{\mathcal{G}}^{(N)}[:, (i_N, j_N)]. \quad (1)$$

Such representation of the matrix in the TT–format is called *TT–matrix* (Oseledets, 2010; Novikov et al., 2015) and is also known as Matrix Product Operator (MPO) (Pirvu et al., 2010) in physics literature. The factorizations $(I_1, I_2, \ldots I_N) \times (J_1, J_2, \ldots J_N)$ will be referred to as the *shape* of TT–matrix, or *TT–shapes*. The process of constructing the TT–matrix from the standard matrix is visualized in Figure 1 for the tensor of order 3. Note, that in this case the TT–cores are in face 4-th order tensors, but

[1] by reshape we mean a column-major `reshape` command, implemented, for example, as `numpy.reshape` in `Python`.

all the operations defined for tensors in the TT–format are naturally extended to TT–matrices.

3.4. TT–embedding

By *TT–embedding*, we call a layer with trainable parameters (TT–cores) represented as a TT–matrix $\boldsymbol{\mathcal{E}}$ of the underlying tensor shape $(I_1, I_2, \dots I_N) \times (J_1, J_2, \dots J_N)$, which can be transformed into a valid embedding layer $E \in \mathbb{R}^{I \times J}$, with $I = \prod_{k=1}^{N} I_k$ and $J = \prod_{k=1}^{N} J_k$. To specify the shapes of TT–cores one has also to provide the TT–ranks, which are treated as hyperparameters of the layer and explicitly define the total compression ratio.

In order to compute the embedding for a particular word indexed i in the vocabulary, we first map the row index i into the N-dimensional vector index $(i_1, \dots, i_N)$, and then calculate components of the embedding with formula (1). Note, that the computation of all its components is equivalent to selecting the particular slices in TT-cores (slices of shapes $J_1 \times R_1$ in $\boldsymbol{\mathcal{G}}^{(1)}$, $R_1 \times J_2 \times R_2$ in $\boldsymbol{\mathcal{G}}^{(2)}$ and so on) and performing a sequence of matrix multiplications, which is executed efficiently in modern linear algebra packages, such as cuBLAS. The procedure of computing the mapping $i \to (i_1, \dots, i_N)$ is given by Algorithm 1.

Algorithm 1 The algorithm implementing the bijection $\boldsymbol{\mathcal{I}}(i)$ as described in Section 3.3.

Require: I – vocabulary size, $\{I_k\}_{k=1}^{N}$ – an arbitrary factorization of I, i – index of the target word in vocabulary.
Returns: $\boldsymbol{\mathcal{I}}(i) = (i_1, \dots, i_N)$ – N-dimensional index.
Initialize: $L = \{1, I_1, I_1 I_2, \dots, I_1 I_2 \dots I_{N-1}\}$
for $k = N$ **to** 1 **do**
  $i_k \leftarrow \texttt{floor}(i / L[k])$
  $i \leftarrow i \mod L[k]$
end for

In order to construct TT–embedding layer for a vocabulary of size I and embedding dimension J, and to train a model with such a layer, one has to perform the following steps.

- Provide factorizations of I and J into factors $I = I_1 \times I_2 \times \dots \times I_N$ and $J = J_1 \times J_2 \times \dots \times J_N$, and specify the set of TT–ranks $\{R_1, R_2, \dots, R_{N-1}\}$.
- Initialize the set of parameters of the embedding $\boldsymbol{\Theta} = \{\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}\}_{k=1}^{N}$. Concrete initialization scenarios are discussed further in the text.
- During training, given a batch of indices $\{i_1, i_2, \dots i_b\}$, compute the corresponding embeddings $\{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_b\}$ using Eq. (1) and Algorithm 1.
- Computed embeddings can be followed by any standard layer such as LSTM (Hochreiter & Schmidhuber, 1997) or self-attention (Vaswani et al., 2017), and trained with backpropagation since they differentially depend on the parameters $\boldsymbol{\Theta}$.

