Tensorized Embedding Layers

Oleksii Hrinchuk[1,2,*], **Valentin Khrulkov**[1,4,*], **Leyla Mirvakhabova**[1,*], **Elena Orlova**[1] & **Ivan Oseledets**[1,3]
[1] Skolkovo Institute of Science and Technology, Moscow, Russia
[2] Moscow Institute of Physics and Technology, Moscow, Russia
[3] Institute of Numerical Mathematics, RAS, Moscow, Russia
[4] Yandex.Research, Moscow, Russia

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, the corresponding weight matrices can be enormous, which precludes their deployment in a limited resource setting. We introduce a novel way of parameterizing embedding layers based on the Tensor Train decomposition, which allows compressing the model significantly at the cost of a negligible drop or even a slight gain in performance. We evaluate our method on a wide range of benchmarks in natural language processing 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, these layers often occupy a large portion of model weights 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 (Jaderberg et al., 2014). Such approaches are successful at compressing the pre-trained models, but they do not facilitate the training itself. Furthermore, they usually require an additional fine-tuning stage to recover the performance of the original model (Lebedev et al., 2015; Chen et al., 2018a).

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 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 overall number of parameters can be relatively small (and constant) during training.

The main contributions of our paper are:

- We propose to replace a standard dense embedding matrix with a novel compactly parameterized TT–embedding layer.
- We provide a theoretical justification of the proposed method from the softmax bottleneck (Yang et al., 2017b) perspective.
- We propose a novel initialization scheme for layers factorized with TT decomposition (Oseledets, 2010; Novikov et al., 2015).
- We evaluate TT–embedding on a variety of benchmarks in NLP and report better compression-accuracy trade-off than standard embedding and its low-rank decomposition.

2 Related work

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

[*] The first three authors contributed equally to this work

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 4847–4860

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). Cheong and Daniel (2019) successfully compressed the Transformer architecture with the combination of pruning and quantization.

There is a plethora of prior work on compressing the embedding layers used in NLP models. 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. Variani et al. (2018) introduced WEST, a compression method based on structured sparse and structured dense decomposition of the embedding matrix. Chen et al. (2018a) proposed to compress the pre-trained embedding matrix by capitalizing on the power-law distribution of words and using smaller dimensionality (lower rank) for the embeddings of less frequent words. Baevski and Auli (2018) used a similar idea in end-to-end fashion by training such structured low-rank embeddings from scratch. However, both of these methods rely on the assumption of power-law distribution of tokens and are not efficient when dealing with other popular tokenizations, such as wordpieces (Schuster and Nakajima, 2012; Wu et al., 2016) or BPEs (Sennrich et al., 2015). The effectiveness of simple low-rank factorized embeddings has been recently re-discovered by Lan et al. (2019), and we refer to this method as to important baseline. Also, Lam (2018) proposed a quantization algorithm for compressing word vectors, but its benefits are orthogonal to those of low-rank matrix and tensor factorizations and they can be used together, complementing each other.

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., 2017a; Tjandra et al., 2017; Yu et al., 2017) neural networks. Furthermore, Lebedev et al. (2015) showed that convolutional layers could be also compressed with canonical (CP) tensor decomposition (Carroll and Chang, 1970; Harshman, 1970). Finally, Wang et al. (2018) compressed both fully-connected and convolutional layers with Tensor Ring decomposition (Zhao et al., 2016). Recently, Ma et al. (2019) succesfully applied Block-Term Tensor Decomposition to the compression of self-attention modules in the Transformer (Vaswani et al., 2017) architecture. In this work, we show the benefits of applying tensor machinery to the compression of embedding layers, which are an essential component of all models used in NLP.

3 Motivation

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

In this section, we briefly discuss our motivation of using tensorized embedding layers instead of both standard embedding layers and their low-rank factorized counterpart.

3.1 Compression ratio perspective

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 $\mathbf{V} \in \mathbb{R}^{J \times R}$ 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 (see Figure 1 in (Chen et al., 2018b)). To overcome this bound and achieve significant compression ratio even for matrices of disproportional dimensionalities, we reshape them into multidimensional tensors and apply the *Tensor Train* decomposition, which allows for more compact representation with the number of parameters falling down to logarithmic with respect to I.

3.2 Softmax bottleneck perspective

We hypothesize that such tensorized embeddings are not only superior in terms of more efficient compression, but are more theoretically justified for the usage in NLP tasks than embedding layers based on matrix factorization. Our analysis is based on *softmax bottleneck* theory (Yang et al., 2017b) and the fact that modern NLP architectures typically use the same weights for both embedding and softmax layers (Press and Wolf, 2016; Inan et al., 2016).

