BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 66, No. 6, 2018
DOI: 10.24425/bpas.2018.125926

INVITED PAPER

Exponential machines

A. NOVIKOV[1,2]*, M. TROFIMOV[4], and I. OSELEDETS[2,3]

[1]National Research University Higher School of Economics
[2]Institute of Numerical Mathematics RAS
[3]Skolkovo Institute of Science and Technology
[4]Federal Research Center "Computer Science and Control" RAS

Abstract. Modeling interactions between features improves the performance of machine learning solutions in many domains (e.g. recommender systems or sentiment analysis). In this paper, we introduce Exponential machines (ExM), a predictor that models all interactions of every order. The key idea is to represent an exponentially large tensor of parameters in a factorized format called tensor train (TT). The tensor train format regularizes the model and lets you control the number of underlying parameters. To train the model, we develop a stochastic Riemannian optimization procedure, which allows us to fit tensors with $\approx 2^{56}$ entries. We show that the model achieves state-of-the-art performance on synthetic data with high-order interactions and that it works on par with high-order factorization machines on a recommender system dataset MovieLens 100 K.

Key words: tensor decomposition, tensor train, factorization machines, Riemannian optimization.

1. Introduction

Machine learning problems with categorical data require modeling interactions between features to solve them. As an example, consider a sentiment analysis problem – detecting whether a review is positive or negative – and the following dataset: 'I liked it', 'I did not like it', 'I'm not sure'. Judging by the presence of the word 'like' or the word 'not' alone, it is hard to understand the tone of the review. But the presence of the pair of words 'not' and 'like' strongly indicates a negative opinion.

If the dictionary has d words, modeling pairwise interactions requires $O(d^2)$ parameters and will probably overfit to the data. Taking into account all interactions (all pairs, triplets, etc. of words) requires impractical 2^d parameters.

In this paper, we show a scalable way to account for all 2^d interactions. Our contributions are:

- We propose a predictor that models all 2^d interactions of d-dimensional data by representing the exponentially large tensor of parameters in a compact multilinear format – Tensor Train (TT-format) (Sec. 3). Factorizing the parameters into the TT-format leads to a better generalization, a linear with respect to d number of underlying parameters and inference time (Sec. 5). The TT-format lets you control the number of underlying parameters through the *TT-rank* – a generalization of the matrix rank to tensors.
- We develop a stochastic Riemannian optimization learning algorithm (Sec. 6.1). In contrast to SGD and its variation (Adam), the Riemannian approach worked reliably across different initialization schemes and in some experiments reached better loss values (Sec. 9.2).
- We show that the linear model (e.g. logistic regression) is a special case of our model with the TT-rank equal 2 (Sec. 6.2).
- We extend the model to handle interactions between functions of the features, not just between the features themselves (Sec. 8).

2. Linear model

In this section, we describe a generalization of a class of machine learning algorithms – the linear model. Let us fix a training dataset of pairs $\left\{\left(x^{(f)}, y^{(f)}\right)\right\}_{f=1}^{N}$ is a d-dimensional feature vector of f-th object, and $y^{(f)}$ is the corresponding target variable. Also fix a loss function $\ell(\hat{y}, y) : \mathbb{R}^2 \to \mathbb{R}$, which takes as input the predicted value $\hat{y}$ and the ground truth value y. We call a model linear, if the prediction of the model depends on the features x only via the dot product between the features x and the d-dimensional vector of parameters w:

$$\hat{y}_{\text{linear}}(x) = \langle x, w \rangle + b, \tag{1}$$

where $b \in \mathbb{R}$ is the bias parameter.

One of the approaches to learn the parameters w and b of the model is to minimize the following loss

$$\sum_{f=1}^{N} \ell\left(\left\langle x^{(f)}, w \right\rangle + b, y^{(f)}\right) + \frac{\lambda}{2}\|w\|_2^2, \tag{2}$$

where λ is the regularization parameter. For the linear model we can choose any regularization term instead of L_2, but later

*e-mail: novikov@bayesgroup.ru

Manuscript submitted 2018-04-26, initially accepted for publication 2018-05-16, published in December 2018.

the choice of the regularization term will become important (see Sec. 6.1).

Several machine learning algorithms can be viewed as a special case of the linear model with an appropriate choice of the loss function $\ell(\hat{y}, y)$: least squares regression (squared loss), Support Vector Machine (hinge loss), and logistic regression (logistic loss).

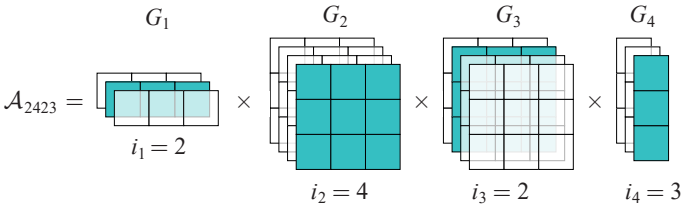

Fig. 1. An illustration of the TT-format for a 3×4×4×3 tensor $\mathcal{A}$ with the TT-rank equal 3

3. Our model

Before introducing our model equation in the general case, consider a 3-dimensional example. The equation includes one term per each subset of features (each interaction)

$$\begin{aligned}\hat{y}(x) &= \mathcal{W}_{000} + \mathcal{W}_{100}x_1 + \mathcal{W}_{010}x_2 + \mathcal{W}_{001}x_3 + \\ &= \mathcal{W}_{110}x_1x_2 + \mathcal{W}_{101}x_1x_3 + \mathcal{W}_{011}x_2x_3 + \\ &= \mathcal{W}_{111}x_1x_2x_3.\end{aligned} \tag{3}$$

Note that all permutations of features in a term (e.g. x_1x_2 and x_2x_1) correspond to a single term and have exactly one associated weight (e.g. $\mathcal{W}_{110}$).