TT–embedding implies a specific structure on the order of tokens in the vocabulary (the order of rows in the embedding matrix), and determining the optimal order is an appealing problem to solve. However, we leave this problem for future work and use the order produced by the standard tokenizer (sorted by frequency) in our current experiments.

Initialization The standard way to initialize an embedding matrix $\mathbf{E} \in \mathbb{R}^{I \times J}$ is via, e.g., Glorot initializer (Glorot & Bengio, 2010), which initializes each element as $\mathbf{E}(i,j) \sim \mathcal{N}\left(0, \frac{2}{I+J}\right)$. For the TT–embedding, we can only initialize the TT–cores, and the distribution of the elements of the resulting matrix $\boldsymbol{\mathcal{E}}$ is rather non–trivial. However, it is easy to verify that if we initialize each TT–core element as $\boldsymbol{\mathcal{G}}^{(k)}(r_{k-1}, i_k, r_k) \sim \mathcal{N}(0,1)$, the resulting distribution of the matrix elements $\boldsymbol{\mathcal{E}}(i,j)$ has the property that $\mathbb{E}[\boldsymbol{\mathcal{E}}(i,j)] = 0$ and $\mathrm{Var}[\boldsymbol{\mathcal{E}}(i,j)] = \prod_{k=1}^{N} R_k = R^2$. Capitalizing on this observation, in order to obtain the desired variance $\mathrm{Var}[\boldsymbol{\mathcal{E}}(i,j)] = \sigma^2$ while keeping $\mathbb{E}[\boldsymbol{\mathcal{E}}(i,j)] = 0$, we can simply initialize each TT–core as

$$\boldsymbol{\mathcal{G}}^{(k)}(r_{k-1}, i_k, r_k) \sim \mathcal{N}\left(0, \left(\frac{\sigma}{R}\right)^{2/N}\right). \qquad (2)$$

The resulting distribution is not Gaussian, however, it approaches the Gaussian distribution with the increase of the TT–rank (Figure 2).

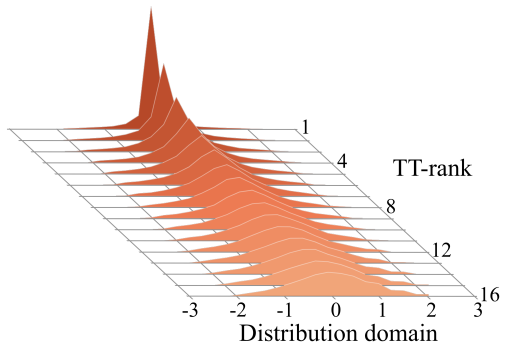

Figure 2. Distribution of a matrix element of the TT–matrix of shape $(5,5,5,5) \times (5,5,5,5)$, with cores initialized by formula (2) with $\sigma = 1$. As the TT–rank increases, the resulting distribution approaches $\mathcal{N}(0,1)$.

In our experiments, we have used the modified Glorot initializer implemented by formula (2), which greatly improved performance, as opposed to initializing TT–cores simply via a standard normal distribution. It is also possible to initialize TT–embedding layer by converting the learned embedding matrix into TT–format using the standard TT–SVD algorithm (Oseledets, 2011), however, this approach requires

the pretrained embedding matrix and does not exhibit better performance in practice.

Hyperparameter selection Our embedding layer introduces two additional structure-specific hyperparameters, namely *TT–shapes* and *TT–ranks*.

TT–embedding does not require the vocabulary size I to be represented *exactly* as the product of factors $I_1, \dots, I_N$, in fact, any factorization $\prod_{k=1}^{k} I_k = \widetilde{I} \geq I$ will suffice. However, in order to achieve the highest possible compression ratio for a fixed value of $\widetilde{I}$, the factors $\{I_k\}_{k=1}^{N}$ should be as close to each other as possible. Our implementation includes a simple automated procedure for selecting a good values of $\{I_k\}_{k=1}^{N}$ during TT–embedding initialization. The factors $J_1, \dots, J_N$ are defined by the embedding dimensionality J which can be easily chosen to support good factorization, e.g., $512 = 8 \times 8 \times 8$ or $480 = 6 \times 5 \times 4 \times 4$.