This theory models a natural language as a collection of pairs of a context and its conditional next token distributions: $\mathcal{L} = \{(c_i, P_\theta(X|c_i)\}_{i=1}^N$, and considers parametric language models with a Softmax function operating on a context vector $\mathbf{h}_c$ and a word embedding $\mathbf{x}_i$ to define the conditional distribution $P_\theta(x|c)$. Given the number of context vectors N, the number of tokens M, and dimensionality of word embeddings d, the following three matrices are defined: $\mathbf{H}_\theta \in \mathbb{R}^{N\times d}$, $\mathbf{W}_\theta \in \mathbb{R}^{M\times d}$, $\mathbf{A} \in \mathbb{R}^{N\times M}$. The rows of these matrices correspond to context vectors, word embeddings, and log probabilities of the true data distribution respectively. Such language model attempts to approximate $\mathbf{A}$ (up to an addition of constant matrices corresponding to a degree of freedom in Softmax) in the form

$$\mathbf{A} = \mathbf{H}_\theta \mathbf{W}_\theta^\top. \tag{1}$$

Note that the rank of $\mathbf{H}_\theta \mathbf{W}_\theta^\top$ is bounded by d, while the matrix $\mathbf{A}$ is presumed to be a high rank matrix (Yang et al., 2017a), which provides an upper bound on *expressivity* of such models. Now, suppose that the matrix $\mathbf{W}_\theta$ is additionally factorized as $\mathbf{W}_\theta = \mathbf{U}_\theta \mathbf{V}_\theta^\top$ with some rank R. Then the rank of right-hand side of Equation (1) is bounded by R, which further reduces expressivity of such models. Contrary to this, we show that tensorized embeddings do not reduce expressivity in the softmax bottleneck sense — while the embedding matrix is compressed it still has *full matrix rank*. We provide a rigorous statement in Section 4.4 and verify benefits of tensorized embeddings over low-rank factorized ones empirically in Section 5.

4 Tensor Train embedding

In this section, we briefly introduce the necessary notation and present the algorithm for 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 $\{0 \le i_k < I_k\}_{k=1}^N$.

4.1 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:

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

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 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 as $\sum_{k=1}^N R_{k-1} I_k R_k$. 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.

4.2 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 $J = \prod_{k=1}^N J_k$,

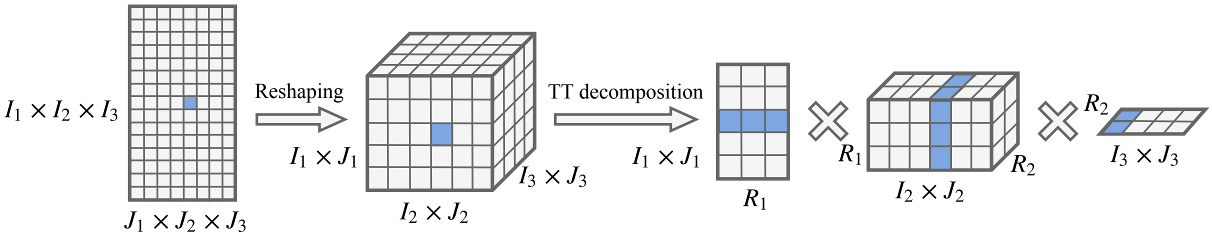

Figure 1: Construction of the TT–matrix from the standard embedding matrix. Blue color depicts how the single element in the initial matrix is transformed into the product of the highlighted vectors and matrices in the TT–cores.

we 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 $\boldsymbol{\mathcal{I}}(i) = (i_1, \dots, i_N)$ and $\boldsymbol{\mathcal{J}}(j) = (j_1, \dots, j_N)$ that map row and column indices i and j of the matrix X to the N-dimensional vector-indices such that $0 \leq i_k < I_k$, $0 \leq j_k < J_k$, $\forall k = 1, \dots, N$. 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:

$$\boldsymbol{\mathcal{X}}((i_1, j_1) \dots (i_N, j_N)) = \boldsymbol{\mathcal{G}}^{(1)}[(i_1, j_1), :] \dots \boldsymbol{\mathcal{G}}^{(N)}[:, (i_N, j_N)]. \quad (2)$$

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 (Pirvu et al., 2010) in physics literature. The factorizations $(I_1, I_2, \dots I_N) \times (J_1, J_2, \dots J_N)$ will be referred to as the *shape* of the TT–matrix, or *TT–shapes*. The construction of 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 fact 4-th order tensors as the indices are given by tuples (i_k, j_k), but all the operations defined for tensors in the TT–format are naturally extended to TT–matrices.

4.3 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 (2). 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 BLAS. Pseudocode for the procedure of computing the mapping $i \to (i_1, \dots, i_N)$ is given in Appendix A.

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 $I = I_1 \times I_2 \times \cdots \times I_N$ and $J = J_1 \times J_2 \times \cdots \times J_N$, 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 Equation (2).
- Computed embeddings can be followed by any standard layer such as LSTM (Hochreiter and 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

[1] by reshape we mean a column-major `reshape` command such as `numpy.reshape` in `Python`.

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.