In the general case, we enumerate the subsets of features with a binary vector $(i_1, \ldots, i_d)$, where $i_k = 1$ if the k-th feature belongs to the subset. The model equation looks as follows

$$\hat{y}(x) = \sum_{i_1=0}^{1} \ldots \sum_{i_d=0}^{1} \mathcal{W}_{i_1 \ldots i_d} \prod_{k=1}^{d} x_k^{i_k}. \tag{4}$$

Here we use the notation that $0^0 = 1$. The model is parametrized by a d-dimensional tensor $\mathcal{W}$, which consists of 2^d elements.

The model equation (4) is linear with respect to the weight tensor $\mathcal{W}$. To emphasize this fact and simplify the notation we rewrite the model equation (4) as a tensor dot product $\hat{y}(x) = \langle \mathcal{X}, \mathcal{W} \rangle$, where the tensor X is defined as follows

$$\mathcal{X}_{i_1 \ldots i_d} = \prod_{k=1}^{d} x_k^{i_k}. \tag{5}$$

Note that there is no need in a separate bias term, since it is already included in the model as the weight tensor element $\mathcal{W}_{0 \ldots 0}$ (see the model equation example (3)).

The key idea of our method is to compactly represent the exponentially large tensor of parameters $\mathcal{W}$ in the Tensor Train format [22].

4. Tensor train

A d-dimensional tensor $\mathcal{A}$ is said to be represented in the Tensor Train (TT) format [22], if each of its elements can be computed as the following product of $d-2$ matrices and 2 vectors

$$\mathcal{A}_{i_1 \ldots i_d} = G_1[i_1] \ldots G_d[i_d], \tag{6}$$

where for any $k = 2, \ldots, d-1$ and for any value of i_k, $G_k[i_k]$ is an $r \times r$ matrix, $G_1[i_1]$ is a $1 \times r$ vector and $G_d[i_d]$ is an $1 \times r$ vector (see Fig. 1). We refer to the collection of matrices G_k corresponding to the same dimension k (technically, a 3-dimensional array) as the k-th *TT*-core, where $k = 1, \ldots, d$. The size r of the slices $G_k[i_k]$ controls the trade-off between the representational power of the TT-format and computational efficiency of working with the tensor. We call r the *TT*-rank of the tensor $\mathcal{A}$.

An attractive property of the TT-format is the ability to perform algebraic operations on tensors without materializing them, i.e. by working with the TT-cores instead of the tensors themselves. The TT-format supports computing the norm of a tensor and the dot product between tensors; element-wise sum and element-wise product of two tensors (the result is a tensor in the TT-format with increased TT-rank), and some other operations [22].

5. Inference

In this section, we return to the model proposed in Sec. 3 and show how to compute the model equation (4) in linear time. To avoid the exponential complexity, we represent the weight tensor $\mathcal{W}$ and the data tensor $\mathcal{X}$ (5) in the TT-format. The TT-ranks of these tensors determine the efficiency of the scheme. During the learning, we initialize and optimize the tensor $\mathcal{W}$ in the TT-format and explicitly control its TT-rank. The TT-rank of the tensor $\mathcal{X}$ always equals 1. Indeed, the following TT-cores give the exact representation of the tensor $\mathcal{X}$

$$G_k[i_k] = x_k^{i_k} \in \mathbb{R}^{1 \times 1}, \quad k = 1, \ldots, d$$

The k-th core $G_k[i_k]$ is a 1×1 matrix for any value of $i_k \in \{0, 1\}$, hence the TT-rank of the tensor $\mathcal{X}$ equals 1.

Now that we have TT-representations of tensors $\mathcal{W}$ and $\mathcal{X}$, we can compute the model response $\hat{y}(x) = \langle \mathcal{X}, \mathcal{W} \rangle$ in linear time with respect to the number of features d.

Theorem 1. The computational complexity of computing the model response $\hat{y}(x)$ is $O(r^2 d)$, where r is the TT-rank of the weight tensor $\mathcal{W}$.

Proof. Let us rewrite the definition of the model response (4) assuming that the weight tensor $\mathcal{W}$ is represented in the TT-format (6)

$$\hat{y}(x) = \sum_{i_1,\ldots,i_d} \mathcal{W}_{i_1\ldots i_d} \left(\prod_{k=1}^{d} x_k^{i_k}\right) = \sum_{i_1,\ldots,i_d} G_1[i_1]\ldots G_d[i_d] \left(\prod_{k=1}^{d} x_k^{i_k}\right).$$

Let us group the factors that depend on the variable i_k, $k = 1, \ldots, d$

$$\hat{y}(x) = \sum_{i_1,\ldots,i_d} x_1^{i_1} G_1[i_1]\ldots x_d^{i_d} G_d[i_d] = \left(\sum_{i_1=0}^{1} x_1^{i_1} G_1[i_1]\right)\ldots\left(\sum_{i_d=0}^{1} x_d^{i_d} G_d[i_d]\right) = \underbrace{A_1}_{1\times r}\underbrace{A_2}_{r\times r}\ldots\underbrace{A_d}_{r\times 1},$$

where the matrices A_k for $k = 1, \ldots, d$ are defined as follows

$$A_k = \sum_{i_k=0}^{1} x_k^{i_k} G_k\ [i_k] = G_k\ [0] + x_k G_k\ [1],$$

The final value $\hat{y}\ (x)$ can be computed from the matrices A_k via $d-1$ matrix-by-vector multiplications and 1 vector-by-vector multiplication, which yields $O\left(r^2 d\right)$ complexity.

Note that the proof is constructive and corresponds to the implementation of the inference algorithm. □

The TT-rank of the weight tensor $\mathcal{W}$ is a hyper-parameter of our method and it controls the efficiency vs. flexibility trade-off. A small TT-rank regularizes the model and yields fast learning and inference but restricts the set of possible tensors $\mathcal{W}$. A sufficiently large TT-rank allows any value of the tensor $\mathcal{W}$ and effectively leaves us with the full polynomial model without any advantages of the TT-format.