The values of TT–ranks directly define the compression ratio, so choosing them to be too small or too large will result into either significant performance drop or little reduction of the number of parameters. In our experiments, we set all TT–ranks to be equal to 16 for the problems with small vocabularies and 32 or 64 for the problems with larger vocabularies, which allowed us to achieve significant compression of the embedding layer, at the cost of a tiny sacrifice in the metrics of interest.

4. Experiments

Code We have implemented TT–embeddings described in Section 3 in `Python` using `PyTorch` (Paszke et al., 2017). The code is available at the repository https://github.com/KhrulkovV/tt-pytorch.

Experimental setup We tested our approach on several popular NLP tasks:

- **Sentiment analysis** — as a starting point in our experiments, we test TT–embeddings on a rather simple task of predicting polarity of a sentence.
- **Neural Machine Translation (NMT)** — to verify the applicability of TT–embeddings in more practical problems, we test it on a more challenging task of performing translation from one language to another.
- **Language Modeling (LM)** — finally, we evaluate TT–embeddings on language modeling tasks in the case of extremely large vocabularies.

Moreover, since our approach is not limited to NLP tasks but can also be applied to any problem possessing categorical features, we have performed the following experiment:

- **Click Through Rate (CTR) prediction** — we show that TT–embeddings can be successfully applied for the task of binary classification with numerous categorical features of significant cardinality.

In order to prove the generality and wide applicability of the proposed approach, we tested it on various popular architectures, such as MLPs (CTR), LSTMs (sentiment analysis), and Transformers (LM, NMT).

4.1. Sentiment analysis

Sentiment analysis is a classification task, where one has to predict whether the sequence of tokens (usually words or sentences) contains either positive or negative meaning. For this experiment, we have used the IMDB dataset (Maas et al., 2011) with two categories, and the Stanford Sentiment Treebank (SST) with five categories. We have taken the most frequent 25000 words for the IMDB dataset and 17200 for SST, embedded them into a J–dimensional space using either standard embedding or TT–embedding layer, and performed classification using a standard bidirectional two–layer LSTM with hidden size h, and dropout rate 0.5. For our experiments, we have set h to 128, and trained the model for various values of J and various TT-shapes (for TT–embedding).

Our findings are summarized in Table 1. We observe that the models with largely compressed embedding layers can perform equally or even better than the full uncompressed models. For instance, in the case of TT3 for the IMDB dataset, the number of parameters in the embedding layer was reduced from 6400000 to just 14496, while the test accuracy had not changed significantly. This suggests that learning individual independent embeddings for each particular word is superfluous, as the expressive power of LSTM is sufficiently large to make use of these intertwined, yet more compact embeddings. Moreover, slightly better test accuracy of the compressed models in certain cases (e.g., for the SST dataset of a rather small size) insinuates that imposing specific tensorial low–rank structure on the embedding matrix can be viewed as a special form of *regularization*, thus potentially improving the generalization power of the model. A detailed and comprehensive test of this hypothesis goes beyond the scope of this paper, and we leave it for future work.

4.2. Neural Machine Translation

In the task of Neural Machine Translation, the goal is to map an input sequence of symbols $\mathbf{x} = (x_1, x_2, \dots, x_n)$ representing a phrase in one language, to an output sequence $\mathbf{y} = (y_1, y_2, \dots, y_m)$ representing the same phrase in a different language. A typical architecture employed in this task is based on an encoder–decoder framework, which

Table 1. Sentiment analysis results. Embedding compression is calculated as the ratio between the number of parameters in the full embedding layer and TT–embedding layer. The LSTM parts are identical in both models, and the TT–ranks were set to 16 in these experiments. In IMDB experiments, we observe that both the best accuracy and the highest compression ratio are achieved with 6 TT–cores in TT–embedding layer. As for the SST dataset, the highest performance is attained with 3 TT–cores, while the best compression ratio is realized in the experiment with 6 TT–cores.