We also experimented with a more general form of TT-decomposition, namely Tensor Ring (TR) decomposition (Zhao et al., 2016; Wang et al., 2018). This decomposition by construction has the appealing property of being circular permutation invariant (and, thus, more robust with respect to the order of the tokens), which could have potentially provided an improvement over the TT-based models with simple frequency based ordering. However, despite having stronger generalization abilities, TR might require more intricate optimization procedure (Section 2.5 in Grasedyck et al. (2013)), and we did not observe the benefits of using TR instead of TT in our experiments (Appendix C).

Initialization The standard way to initialize an embedding matrix $\mathbf{E} \in \mathbb{R}^{I \times J}$ is via, e.g., Glorot initializer (Glorot and 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 $\Sigma^2 := \mathrm{Var}[\boldsymbol{\mathcal{E}}(i,j)] = \prod_{k=1}^{N} R_k$. 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

$$\boldsymbol{\mathcal{G}}^{(k)}(r_{k-1}, i_k, r_k) \sim \mathcal{N}\left(0, \left(\frac{\sigma}{\Sigma}\right)^{2/N}\right). \quad (3)$$

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

In our experiments, we have used the modified Glorot initializer implemented by formula (3), 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 TT–SVD algorithm (Oseledets, 2011), however, this approach requires the pretrained embedding matrix and performs worse in practice (Garipov et al., 2016).

[2]Asymptotic normality is a consequence of application of the Central Limit Theorem.

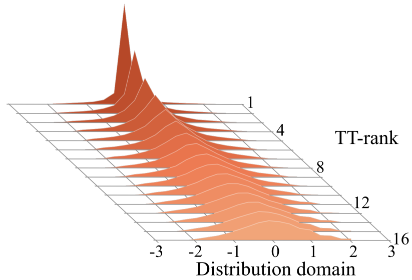

Figure 2: Distribution of matrix elements of the TT–matrix of shape $(5,5,5,5) \times (5,5,5,5)$ initialized by formula (3) with $\sigma = 1$. As the TT–rank increases, the resulting distribution approaches Gaussian $\mathcal{N}(0,1)$.

Hyperparameter selection TT–embedding 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 $I_1, \ldots, I_N$, in fact, any factorization $\prod_{k=1}^{N} I_k = \widetilde{I} \geq I$ 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 (Novikov et al., 2015; Yang et al., 2017a). Our implementation includes a simple automated procedure for selecting a good set of values $(\{I_k\}_{k=1}^{N}, \{J_k\}_{k=1}^{N})$ during TT–embedding initialization. The factors $J_1, \ldots, J_N$ 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 16 for problems with small vocabularies and $64-192$ for problems with larger vocabularies which resulted in a good trade-off between compression ratio and the metric of interest.

4.4 Expressivity of TT–embedding

Recall that in Section 3 we argued that one advantage of TT–embeddings is the property of being full rank matrices despite providing a significant data compression. Let us now formalize this statement.

For a fixed $I = \prod_{k=1}^{N} I_k$, $J = \prod_{k=1}^{N} J_k$, and a set of ranks $\mathbf{R} = (R_1, R_2, \ldots, R_{N-1})$, we consider $\mathcal{M}_{\mathbf{R}}$, the set of all tensors represented in the

TT-matrix format such that for any $\boldsymbol{\mathcal{X}} \in \mathcal{M}_{\mathbf{R}}$

$$\text{TT-rank}(\boldsymbol{\mathcal{X}}) \leq \mathbf{R},$$

entry-wise. Let $\mathbf{X}$ denote an ordinary matrix of size $N \times M$ obtained from the TT-matrix $\boldsymbol{\mathcal{X}}$ with the inverse of procedure decsribed in Section 4.2 (application of formulas from Section 4.1, followed by transposing and reshaping). We show that the following results holds true.

Theorem 1. *For all $\boldsymbol{\mathcal{X}} \in \mathcal{M}_{\mathbf{R}}$ besides a set of measure zero*

$$\operatorname{rank} \mathbf{X} = \min(I, J),$$

where the ordinary matrix rank is assumed.

See Appendix B for a proof.

This theorem states that for almost all TT-embeddings (besides a negligible set), the corresponding standard embedding matrix is full-rank. Thus, using the same matrix in the softmax layer, we can achieve significant compression without hitting the softmax bottleneck, as opposed to the low-rank matrix factorization.

5 Experiments

Code We have implemented TT–embeddings described in Section 4 in `Python` using `PyTorch` (Paszke et al., 2019). The code is available at the anonymous repository https://github.com/tt-embedding/tt-embeddings.

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 sentiment.
- **Neural Machine Translation (NMT)** — to verify the applicability of TT–embeddings in more practical problems, we test it on a more challenging task of machine translation.
- **Language Modeling (LM)** — then, we evaluate TT–embeddings on language modeling task in the case of extremely large vocabulary.
- **Click Through Rate (CTR) prediction** — finally, we show that TT–embeddings can be applied for the binary classification with categorical features of significant cardinality.