6. Learning

Learning the parameters of the proposed model corresponds to minimizing the loss under the TT-rank constraint:

$$\begin{aligned} &\underset{\mathcal{W}}{\text{minimize}} && L(\mathcal{W}), \\ &\text{subject to} && \text{TT-rank}(\mathcal{W}) = r_0, \end{aligned} \tag{7}$$

where the loss is defined as follows

$$L\ (\mathcal{W}) = \sum_{f=1}^{N} \ell\left(\left\langle \mathcal{X}^{\ (f)}, \mathcal{W}\right\rangle, y^{\ (f)}\right) + \frac{\lambda}{2}\|\mathcal{W}\|_F^2. \tag{8}$$

Here by the Frobenius norm $\|\cdot\|_F$ we mean the square root of sum of squares of the elements

$$\|\mathcal{W}\|_F^2 = \sum_{i_1=0}^{1} \ldots \sum_{i_d=0}^{1} \mathcal{W}_{i_1\ldots i_d}^2.$$

We consider two approaches to solve problem (7). In a baseline approach, we optimize the objective $L\ (\mathcal{W})$ with the stochastic gradient descent applied to the underlying parameters of the TT-format of the tensor $\mathcal{W}$.

An alternative to the baseline is to perform gradient descent with respect to the tensor $\mathcal{W}$, that is subtract the gradient from the current estimate of $\mathcal{W}$ on each iteration. The TT-format indeed allows to subtract tensors, but this operation increases the TT-rank on each iteration, making this approach impractical.

To improve upon the baseline and avoid the TT-rank growth, we exploit the geometry of the set of tensors that satisfy the TT-rank constraint (7) to build a Riemannian optimization procedure (Sec. 6.1). We experimentally show the advantage of this approach over the baseline in Sec. 9.2.

6.1. Riemannian optimization. The set of all d-dimensional tensors with fixed TT-rank r

$$\mathcal{M}_r = \left\{\mathcal{W} \in \mathbb{R}^{2\times\ldots\times 2} \colon \text{TT-rank}(\mathcal{W}) = r\right\}$$

forms a Riemannian manifold [13]. This observation allows us to use Riemannian optimization to solve problem (7). Riemannian gradient descent consists of the following steps which are repeated until convergence (see Fig. 2 for an illustration):

- Project the gradient $\frac{\partial L}{\partial \mathcal{W}}$ –on the tangent space of $\mathcal{M}_r$ taken at the point $\mathcal{W}$. We denote the tangent space as $T_{\mathcal{W}}\mathcal{M}_r$ and the projection as $\mathcal{G} = P_{T_{\mathcal{W}}\mathcal{M}_r}\left(\frac{\partial L}{\partial \mathcal{W}}\right)$.
- Follow along $\mathcal{G}$ with some step α (this operation increases the TT-rank).
- Retract the new point $\mathcal{W} - \alpha\mathcal{G}$ back to the manifold $\mathcal{M}_r$, that is decrease its TT-rank to r.

We now describe how to implement each of the steps outlined above.

Projecting a TT-tensor $\mathcal{Z}$ on the tangent space of $\mathcal{M}_r$ at a point $\mathcal{W}$ can be done in two steps: preprocessing the tensor $\mathcal{W}$ in $O\left(dr^3\right)$ operations and projecting the tensor $\mathcal{Z}$ in

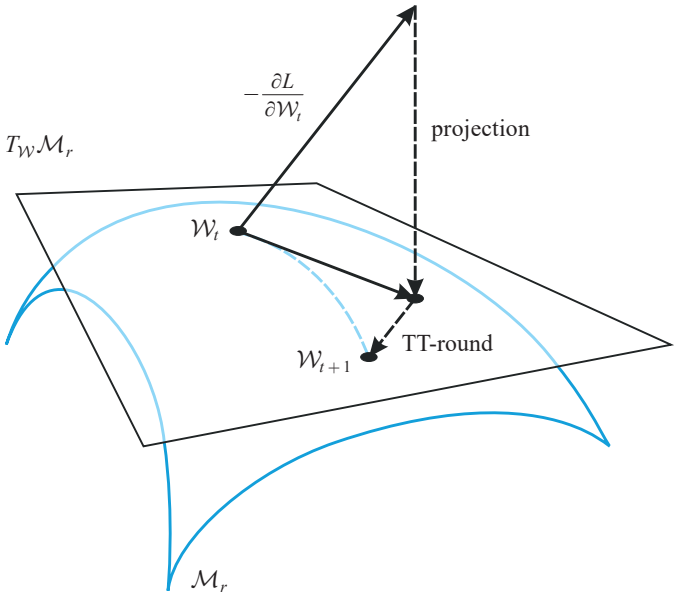

Fig. 2. An illustration of one step of the Riemannian gradient descent. The step-size α is assumed to be 1 for clarity of the figure

$O\left(dr^2\text{TT-rank}\left(\mathcal{Z}\right)^2\right)$ operations [18]. The TT-rank of the projection is bounded by a constant that is independent of the TT-rank of the tensor $\mathcal{Z}$:

$$\text{TT-rank}\left(P_{T_\mathcal{W}\mathcal{M}_r}(\mathcal{Z})\right) \leq 2\text{TT-rank}(\mathcal{W}) = 2r.$$

Let us consider the gradient of the loss function (8)

$$\frac{\partial L}{\partial \mathcal{W}} = \sum_{f=1}^{N} \frac{\partial \ell}{\partial \hat{y}} \mathcal{X}^{(f)} + \lambda \mathcal{W}. \tag{9}$$