Dataset	Model	Embedding shape	Test acc.	Compr.
IMDB	Full	25000×256	0.886	1
	TT1	$(25, 30, 40) \times (4, 8, 8)$	0.871	93
	TT2	$(10, 10, 15, 20) \times (4, 4, 4, 4)$	0.886	232
	TT3	$(5, 5, 5, 5, 6, 8) \times (2, 2, 2, 2, 4, 4)$	**0.888**	**441**
SST	Full	17200×256	0.374	1
	TT1	$(24, 25, 30) \times (4, 8, 8)$	**0.415**	78
	TT2	$(10, 10, 12, 15) \times (4, 4, 4, 4)$	0.411	182
	TT3	$(4, 5, 5, 5, 6, 6) \times (2, 2, 2, 2, 4, 4)$	0.399	**307**

Table 2. Application of TT–embeddings to the task of English-to-German translation. WMT 14 En–De dataset was used for training, and news-commentary-11 for testing. The Transformer architecture ('base model') from (Vaswani et al., 2017) was used for this task. All layers except for embeddings are identical, and the models were trained using the same learning rate schedule, defined by Eq. (3) in (Vaswani et al., 2017). In these experiments, the embedding dimension was fixed to $J = 512$.

Model	Embedding shape	TT–rank	Test BLEU	Compr.
Full	31555×512	—	**30.97**	1
TT1	$(25, 32, 40) \times (8, 8, 8)$	16	29.08	219
TT2	$(25, 32, 40) \times (8, 8, 8)$	32	29.71	57
TT3	$(10, 10, 16, 20) \times (4, 4, 4, 8)$	32	28.67	143

maps the input sequence into continuous representations $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ and uses them for generating the output sequence $\mathbf{y}$, commonly also making use of the *attention mechanism*. The encoder–decoder framework serves as a foundation of most part of modern NMT models (Cho et al., 2014; Bahdanau et al., 2014; Sutskever et al., 2014; Vaswani et al., 2017).

For this experiment, we have trained the popular Transformer model (Vaswani et al., 2017) on WMT 2014 English–German dataset consisting of roughly 4.5 million sentence pairs. For validation, we used the news–commentary–v11 dataset. Sentences were tokenized using the SentencePiece[2] software, resulting in 31555 tokens for each language. As the baseline implementation of Transformer, we have used 'base model' architecture from (Vaswani et al., 2017) implemented in the OpenNMT–py[3] library (Klein et al., 2017), and for our experiments we have replaced each of the embedding layers with the corresponding TT–embedding. For evaluation we used beam search with a beam size of 5 and length penalty $\alpha = 0.6$.

[2] https://github.com/google/sentencepiece

[3] https://github.com/OpenNMT/OpenNMT-py

Our results are summarized in Table 2. We observe that even in this rather challenging task, both embedding layers can be compressed significantly, at the cost of a small drop in the BLEU score. Compared to the sentiment analysis, NMT is a much more complex task which benefits more from additional capacity (in the form of more powerful RNN or more transformer blocks) rather than regularization (Vaswani et al., 2017; Baevski & Auli, 2018), which may explain why we did not manage to improve the model by regularizing its embedding layers. However, note, that for a fixed memory budget, TT-embeddings allow to include more transformer blocks, which may lead to a more powerful model with the same number of parameters as in the full model with standard embedding layer.