To prove the generality and wide applicability of the proposed approach, we tested it on various architectures, such as MLPs (CTR), LSTMs (sentiment analysis), and Transformers (NMT, LM). The baselines we compare with are

1. Standard embedding layer parametrized by a matrix $\mathbf{E} \in \mathbb{R}^{I \times J}$ with the baseline compression ratio of 1.
2. Low-rank factorized embedding layer parametrized by two matrices $\mathbf{U} \in \mathbb{R}^{I \times D}$ and $\mathbf{V} \in \mathbb{R}^{J \times D}$ such that the corresponding embedding matrix is $\mathbf{E} = \mathbf{U}\mathbf{V}^\top$. The compression ratio in this case is $\frac{I \times J}{(I+J) \times D} \approx \frac{J}{D}$.

Note that Transformers in LM and NMT use the same weight matrix for their embedding and softmax layers (Press and Wolf, 2016; Inan et al., 2016) which already significantly reduces model size. Untying weights and tensorizing the embedding layer only will lead to the increase in the number of parameters instead of compression. In our experiments, we use two separate TT-decompositions of the same shape for embedding and softmax layers and report the compression ratios as $\frac{|V| \times d_{\text{model}}}{2 \times \text{TT-params}}$.

5.1 Sentiment analysis

For this experiment, we have used the IMDB dataset (Maas et al., 2011) with two categories, and the Stanford Sentiment Treebank (SST) (Socher et al., 2013) 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 $h = 128$, and dropout rate $P_{\text{drop}} = 0.5$.

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. This suggests that learning individual independent embeddings for each particular word is superfluous, as the expressive power of LSTM is sufficient 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

Table 1: Sentiment analysis, LSTM on IMDB and SST datasets. 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.

Dataset	Model	Embedding shape	Test acc.	Emb compr.	Total params
IMDB	Full	25000×256	0.886	1	7.19M
	TT1	$(25, 30, 40) \times (4, 8, 8)$	0.871	93	0.86M
	TT2	$(10, 10, 15, 20) \times (4, 4, 4, 4)$	0.888	232	0.82M
	TT3	$(5, 5, 5, 5, 6, 8) \times (2, 2, 2, 2, 4, 4)$	**0.897**	441	0.81M
SST	Full	17200×256	0.374	1	5.19M
	TT1	$(24, 25, 30) \times (4, 8, 8)$	**0.415**	78	0.85M
	TT2	$(10, 10, 12, 15) \times (4, 4, 4, 4)$	0.411	182	0.82M
	TT3	$(4, 5, 5, 5, 6, 6) \times (2, 2, 2, 2, 4, 4)$	0.399	307	0.81M

Table 2: NMT, Transformer-big on WMT'14 English-to-German dataset. Both case-sensitive tokenized BLEU (higher is better) and de-tokenized SacreBLEU (Post, 2018) on newstest2014 are reported. In case of low-rank (LR) factorization, rank is the factorization rank; in case of TT-embedding (TT), rank is the TT-rank.

Model	Embedding shape	Rank	Token BLEU	Sacre BLEU	Emb compr.	Total params
Big	32768×1024	—	**29.58**	**28.84**	1	210M
Big+LR1	$(32768 \times 64),\ (64 \times 1024)$	64	28.98	28.26	15.5	179M
Big+LR2	$(32768 \times 32),\ (32 \times 1024)$	32	27.79	27.04	31	178M
Big+LR3	$(32768 \times 16),\ (16 \times 1024)$	16	24.80	24.12	62	177M
Big+TT1	$(32, 32, 32) \times (8, 8, 16)$	64	29.17	28.53	15.3	179M
Big+TT2	$(32, 32, 32) \times (8, 8, 16)$	48	28.53	27.97	26.8	178M
Big+TT3	$(32, 32, 32) \times (8, 8, 16)$	32	28.26	27.70	58.5	177M

model generalization. A detailed and comprehensive test of this hypothesis goes beyond the scope of this paper, and we leave it for future work.

5.2 Neural Machine Translation

For this experiment, we have trained the Transformer-big model ($d_{\text{model}} = 1024$, $d_{\text{ff}} = 4096$, $h = 16$) from Vaswani et al. (2017) on WMT 2014 English–German dataset consisting of roughly 4.5 million sentence pairs. We evaluated on newstest2014 dataset using beam search with a beam size of 4 and no length penalty. We did not employ checkpoint averaging and used the last checkpoint to compute the BLEU score. Sentences were tokenized with YouTokenToMe[3] byte-pair-encodings, resulting in a joint vocabulary of 32768 tokens. For the full list of hyperparameters, see the Appendix D.

[3]https://github.com/VKCOM/YouTokenToMe

Our results are summarized in Table 2. We observe that even in this rather challenging task, both embedding and softmax layers can be compressed significantly, at the cost of a small drop in the BLEU score. However, with the increase of compression factor, the performance deteriorates rapidly. 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 (Bahdanau et al., 2014; Vaswani et al., 2017; Wu et al., 2019), which may explain why we did not manage to improve the model by regularizing its embedding layers with TT-embedding.

Compared to the low-rank factorization of the embedding layer, the BLEU score of the Transformer with TT-embedding is higher and degrades much slower with the decrease of TT-rank. We hy-

Table 3: LM, Transformer-XL (Dai et al., 2019) on the WikiText-103 dataset. Lower perplexity (PPL) is better.