Using the fact that $P_{T_\mathcal{W}\mathcal{M}_r}(\mathcal{W}) = \mathcal{W}$ and that the projection is a linear operator we get

$$P_{T_\mathcal{W}\mathcal{M}_r}\left(\frac{\partial L}{\partial \mathcal{W}}\right) = \sum_{f=1}^{N} \frac{\partial \ell}{\partial \hat{y}} P_{T_\mathcal{W}\mathcal{M}_r}\left(\mathcal{X}^{(f)}\right) + \lambda \mathcal{W}. \tag{10}$$

Since the resulting expression is a weighted sum of projections of individual data tensors $\mathcal{X}^{(f)}$, we can project them in parallel. Since the TT-rank of each of them equals 1 (see Sec. 5), the total computational complexity of all N projections is $O\left(dr^2(r+N)\right)$. The TT-rank of the projected gradient is less than or equal to $2r$ regardless of the dataset size N.

Note that here we used the particular choice of the regularization term. For terms other than L_2 (e.g. L_1), the gradient may have arbitrary large TT-rank.

As a retraction – a way to return back to the manifold $\mathcal{M}_r$ – we use the TT-rounding procedure [22]. For a given tensor $\mathcal{W}$ and rank r the TT-rounding procedure returns a tensor $\widehat{\mathcal{W}} = \text{TT-round}(\mathcal{W}, r)$ such that its TT-rank equals r and the Frobenius norm of the residual $\|\mathcal{W} - \widehat{\mathcal{W}}\|_F$ is as small as possible. The computational complexity of the TT-rounding procedure is $O\left(dr^3\right)$.

Since we aim for big datasets, we use a stochastic version of the Riemannian gradient descent: on each iteration we sample a random mini-batch of objects from the dataset, compute the stochastic gradient for this mini-batch, make a step along the projection of the stochastic gradient, and retract back to the manifold (Algorithm 1).

Algorithm 1 Riemannian optimization

Input: Dataset $\{(x^{(f)}, y^{(f)})\}_{f=1}^N$, desired TT-rank r_0, number of iterations T, mini-batch size M, learning rate α, regularization strength λ
Output: $\mathcal{W}$ that approximately minimizes (7)
Train linear model (2) to get the parameters w and b
Initialize the tensor $\mathcal{W}_0$ from w and b with the TT-rank equal r_0
for $t := 1$ **to** T **do**
  Sample M indices $h_1, \ldots, h_M \sim \mathcal{U}(\{1, \ldots, N\})$
  $\mathcal{D}_t := \sum_{j=1}^{M} \frac{\partial \ell}{\partial \hat{y}} \mathcal{X}^{(h_j)} + \lambda \mathcal{W}_{t-1}$
  $\mathcal{G}_t := P_{T_{\mathcal{W}_{t-1}}\mathcal{M}_r}(\mathcal{D}_t)$ (10)
  $\mathcal{W}_t := \text{TT-round}(\mathcal{W}_{t-1} - \alpha\mathcal{G}_t, r_0)$
end for

The computational complexity of an iteration of the stochastic Riemannian gradient descent consists of $O\left(dr^2M\right)$ operations for inference, $O\left(dr^2(r+M)\right)$ operations for gradient projection, and $O\left(dr^3\right)$ operations for retraction, yielding $O\left(dr^2(r+M)\right)$ operations in total.

6.2. Initialization. We found that a random initialization for the TT-tensor $\mathcal{W}$ sometimes freezes the convergence of optimization method (see Sec. 9.2). We propose to initialize the optimization from the solution of the corresponding linear model (1) or from a random linear model.

The following theorem shows how to initialize the weight tensor $\mathcal{W}$ from a linear model.

Theorem 2. For any d-dimensional vector w and a bias term b there exist a tensor $\mathcal{W}$ of TT-rank 2, such that for any d-dimensional vector w and the corresponding object-tensor $\mathcal{X}$ the dot products $\langle x, w\rangle$ and $\langle \mathcal{X}, \mathcal{W}\rangle$ coincide.

To prove the theorem, in the rest of this section we show that the tensor $\mathcal{W}$ from Theorem 2 is representable in the TT-format with the following TT-cores

$$\begin{aligned}
&G_1[0] = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad G_1[1] = \begin{bmatrix} 0 & w_1 \end{bmatrix}, \\
&G_d[0] = \begin{bmatrix} b \\ 1 \end{bmatrix}, \qquad G_d[1] = \begin{bmatrix} w_d \\ 0 \end{bmatrix}, \\
&\forall\, 2 \leq k \leq d-1, \\
&G_k[0] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad G_k[1] = \begin{bmatrix} 0 & w_k \\ 0 & 0 \end{bmatrix},
\end{aligned} \tag{11}$$

and thus the TT-rank of the tensor $\mathcal{W}$ equals 2.

We start the proof with the following lemma:

Lemma 1. For the TT-cores (11) and any $p = 1, \ldots, d-1$ the following invariance holds:

$$G_1[i_1] \ldots G_p[i_p] = \begin{cases} \begin{bmatrix} 1 & 0 \end{bmatrix}, & \text{if } \sum_{q=1}^{p} i_q = 0, \\ \begin{bmatrix} 0 & 0 \end{bmatrix}, & \text{if } \sum_{q=1}^{p} i_q \geq 2, \\ \begin{bmatrix} 0 & w_k \end{bmatrix}, & \text{if } \sum_{q=1}^{p} i_q = 1, \\ & \text{and } i_k = 1. \end{cases}$$

Proof. We prove the lemma by induction. Indeed, for $p = 1$ the statement of the lemma becomes

$$G_1[i_1] = \begin{cases} \begin{bmatrix} 1 & 0 \end{bmatrix}, & \text{if } i_1 = 0, \\ \begin{bmatrix} 0 & w_1 \end{bmatrix}, & \text{if } i_1 = 1, \end{cases}$$

which holds by definition of the first TT-core $G_1[i_1]$.