4.3. Language modeling

The task of language modeling is to estimate the joint probability $P(\mathbf{x})$ of a corpus of tokens $\mathbf{x} = (x_1, \ldots, x_T)$, which resemble sentences, words, word pieces, or single characters. The resulting models can be used to generate text or further fine-tuned to solve other NLP tasks (Radford et al., 2018). In this paper, we employ the standard setting of predicting next token given the sequence of preceding tokens, based

Table 3. WikiText-103 language modeling results. The Transformer-XL architecture from (Dai et al., 2018) was used for this task. All layers except for embeddings are identical. The same training schedule was used for all the models. In these experiments, the embedding dimension was equal to $J = 480$, the number of transformer blocks $H = 16$, the number of attention heads $A = 10$.

Model	Embedding shape	TT–rank	Train ppl	Test ppl	Compr.
Full	267735×480	—	11.305	30.022	1
TT1	$(60, 60, 75) \times (6, 8, 10)$	32	18.442	25.849	243
TT2	$(20, 24, 25, 25) \times (4, 4, 5, 6)$	32	18.597	25.981	572
Full–tied	267735×480	—	16.742	**25.630**	—

Table 4. Criteo CTR results. The hashed dataset is constructed as specified in Section 4.4 with hashing value 10^5, and the unhashed dataset is considered as is. For the baseline algorithm, we have used the hashed version. Large embedding layers (with more than 2000 unique tokens) were replaced by TT–embedding layers with shape factorizations consisting of 3 or 4 factors. In the case of the full dataset, the compression ratio is measured with respect to the original dataset without hashing procedure. In these experiments, we took the TT–rank equal to 16 and the embedding dimension J is 36.

Hashing	Model	Factorization	Test loss	Compr.	Model size
10^5	Full	—	0.4440	1	157 Mb
	TT1	3 factors	**0.4433**	61	18 Mb
	TT2	4 factors	0.4440	**92**	**17** Mb
—	TT1	3 factors	0.4444	1004	20 Mb
	TT2	4 factors	**0.4438**	**2011**	18 Mb

on factorization $P(\mathbf{x}) = \prod_t P(x_t|\mathbf{x}_{<t})$. However, more complex scenarios can also be used, such as masking some words in the sentence and predicting them from the context or predicting next sentences from the previous ones (Devlin et al., 2018).

Specifically, we take the Transformer-XL (Dai et al., 2018), the open source[4] state-of-the-art language modeling architecture at the time of this writing, and replace the standard embedding layer with TT–embedding. Then, we test different model configurations on the WikiText–103 (Merity et al., 2016) dataset and report the results in Table 3.

We compare the model with distinct softmax and embedding layers (Full), the original Transformer-XL model (Full–shared) which ties softmax and embedding layers together as suggested in (Press & Wolf, 2016), and the models with TT–embeddings of different shapes. We see that the model with TT–embedding is superior to the full model which learns the embedding and softmax layers separately and overfits strongly to the training data. A simple modification which uses the same weight matrix in embedding and softmax layers (and can be seen as a form of regularization) performs much better. However, a larger difference between test and train perplexity suggests that it overfits more than the architecture with TT–embedding.

[4] https://github.com/kimiyoung/transformer-xl

4.4. Click Through Rate prediction

Among other applications of the TT–embedding layer, we chose to focus on the experiments lying in the field of click–through rate prediction, a popular task in digital advertising (He et al., 2014). In this paper, we consider the open dataset provided by Criteo for Kaggle Display Advertising Challenge (Criteo Labs, 2014). This dataset consists of 39 categorical features, 45840617 samples and is binary labeled according to whether the user clicked on the given advertisement. Unique values of categorical features are first bijectively mapped into integers. In order to reduce the amount of stored data, if the size of a corresponding vocabulary is immense (e.g., a cardinality of some features in this dataset is of order 10^6), these integers are further hashed by taking modulus with respect to some fixed number such as 10^5. However, due to strong compression properties of TT–embeddings, this is not necessary for our approach. In our experiments, we consider both full and hashed datasets.