Model	Embedding shape	Rank	Valid PPL	Test PPL	Emb compr.	Total params
TXL	267735×512	—	**22.55**	**24.37**	1	192M
TXL+LR1	$(267735 \times 128),\ (128 \times 512)$	128	25.79	26.92	4	89M
TXL+LR1	$(267735 \times 96),\ (96 \times 512)$	96	26.57	27.75	5.3	81M
TXL+LR1	$(267735 \times 48),\ (48 \times 512)$	64	27.46	28.51	10.7	72M
TXL+TT1	$(60, 60, 75) \times (8, 8, 8)$	192	24.38	25.67	3.8	91M
TXL+TT2	$(60, 60, 75) \times (8, 8, 8)$	128	25.53	26.73	8.6	71M
TXL+TT3	$(60, 60, 75) \times (8, 8, 8)$	96	26.73	28.04	15.1	64M

pothesize that this is because of the corresponding embedding matrix being full rank and not suffering from the softmax bottleneck (Yang et al., 2017b).

TT-embeddings induce 8% training iteration time overhead if compared to the baseline Transformer-big due to our current implementation heavily relying on slow `torch.einsum` function while standard embedding and softmax layers make use of fast and highly-optimized Tensor Cores for mixed-precision training. We expect a dedicated CUDA kernel to be much more efficient.

5.3 Language modeling

We took the Transformer-XL (Dai et al., 2019), an open source[4] state-of-the-art language modeling architecture at the time of this writing, and replaced its embedding and softmax layers with TT–factorizations. Then, we tested different model configurations on WikiText–103 (Merity et al., 2016) and reported the results in Table 3. For the full list of hyperparameters, see the Appendix D.

Compared to sentiment analysis and NMT, we were not able to achieve that high compression ratios for embedding and softmax layers in LM. However, in our case of extremely large vocabulary (≈ 270000 words), even moderate 3.8 times compression allowed us to save 100M of weights at the cost of ~ 1.5 perplexity drop. Note that TT-embeddings also outperform low-rank factorization achieving better trade-off between compression and the performance.

6 Click Through Rate prediction

Among other applications of the TT–embedding layer, we chose to focus on CTR prediction, a popular task in digital advertising (He et al., 2014). We consider open dataset provided by Criteo for Kaggle Display Advertising Challenge (Criteo Labs, 2014) which consists of 39 categorical features, 45.8M samples and is binary labeled according to whether the user clicked on the given advertisement. Unique values of categorical features are bijectively mapped into integers. To reduce the memory footprint, 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, and we consider both full and hashed datasets in our experiments.

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 1024 neurons and ReLU activation functions. In all experiments, we used Adam optimizer with the learning rate equal to 0.0005. Since many input features have a large number of unique values (e.g., 10131227) and storing the corresponding embedding matrices would be costly, we employ the hashing procedure mentioned earlier.

CTR with TT–embeddings We substitute the embedding layers with the TT–embedding layers. Besides that, we leave the overall structure of the neural network unchanged with the same parameters as in the baseline approach. Table 4 presents

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

Table 4: CTR prediction. The hashed dataset is constructed with hashing value 10^5. Embedding layers with more than 2000 unique tokens were replaced by TT–embeddings.

Hash	Model	Factorization	TT rank	Hidden size	Test loss	Emb. compr.	Total params
10^5	Full	—	—	1024	0.4440	1	41.2M
	TT1	3 factors	16	1024	**0.4433**	61	4.7M
	TT2	4 factors	16	1024	0.4440	92	4.5M
	TT3	3 factors	2	128	0.4515	2100	0.53M
	TT4	4 factors	2	128	0.4530	4193	0.53M
—	TT1	3 factors	16	1024	0.4444	1004	5.2M
	TT2	4 factors	16	1024	**0.4438**	2011	4.7M

the experimental results on the Criteo CTR dataset. 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 the test loss, and up to 4200 with a small drop in the test loss. 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.

7 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, unlike standard embedding, we can introduce non-linearity into TT-cores to improve their expressive power (Khrulkov et al., 2019). Additionally, 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. 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 (Han et al., 2015) and quantization (Hubara et al., 2017; Xu et al., 2018). 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.

Acknowledgments

We would like to thank Andrzej Cichocki for constructive discussions during the preparation of the manuscript and anonymous reviewers for their valuable feedback. This work was funded by the Ministry of Science and Education of Russian Federation as a part of Mega Grant Research Project 14.756.31.000.

References

Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Weronika Buczyńska, Jarosław Buczyński, and Mateusz Michałek. 2015. The hackbusch conjecture on tensor formats. *Journal de Mathématiques Pures et Appliquées*, 104(4):749–761.

J Douglas Carroll and Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart-Young decomposition. *Psychometrika*, 35(3).

Patrick H Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-jui Hsieh. 2018a. GroupReduce: Block-wise low-rank approximation for neural language model shrinking. *NIPS*.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018b. Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations. *arXiv preprint arXiv:1806.09464*.