Now suppose that the statement of Lemma 1 is true for some $p - 1 \geq 1$. If $i_p = 0$, then $G_p[i_p]$ is an identity matrix and $G_1[i_1] \ldots G_p[i_p] = G_1[i_1] \ldots G_{p-1}[i_{p-1}]$. Also, $\sum_{q=1}^{p} i_q = \sum_{q=1}^{p-1} i_q$, so the statement of the lemma stays the same.

If $i_p = 1$, then there are 3 options:

- If $\sum_{q=1}^{p-1} i_q = 0$, then $\sum_{q=1}^{p} i_q = 1$ and
  $G_1[i_1] \dots G_p[i_p] = \begin{bmatrix} 1 & 0 \end{bmatrix} G_p[1] = \begin{bmatrix} 0 & w_p \end{bmatrix}$.
- If $\sum_{q=1}^{p-1} i_q \geq 2$, then $\sum_{q=1}^{p} i_q \geq 2$ and
  $G_1[i_1] \dots G_p[i_p] = \begin{bmatrix} 0 & 0 \end{bmatrix} G_p[1] = \begin{bmatrix} 0 & 0 \end{bmatrix}$.
- If $\sum_{q=1}^{p-1} i_q = 1$ with $i_k = 1$, then $\sum_{q=1}^{p} i_q \geq 2$ and
  $G_1[i_1] \dots G_p[i_p] = \begin{bmatrix} 0 & w_k \end{bmatrix} G_p[1] = \begin{bmatrix} 0 & 0 \end{bmatrix}$.

Which is exactly the statement of Lemma 1. □

Proof of Theorem 2. The product of all TT-cores can be represented as a product of the first $p = d - 1$ cores times the last core $G_d[i_d]$ and by using Lemma 1 we get

$$\mathcal{W}_{i_1 \dots i_d} = G_1[i_1] \dots G_{d-1}[i_{d-1}] G_d[i_d] = \\ = \begin{cases} b, & \text{if } \sum_{q=1}^{d} i_q = 0, \\ 0, & \text{if } \sum_{q=1}^{d} i_q \geq 2, \\ w_k, & \text{if } \sum_{q=1}^{d} i_q = 1, \\ & \quad \text{and } i_k = 1. \end{cases}$$

The elements of the obtained tensor $\mathcal{W}$ that correspond to interactions of order ≥ 2 equal to zero; the weight that corresponds to x_k equals to w_k; and the bias term $\mathcal{W}_{0 \dots 0} = b$.

The TT-rank of the obtained tensor equal 2 since its TT-cores are of size 2×2. □

7. Order regularization

In this section, we propose a regularization scheme that encourages the model to shrink the coefficients of the high-order terms. In the loss function (8), coefficients of terms of different order are equally regularized by the L_2 penalty. Instead, we propose to make the strength of the regularization to be some predefined number $\beta > 1$ (a hyperparameter) raised to the power of the order of the term, e.g. in the case of 3 features

$$L(\mathcal{W}) = \sum_{f=1}^{N} \ell\left(\left\langle \mathcal{X}^{(f)}, \mathcal{W} \right\rangle, y^{(f)}\right) + \frac{\lambda}{2} \left(\mathcal{W}_{000}^2 + \right. \\ + \beta \mathcal{W}_{001}^2 + \beta \mathcal{W}_{010}^2 + \beta \mathcal{W}_{100}^2 + \beta^2 \mathcal{W}_{011}^2 + \\ \left. + \beta^2 \mathcal{W}_{101}^2 + \beta^2 \mathcal{W}_{110}^2 + \beta^3 \mathcal{W}_{111}^2 \right) \quad (12)$$

In the general case, the loss under this regularization looks as follows

$$L(\mathcal{W}) = \sum_{f=1}^{N} \ell\left(\left\langle \mathcal{X}^{(f)}, \mathcal{W} \right\rangle, y^{(f)}\right) + \\ + \frac{\lambda}{2} \left(\sum_{i_1, \dots, i_d} \beta^{i_1 + \dots + i_d} \mathcal{W}_{i_1 \dots i_d}^2 \right). \quad (13)$$

Order regularization can be incorporated into the learning procedure by noting that the new regularization term is the Frobenius of the weighted parameter tensor $\left\| \mathcal{B} \odot \mathcal{W} \right\|_F^2$, where the tensor of weights $\mathcal{B}$ has rank-1 structure

$$\mathcal{B}_{i_1 \dots i_d} = \prod_{k=1}^{d} \beta^{i_k}.$$

When $\beta = 1$, order regularization coincides with the original L_2 penalty, but increasing $\beta > 1$ leads to better stability and generalization in our experiments (see Sec. 9.3).

8. Extending the model

In this section, we extend the proposed model to handle polynomials of any functions of the features. As an example, consider the logarithms of the features in the 2-dimensional case:

$$\hat{y}^{\log}(x) = \mathcal{W}_{00} + \mathcal{W}_{01} x_1 + \mathcal{W}_{10} x_2 + \mathcal{W}_{11} x_1 x_2 + \\ = \mathcal{W}_{20} \log(x_1) + \mathcal{W}_{02} \log(x_2) + \\ = \mathcal{W}_{12} x_1 \log(x_2) + \mathcal{W}_{21} x_2 \log(x_1) + \\ = \mathcal{W}_{22} \log(x_1) \log(x_2).$$

In the general case, to model interactions between n_g functions $g_1, \dots, g_{n_g}$ of the features we redefine the object-tensor as follows:

$$\mathcal{X}_{i_1 \dots i_d} = \prod_{k=1}^{d} c(x_k, i_k),$$

where

$$c(x_k, i_k) = \begin{cases} 1, & \text{if } i_k = 0, \\ g_1(x_k), & \text{if } i_k = 1, \\ \dots & \\ g_{n_g}(x_k), & \text{if } i_k = n_g, \end{cases}$$

The weight tensor $\mathcal{W}$ and the object-tensor $\mathcal{X}$ are now consist of $(n_g + 1)^d$ elements. After this change to the object-tensor $\mathcal{X}$, learning and inference algorithms will stay unchanged compared to the original model (4).