CTR with the baseline algorithm The task at hand can be treated as a binary classification problem. As a baseline algorithm, we consider the neural network with the following architecture. First, each of the categorical features is passed through a separate embedding layer with embedding size J. After that, the embedded features are concatenated and passed through 4 fully-connected layers of

Table 5. The performance of lightweight models for CTR prediction on Criteo dataset. In these experiments, all TT–ranks were equal to 2, and hidden sizes were taken either equal to 128 or 256. Other parameters are the same as in the previous experiments (Table 4). We can observe the drop in accuracy paired, however, with an impressive compression ratio.

Hashing	Model	Hidden size h	Factorization	Test loss	Compr.	Model size
10^5	Full	128	—	**0.4443**	1	144 Mb
	TT1	128	3 factors	0.4515	2100	2.04 Mb
	TT2	128	4 factors	0.4530	**4193**	**2.01** Mb
10^5	Full	256	—	**0.4435**	1	145 Mb
	TT1	256	3 factors	0.4511	2100	3.15 Mb
	TT2	256	4 factors	0.4521	**4193**	**3.12** Mb

1024 neurons and ReLU activation functions. In all experiments, we use Adam optimizer with the learning rate equal to 0.0005. In this format, since many input features have a large number of unique values (e.g., 10131227) and storing the corresponding embedding matrices would require an immense amount of memory, we employ the hashing procedure mentioned earlier.

CTR with TT–embeddings Similarly, as in the previous experiments, we propose to substitute the embedding layers with the TT–embedding layers. Besides the embedding layers, we leave the overall structure of the neural network unchanged with the same parameters as in the baseline approach. Throughout our experiments, we consider a set of different TT–ranks and various factorizations.

Table 4 presents the experimental results on the Criteo CTR dataset. We have fixed the embedding dimension J equal to 36 and the TT–rank to 16. To the best of our knowledge, our loss value is very close to the state-of-the-art result (Juan et al., 2016). These experiments indicate that the substitution of large embedding layers with TT–embeddings leads to significant compression ratios (up to 2011 times) with a slight improvement in test loss. If we use the hashing procedure, the dataset is already compressed, which is in line with a smaller compressing power of TT–embedding layers. Nevertheless, the total size of the compressed model does not exceed 20 Mb, while the baseline model weighs about 160 Mb. The obtained compression ratio suggests that the usage of TT–embedding layers may be beneficial in CTR prediction tasks; however, rigorous evaluation on large industrial benchmarks would shed more light on this case.

Finally, to make the usage of the proposed method more applicable and practical, we have performed the experiments aiming to compress the model by a greater factor. Since these lightweight models are not as precise as larger ones, they can serve as preliminary prediction methods in the context of industrial purposes. We have considered the following parameters: the rank of underlying TT–matrix was equal to 2, and the hidden size was taken to be either 128 or 256 while leaving the remaining architecture untouched. The performance of these lightweight models is summarized in the Table 5.

5. Discussion and future work

We propose a novel embedding layer, the TT–embedding, for compressing huge lookup tables used for encoding categorical features of significant cardinality, such as the index of a token in natural language processing tasks. The proposed approach, based on the TT–decomposition, experimentally proved to be effective, as it heavily decreases the number of training parameters at the cost of a small deterioration in performance. In addition, our method can be easily integrated into any deep learning framework and trained via backpropagation, while capitalizing on reduced memory requirements and increased training batch size.