Robin Cheong and Robel Daniel. 2019. transformers. zip: Compressing transformers with pruning and quantization. Technical report, Technical report, Stanford University, Stanford, California, 2019. URL https ....

Criteo Labs. 2014. Kaggle Display Advertising Challenge.

Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. 2016. Ultimate tensorization: compressing convolutional and FC layers alike. *arXiv preprint arXiv:1611.03214*.

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.

Lars Grasedyck, Daniel Kressner, and Christine Tobler. 2013. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143.

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

Robin Hartshorne. 2013. *Algebraic geometry*, volume 52. Springer Science & Business Media.

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM.

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

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*.

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

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM.

Valentin Khrulkov, Oleksii Hrinchuk, and Ivan Oseledets. 2019. Generalized tensor models for recurrent neural networks. *arXiv preprint arXiv:1901.10801*.

Durk P Kingma, Tim Salimans, and Max Welling. 2015. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2015. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *ICLR*.

Ekaterina Lobacheva, Nadezhda Chirkova, and Dmitry Vetrov. 2017. Bayesian sparsification of recurrent neural networks. *arXiv preprint arXiv:1708.00077*.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Dawei Song, and Ming Zhou. 2019. A tensorized transformer for language modeling. *arXiv preprint arXiv:1906.09777*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450.

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

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Bogdan Pirvu, Valentin Murg, J Ignacio Cirac, and Frank Verstraete. 2010. Matrix product operator representations. *New Journal of Physics*, 12(2):025012.

Matt Post. 2018. A call for clarity in reporting bleu scores. *arXiv:1804.0877*.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE.

Abigail See, Minh-Thang Luong, and Christopher D Manning. 2016. Compression of neural machine translation models via pruning. *arXiv preprint arXiv:1606.09274*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. 2017. Compressing recurrent neural network with tensor train. *arXiv preprint arXiv:1705.08052*.

Ehsan Variani, Ananda Theertha Suresh, and Mitchel Weintraub. 2018. WEST: Word Encoded Sequence Transducers. *arXiv preprint arXiv:1811.08417*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. 2018. Wide compression: Tensor ring nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9329–9338.

Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*.

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*.

Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. 2018. Deep neural network compression with single and multiple level quantization. *arXiv preprint arXiv:1803.03289*.

Yinchong Yang, Denis Krompass, and Volker Tresp. 2017a. Tensor-train recurrent neural networks for video classification. *arXiv preprint arXiv:1707.01786*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. 2017b. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*.

Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. 2017. Long-term forecasting using tensor-train RNNs. *arXiv preprint arXiv:1711.00073*.

Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. 2016. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*.

A Multiindex construction

Algorithm 1 The algorithm implementing the bijection $\mathcal{I}(i)$ as described in Section 4.2.

Require: I – vocabulary size, $\{I_k\}_{k=1}^N$ – an arbitrary factorization of I,
i – index of the target word in vocabulary.
Returns: $\mathcal{I}(i) = (i_1, \ldots, i_N)$ – N-dimensional index.
Initialize: $L = \{1, I_1, I_1 I_2, \ldots, I_1 I_2 \ldots I_{N-1}\}$

for $k = N$ **to** 1 **do**
  $i_k \leftarrow \texttt{floor}(i/L[k])$
  $i \leftarrow i \mod L[k]$
end for

Algorithm 2 The algorithm implementing the bijection $(i_1, \ldots, i_N) \to i$, inverse to $\mathcal{I}(i)$.

Require: I – vocabulary size, $\{I_k\}_{k=1}^N$ – an arbitrary factorization of I,
$(i_1, \ldots, i_N)$ – N-dimensional index.
Returns: i – index of the target word in vocabulary
Initialize: $L = \{1, I_1, I_1 I_2, \ldots, I_1 I_2 \ldots I_{N-1}\}$

$i \leftarrow 0$
for $k = 1$ **to** N **do**
  $i \leftarrow i + i_k \times L[k]$
end for

B Proof of Theorem 1

Recall that for fixed $I = \prod_{k=1}^N I_k$, $J = \prod_{k=1}^N J_k$, and a set of ranks $\mathbf{R} = (R_1, R_2, \ldots, R_{N-1})$ we defined $\mathcal{M}_\mathbf{R}$, the set of all tensors represented in the TT-matrix format such that for any $\boldsymbol{\mathcal{X}} \in \mathcal{M}_\mathbf{R}$ we have

$$\text{TT-rank}(\boldsymbol{\mathcal{X}}) \leq \mathbf{R},$$

entry-wise. Let $\mathbf{X}$ denote an ordinary matrix of size $N \times M$ obtained from the TT-matrix $\boldsymbol{\mathcal{X}}$ with the inverse of procedure decsribed in Section 4.2 (application of formulas from Section 4.1, followed by transposing and reshaping).