Categorical features. Our basic model handles categorical features $x_k \in \{1, \dots, K\}$ by converting them into one-hot vectors $x_{k,1}, \dots, x_{k,K}$. The downside of this approach is that it wastes the model capacity on modeling non-existing interactions between the one-hot vector elements $x_{k,1}, \dots, x_{k,K}$ which correspond to the same categorical feature. Instead, we propose to use one TT-core per categorical feature and use the model extension technique with the following function

$$c(x_k, i_k) = \begin{cases} 1, & \text{if } x_k = i_k \text{ or } i_k = 0, \\ 0, & \text{otherwise.} \end{cases}$$

This allows us to cut the number of parameters per categorical feature from $2Kr^2$ to $(K+1)r^2$ without losing any representational power.

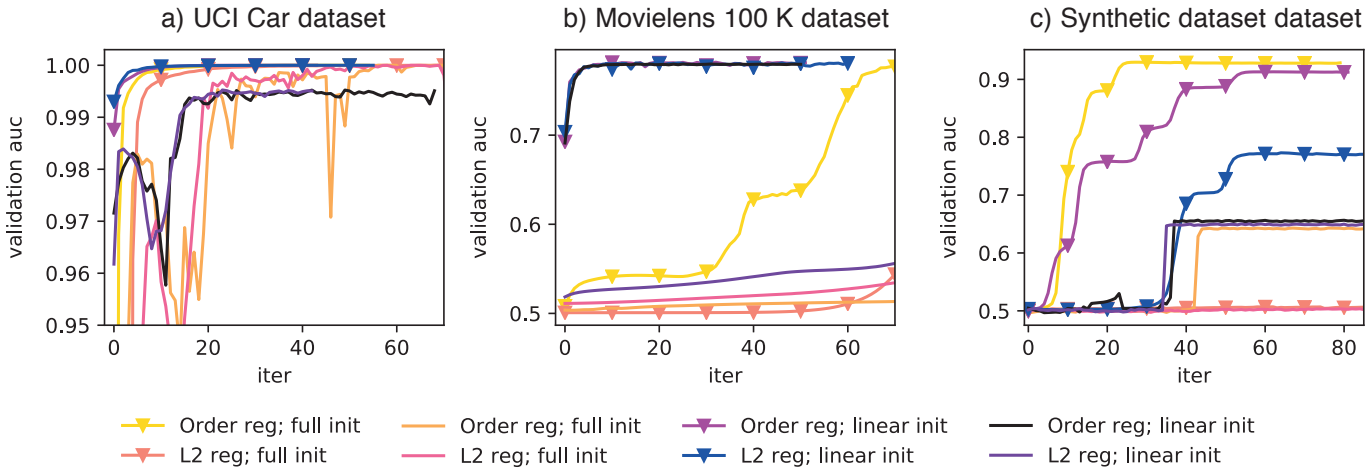

Fig. 3. A comparison between Riemannian optimization (triangle markers) and Adam applied to the underlying parameters of the TT-format (no markers). 'Order reg' corresponds to using the order regularization (see Sec. 7) and choosing the parameter β by grid-search w.r.t. validation performance; 'L2 reg' corresponds to using L_2 regularization. 'Full init' corresponds to initializing the model with a TT-tensor with random TT-cores; 'linear init' corresponds to initializing the TT-tensor such that the model is linear with random coefficients. See details in Sec. 9.2 and 9.3

9. Experiments

We release a Python implementation of the proposed algorithm[1]. For the operations related to the TT-format, we use the T3F library [20] which is built on top of TensorFlow library [11].

9.1. Datasets. The datasets used in the experiments are:

1. **UCI [10] Car dataset** is a classification problem with 1728 objects and 21 binary features (after one-hot encoding). We randomly split the data into 864 training and 864 test objects. For simplicity, we binarized the labels: we picked the first class ('unacc') and made a one-versus-rest binary classification problem from the original Car dataset.
2. **Synthetic data.** We generated 100 000 train and 100 000 test objects with 30 features. Each entry of the data matrix X was independently sampled from $\{-1, +1\}$ with equal probabilities 0.5. We also uniformly sampled 20 subsets of features (interactions) of order 6: $j_1^1, \ldots, j_6^1, \ldots, j_1^{20}, \ldots, j_6^{20} \sim \sim \mathcal{U}\{1, \ldots, 30\}$. We set the ground truth target variable to a deterministic function of the input: $y(x) = \sum_{z=1}^{20} \varepsilon_z \prod_{h=1}^{6} x_{j_h^z}$, and sampled the weights of the interactions from the uniform distribution: $\varepsilon_1, \ldots, \varepsilon_{20} \sim \mathcal{U}(-1, 1)$.
3. **MovieLens 100 K.** MovieLens 100 K is a recommender system dataset with 943 users and 1682 movies [11]. We followed [3] in preparing the features and in turning the problem into binary classification. For users, we treated age (rounded to decades), living area (the first digit of the zip-code), gender, and occupation as categorical features. For movies, we used the release year (rounded to decades) and genres. Original ratings were binarized using 5 as a threshold. This results in 21 200 positive samples, half of which were used for training (with the equal amount of sampled negative examples) and the rest were used for testing.