Our experimental results suggest several appealing directions for future work. First of all, TT–embeddings impose a concrete tensorial low-rank structure on the embedding matrix, which was shown to improve the generalization ability of the networks acting as a regularizer. The properties and conditions of applicability of this regularizer are subject to more rigorous analysis. Secondly, it is important to understand how the order of tokens in the vocabulary affects the properties of the networks with TT–embedding. We hypothesize that there exists the optimal order of tokens which better exploits the particular structure of TT–embedding and leads to a boost in performance and/or compression ratio. Additionally, another interesting direction is to determine the optimal number of factors of TT–cores as our extensive experiments demonstrate a slight dependence of total accuracy on the number of factors. Finally, the idea of applying higher–order tensor decompositions to reduce the number of parameters in neural nets is complementary to more traditional methods such as pruning and quantization. Thus, it would be interesting to make a thorough comparison of all these methods and investigate whether their combination may lead to even stronger compression.

Acknowledgements

We would like to thank Andrzej Cichocki for constructive discussions during the preparation of the manuscript. This work was supported by the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001).

References

Baevski, A. and Auli, M. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*, 2018.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Carroll, J. D. and Chang, J.-J. Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart-Young decomposition. *Psychometrika*, 35(3):283–319, 1970.

Chen, P. H., Si, S., Li, Y., Chelba, C., and Hsieh, C.-j. GroupReduce: Block-wise low-rank approximation for neural language model shrinking. *arXiv preprint arXiv:1806.06950*, 2018a.

Chen, T., Min, M. R., and Sun, Y. Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations. *arXiv preprint arXiv:1806.09464*, 2018b.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Criteo Labs. Kaggle Display Advertising Challenge, 2014. URL https://www.kaggle.com/c/criteo-display-ad-challenge.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-XL: Language modeling with longer-term dependency. *arXiv preprint arXiv:1901.02860*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Garipov, T., Podoprikhin, D., Novikov, A., and Vetrov, D. Ultimate tensorization: compressing convolutional and FC layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

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

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015b.

Harshman, R. A. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 1970.

He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pp. 1–9. ACM, 2014.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2017.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

Juan, Y., Zhuang, Y., Chin, W.-S., and Lin, C.-J. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 43–50. ACM, 2016.

Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. ACL*, 2017. doi: 10.18653/v1/P17-4012. URL https://doi.org/10.18653/v1/P17-4012.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Lam, M. Word2Bits-quantized word vectors. *arXiv preprint arXiv:1803.05651*, 2018.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., and Lempitsky, V. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Lobacheva, E., Chirkova, N., and Vetrov, D. Bayesian sparsification of recurrent neural networks. *arXiv preprint arXiv:1708.00077*, 2017.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Oseledets, I. V. Approximation of $2^d \times 2^d$ matrices using tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2130–2145, 2010.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NeurIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, 2017.

Pirvu, B., Murg, V., Cirac, J. I., and Verstraete, F. Matrix product operator representations. *New Journal of Physics*, 12(2):025012, 2010.

Press, O. and Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. 2018. URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6655–6659. IEEE, 2013.

See, A., Luong, M.-T., and Manning, C. D. Compression of neural machine translation models via pruning. *arXiv preprint arXiv:1606.09274*, 2016.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

Tjandra, A., Sakti, S., and Nakamura, S. Compressing recurrent neural network with tensor train. *arXiv preprint arXiv:1705.08052*, 2017.

Variani, E., Suresh, A. T., and Weintraub, M. WEST: Word Encoded Sequence Transducers. *arXiv preprint arXiv:1811.08417*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Xu, Y., Wang, Y., Zhou, A., Lin, W., and Xiong, H. Deep neural network compression with single and multiple level quantization. *arXiv preprint arXiv:1803.03289*, 2018.

Xue, J., Li, J., and Gong, Y. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pp. 2365–2369, 2013.

Yang, Y., Krompass, D., and Tresp, V. Tensor-train recurrent neural networks for video classification. *arXiv preprint arXiv:1707.01786*, 2017.

Yu, R., Zheng, S., Anandkumar, A., and Yue, Y. Long-term forecasting using tensor-train RNNs. *arXiv preprint arXiv:1711.00073*, 2017a.

Yu, X., Liu, T., Wang, X., and Tao, D. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7370–7379, 2017b.