Our analysis is based on the fact that $\mathcal{M}_\mathbf{R}$ forms an *irreducible algebraic set* (Buczyńska et al., 2015; Hartshorne, 2013). Concretely, we will use the fact that for an irreducible algebraic set $\mathcal{A}$ any algebraic subset $\mathcal{B}$ either has measure zero, or coincides with $\mathcal{A}$. We start with a simple lemma.

Lemma 1. *Let*

$$\mathcal{B} = \{\boldsymbol{\mathcal{X}} \in \mathcal{M}_\mathbf{R} \colon \operatorname{rank} \mathbf{X} < \min(I, J)\},$$

then $\mathcal{B}$ is an algebraic subset of $\mathcal{M}_\mathbf{R}$.

Proof. We need to show that $\mathcal{B}$ is cut out by polynomial equations on $\mathcal{M}_\mathbf{R}$. This readily follows from the facts that $\operatorname{mat}(\cdot)$ is a linear mapping, and that the upper bound on matrix rank can be specified by requiring all minors of specific size to vanish (which is a polynomial constraint). □

We now show that $\mathcal{B}$ is in fact a proper subset of $\mathcal{M}_\mathbf{R}$, i.e., $\mathcal{B} \subsetneq \mathcal{M}_\mathbf{R}$.

Lemma 2. *For any $\mathcal{M}_\mathbf{R}$ there exists $\boldsymbol{\mathcal{X}} \in \mathcal{M}_\mathbf{R}$ with*

$$\operatorname{rank} \mathbf{X} = \min(I, J).$$

Proof. We provide a concrete example of such a tensor. Define the collection of TT–cores $\{\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{1 \times I_k \times J_k \times 1}\}_{k=1}^N$ using the equations

$$\boldsymbol{\mathcal{G}}^{(k)}[:, (i, j), :] = \delta_{ij}, \tag{4}$$

with δ_{ij} denoting the Kronecker delta symbol. It easy to verify that $\mathbf{X}$ of a tensor $\boldsymbol{\mathcal{X}}$ specified by this collection of cores takes a very simple form: $\mathbf{X}[i, j] = \delta_{ij}$, which clearly is of maximal rank. □

Using Lemmas 1 and 2 and based on previous discussion on properties of algebraic sets we conclude that the following theorem holds.

Theorem 1. *For all $\boldsymbol{\mathcal{X}} \in \mathcal{M}_\mathbf{R}$ besides a set of measure zero*

$$\operatorname{rank} \mathbf{X} = \min(I, J),$$

where the ordinary matrix rank is assumed.

C Tensor Ring Embedding

Tensor Ring (TR) decomposition is a generalization to TT-decomposition where the first and the last cores are 3-dimensional tensors which corresponds to $R_0 = R_N > 1$. Formally, a tensor $\boldsymbol{\mathcal{X}}$ is said to be represented in the TR format (Zhao et al., 2016) if each element of $\boldsymbol{\mathcal{X}}$ can be computed as:

$$\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_d) = \sum_{r_0=1}^{R_0} \sum_{r_1=1}^{R_1} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \boldsymbol{\mathcal{G}}^{(1)}(r_0, i_1, r_1) \boldsymbol{\mathcal{G}}^{(2)}(r_1, i_2, r_2) \ldots \boldsymbol{\mathcal{G}}^{(N)}(r_{N-1}, i_N, r_0).$$

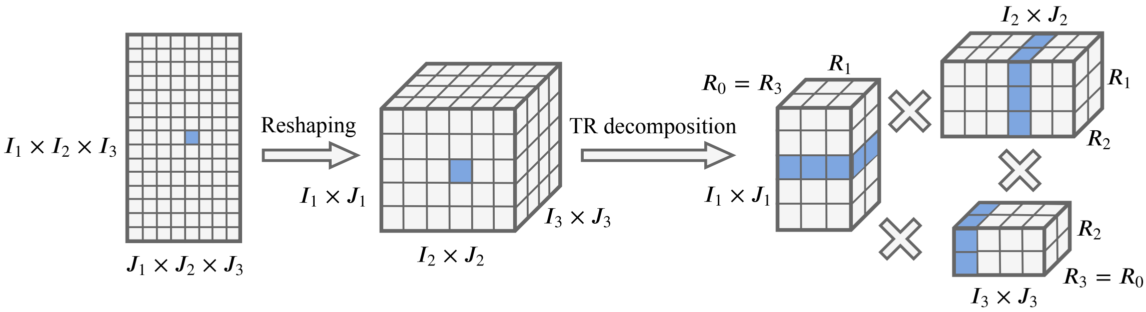

Figure 3: Construction of the TR–matrix from the standard embedding matrix. Blue color depicts how the single element in the initial matrix is transformed into the product of the highlighted matrices. In contrast to TT-embedding, *matrix trace* operator is applied to the final matrix, resulting in a scalar (highlighted element).

Table 5: Sentiment analysis, LSTM with either TT-embedding or TR-embedding on IMDB and SST datasets.