[1] https://github.com/Bihaqo/exp-machines

9.2. Comparing optimizers and initialization schemes. In this experiment, we compare two approaches to training the model: Riemannian optimization vs. the Adam optimization method [15], which is a variation of the stochastic gradient descent (SGD) with adaptive learning rates (for details on optimization methods, see Sec. 6). In all our experiments, plain SGD is highly inferior to both Riemannian approach and Adam, so we exclude SGD from the figures.

We also compare two ways of randomly initializing the weight tensor $\mathcal{W}$: 1) filling its TT-cores with independent Gaussian noise; 2) initializing $\mathcal{W}$ to represent a linear model with random coefficients sampled from a standard Gaussian (see Sec. 6.2).

In this and later experiments, we tune the TT-rank and regularization strength of the model, as well as the learning rate for both Riemannian and SGD optimizers with respect to the validation loss on the last iteration by the grid search. We exclude optimization runs that diverged or encountered NaN values.

We report that while for some experiments and parameter settings the Adam optimization method worked on par with the Riemannian approach, Riemannian optimization is more robust to the initialization choice and is superior on the synthetic dataset (Fig. 3).

We also report that the proposed initialization from a random linear model outperforms initialization from a fully random TT-tensor on both UCI Car and the Movielens dataset and on the synthetic dataset when using L_2 regularization. A possible explanation, is that by initializing the model in such a way that high-order terms dominate we may force the optimizer to focus on high-order terms at the beginning; with linear initialization, optimization starts with figuring right values for low-order (linear) terms. It seems better to start with fitting a simpler (linear) model and only then to correct the predictions with higher-order terms. This also explains why using full initialization outperforms the linear initialization on the synthetic dataset with order regularization: on this dataset linear models cannot achieve AUC higher than 0.5 and it is beneficial

to consider higher-order interactions from the beginning of the optimization.

9.3. Order regularization. In this experiment, we assess the influence of order regularization on the learning process. We report that order regularization improves the validation AUC, especially when used with full initialization (a random TT-tensor), which includes high-order terms (Fig. 3).

9.4. Comparison to other approaches. On the synthetic dataset with high-order interactions we compared Exponential Machines (the proposed method) with scikit-learn implementation [23] of logistic regression, random forest, and kernel SVM; FastFM implementation [2] of 2-nd order Factorization Machines; our implementation of high-order Factorization Machines[2]; and a feed-forward neural network implemented in TensorFlow [1]. We used 6-th order FM with the Adam optimizer [15] for which we had chosen the best rank (20) and learning rate (0.003) based on the validation loss after the first 50 iterations. We tried several feed-forward neural networks with ReLU activations and up to 4 fully-connected layers and 128 hidden units. We compared the models based on the Area Under the Curve (AUC) metric since it is applicable to all methods and is robust to unbalanced labels. We report that our model works on par with high-order Factorization Machines and outperforms all the baselines in the case of limited training time budget (yielding 0.86 test AUC in 312 seconds, see Table 1).

Table 1
A comparison between models on synthetic data with high-order interactions (Sec. 9.4). We report the inference time on 100 000 test objects in the last column

Method	Test AUC	Training time (s)	Inference time (s)
Log. reg.	0.50	0.4	0.0
RF	0.55	21.4	6.5
Neural Network	0.50	47.2	0.1
SVM RBF	0.50	2262.6	5380
SVM poly. 2	0.50	1152.6	4260
SVM poly. 6	0.56	4090.9	3774
2-nd order FM	0.50	638.2	0.5
6-th order FM	0.57	549	3
6-th order FM	0.86	6039	3
6-th order FM	**0.96**	38918	3
ExM rank 8	0.76	48	0.2
ExM rank 12	0.86	312	0.2
ExM rank 16	0.92	720	0.2
ExM rank 20	**0.96**	4056	0.3

On the MovieLens 100 K dataset we used the categorical features representation described in Sec. 8. Our model obtained 0.783 test AUC with the TT-rank equal 3 in 131 seconds; logistic regression obtained 0.782; our implementation of 3-rd order FM obtained 0.782; and the implementation from [3] obtained 0.786 with 3-rd order FM on the same data.

[2] https://github.com/geffy/tffm

9.5. TT-rank. The TT-rank is one of the main hyperparameters of the proposed model. Two possible strategies can be used to choose it: grid-search or DMRG-like algorithms (see Related Work, Sec. 10). In our experiments, we opted for the former and observed that the model is fairly robust to the choice of the TT-rank (see Fig. 4), but a too small TT-rank can hurt the accuracy (see Table 1).

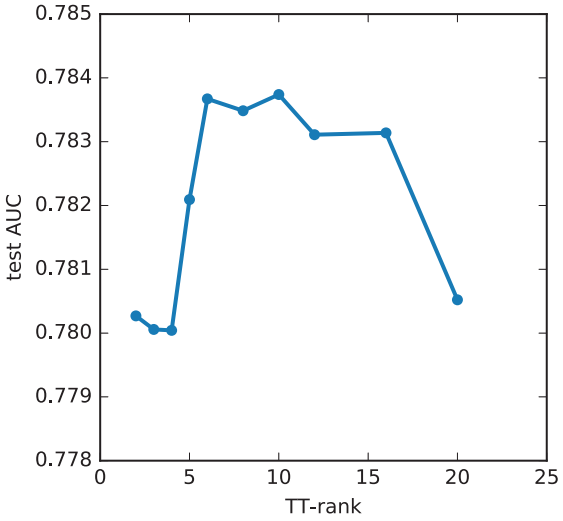

Fig. 4. The influence of the TT-rank on the test AUC for the MovieLens 100 K dataset

10. Related work

Kernel SVM is a flexible non-linear predictor and, in particular, it can model interactions between features when used with the polynomial kernel [6]. As a downside, it scales at least quadratically with the dataset size [6] and overfits on highly sparse data.