Dataset	**Model**	**Embedding shape**	**Rank**	**Test acc.**	**Emb compr.**	**Total params**
IMDB	Full	25000×256	—	0.886	1	7.19M
	TT1	$(25, 30, 40) \times (4, 8, 8)$	16	0.871	93	0.86M
	TT2	$(10, 10, 15, 20) \times (4, 4, 4, 4)$	16	0.888	232	0.82M
	TT3	$(5, 5, 5, 5, 6, 8) \times (2, 2, 2, 2, 4, 4)$	16	**0.897**	441	0.81M
	TR1	$(25, 30, 40) \times (4, 8, 8)$	16	0.869	45	0.87M
	TR2	$(10, 10, 15, 20) \times (4, 4, 4, 4)$	16	0.872	109	0.86M
	TR3	$(5, 5, 5, 5, 6, 8) \times (2, 2, 2, 2, 4, 4)$	16	0.884	215	0.82M
	TR4	$(25, 30, 40) \times (4, 8, 8)$	8	0.854	152	0.83M
	TR5	$(10, 10, 15, 20) \times (4, 4, 4, 4)$	8	0.882	455	0.80M
	TR6	$(5, 5, 5, 5, 6, 8) \times (2, 2, 2, 2, 4, 4)$	8	0.890	1042	0.80M
SST	Full	17200×256	—	0.374	1	5.19M
	TT1	$(24, 25, 30) \times (4, 8, 8)$	16	0.415	78	0.85M
	TT2	$(10, 10, 12, 15) \times (4, 4, 4, 4)$	16	0.411	182	0.82M
	TT3	$(4, 5, 5, 5, 6, 6) \times (2, 2, 2, 2, 4, 4)$	16	0.399	307	0.81M
	TR1	$(24, 25, 30) \times (4, 8, 8)$	8	**0.427**	128	0.83M
	TR2	$(10, 10, 12, 15) \times (4, 4, 4, 4)$	8	0.411	366	0.80M
	TR3	$(4, 5, 5, 5, 6, 6) \times (2, 2, 2, 2, 4, 4)$	8	0.394	800	0.78M

Similar to TT, we can define TR-matrix (see Figure 3) and corresponding TR-embedding layer.

While our results (Table 5 and Table 6) suggest that TT-embedding shows better compression-performance trade-off than its TR counterpart, much more experimentation is needed to properly compare these two approaches (for example, we see that TR is a promising direction for future work as it outperforms TT on SST-2 benchmark). However, such analysis is computationally heavy and goes beyond the scope of this paper.

D Complete list of hyperparameters

Table 7 and Table 8 contain full lists of hyperparameters we used for training Transformer models for neural machine translation and language modeling respectively.

Table 6: NMT, Transformer-big with either TT- or TR-embedding on WMT'14 English-to-German dataset.

Model	**Embedding shape**	**Rank**	**Token BLEU**	**Sacre BLEU**	**Emb compr.**	**Total params**
Big	32768×1024	—	**29.58**	**28.84**	1	210M
Big+TT1	$(32, 32, 32) \times (8, 8, 16)$	64	29.17	28.53	15.3	179M
Big+TT2	$(32, 32, 32) \times (8, 8, 16)$	48	28.53	27.97	26.8	178M
Big+TT3	$(32, 32, 32) \times (8, 8, 16)$	32	28.26	27.70	58.5	177M
Big+TR1	$(32, 32, 32) \times (8, 8, 16)$	32	28.64	28.07	16	179M
Big+TR2	$(32, 32, 32) \times (8, 8, 16)$	16	28.10	27.50	64	177M

Table 7: Hyperparameters of Transformer-big used for neural machine translation on WMT'14.

Parameter	**Value**
Data cleaning	
max sequence length in tokens	128
max source / target ratio	2.5
Model	
vocabulary size, $\|V\|$	32768
hidden size, d_{model}	1024
intermediate FF layer size, d_{ff}	4096
number of attention heads, h	16
number of layers in enc / dec	6
Optimization	
optimizer	NovoGrad
learning rate	0.04
betas, (β_1, β_2)	$(0.95, 0.25)$
learning rate decay policy	cosine
weight decay	0.0001
batch size in tokens	393216
number of training steps	80000
number of warmup steps	4000
Regularization	
global dropout, P_{drop}	0.2
label smoothing	0.1
Inference	
beam search beam size	4
length penalty	0

Table 8: Hyperparameters of Transformer-XL used for language modeling on WikiText-103.

Parameter	**Value**
Model	
vocabulary size, $\|V\|$	267735
hidden size, d_{model}	512
intermediate FF layer size, d_{ff}	2048
number of attention heads, h	8
number of layers	16
Optimization	
optimizer	NovoGrad
learning rate	0.025
betas, (β_1, β_2)	$(0.95, 0.25)$
learning rate decay policy	cosine
weight decay	0.0001
batch size in sequences	1024
target sequence length	128
memory sequence length	128
number of training steps	300000
number of warmup steps	3000
Regularization	
global dropout, P_{drop}	0.15
Inference	
batch size	4
target sequence length	128
memory sequence length	640
max positional encodings length	400