Factorization Machine (FM) [24] is a general predictor that models pairwise interactions. FM overcomes the problems of polynomial SVM by restricting the rank of the weight matrix, which leads to a linear number of parameters and generalizes better on sparse data. FM running time is linear with respect to the number of nonzero elements in the data, which allows scaling to billions of training entries on sparse problems.

For high-order interactions FM uses CP-format [7, 12] to represent the tensor of parameters. The choice of the tensor factorization is the main difference between the high-order FM and Exponential Machines. The TT-format comes with two advantages over the CP-format: first, the TT-format allows for Riemannian optimization; second, the problem of finding the best TT-rank r approximation to a given tensor always has a solution and can be solved in polynomial time. We found Riemannian optimization superior to the SGD baseline (Sec. 6) that was used in several other models parametrized by a tensor factorization [24, 16, 21]. Note that CP-format also allows for Riemannian optimization, but only for 2-order tensors (and thereafter 2-order FM).

A number of works used full-batch or stochastic Riemannian optimization for data processing tasks [19, 27, 29, 31]. The last work [31] is especially interesting in the context of our method, since it improves the convergence rate of stochastic Riemannian gradient descent and is directly applicable to our learning procedure. For an overview of applications of tensor methods and Riemannian optimization for large scale data analysis, see [8, 9].

In a concurrent work, a model similar to Exponential Machines was proposed which, however, relies on the trigonometric basis $\left(\cos(\pi/2 x), \sin(\pi/2 x)\right)$ in contrast to polynomials $(1, x)$ used in Exponential Machines (see Sec. 8 for an explanation on how to change the basis) [26]. They also proposed a different learning procedure inspired by the DMRG algorithm [25], which allows to automatically choose the ranks of the model, but is hard to adapt to the stochastic regime. One of the possible ways to combine strengths of the DMRG and Riemannian approaches is to do a full DMRG sweep once in a few epochs of the stochastic Riemannian gradient descent to adjust the ranks.

Another concurrent work focused on the theoretical properties of the model presented in this paper: they showed the connection between this model and recurrent neural networks and used it to prove that RNNs are exponentially more expressive than shallow networks [14].

Other relevant works include the model that approximates the decision function with a multidimensional Fourier series whose coefficients lie in the TT-format [28]; and models that are similar to FM but include squares and other powers of the features: Tensor Machines [30] and Polynomial Networks [17]. Tensor Machines also enjoy a theoretical generalization bound.

Another relevant work boosted the efficiency of FM and Polynomial Networks by casting their training as a low-rank tensor estimation problem, thus making it multi-convex and allowing for efficient use of Alternative Least Squares types of algorithms [4]. Note that Exponential Machines are inherently multi-convex.

11. Discussion

We presented a predictor that models all interactions of every order. To regularize the model and to make the learning and inference feasible, we represented the exponentially large tensor of parameters in the Tensor Train format. To train the model, we used Riemannian optimization in the stochastic regime and report that it outperforms a popular baseline based on the stochastic gradient descent. We found that training process is sensitive to initialization and proposed an initialization strategy based on the solution of the corresponding linear problem and an initialization that corresponds to a random linear model. The solutions developed in this paper for the stochastic Riemannian optimization may suit other machine learning models parametrized by tensors in the TT-format.

Acknowledgements. Sections 7, 8, and 9.3 were supported by Russian Science Foundation (project №17-71-20072). The rest of the study was supported by the Ministry of Education and Science of the Russian Federation (grant №14.756.31.0001).

References

[1] M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous systems", 2015. Software available from tensorflow.org.

[2] I. Bayer, "FASTFM: A library for factorization machines", *Journal of Machine Learning Research*, 2016.

[3] M. Blondel, A. Fujino, N. Ueda, and M. Ishihata, "Higher-order factorization machines", *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.

[4] M. Blondel, M. Ishihata, A. Fujino, and N. Ueda, "Polynomial networks and factorization machines: New insights and efficient training algorithms", In *Advances in Neural Information Processing Systems 29 (NIPS)*. 2016.

[5] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning", *The Journal of Machine Learning Research*, 6, 1579–1619, 2005.

[6] B.E. Boser, I.M. Guyon, and V.N. Vapnik. "A training algorithm for optimal margin classifiers", In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[7] J.D. Caroll and J.J. Chang, "Analysis of individual differences in multidimensional scaling via n-way generalization of eckartyoung decomposition", *Psychometrika*, 35, 283–319, 1970.

[8] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions", *Foundations and Trends® in Machine Learning*, 9(4–5), 249–429, 2016.

[9] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives", *Foundations and Trends® in Machine Learning*, 9(6), 431–673, 2017.

[10] D. Dheeru and E.K. Taniskidou, "UCI machine learning repository", 2017.

[11] F.M. Harper and A.J. Konstan, "The movielens datasets: History and context", *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2015.

[12] R.A. Harshman, "Foundations of the parafac procedure: models and conditions for an explanatory multimodal factor analysis", *UCLA Working Papers in Phonetics*, 16, 1–84, 1970.

[13] S. Holtz, T. Rohwedder, and R. Schneider, "On manifolds of tensors of fixed tt-rank", *Numerische Mathematik*, pages 701–731, 2012.

[14] V. Khrulkov, A. Novikov, and I. Oseledets, "Expressive power of recurrent neural networks", In *International Conference on Learning Representations (ICLR)*, 2018.

[15] D. Kingma and J. Ba. Adam, "A method for stochastic optimization", In I*nternational Conference on Learning Representations (ICLR)*, 2015.

[16] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations (ICLR)*, 2014.

[17] R. Livni, S. Shalev-Shwartz, and O. Shamir. "On the computational efficiency of training neural networks", In *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014.

[18] C. Lubich, I. V. Oseledets, and B. Vandereycken, "Time integration of tensor trains", *SIAM Journal on Numerical Analysis*, pages 917–941, 2015.

[19] G. Meyer, S. Bonnabel, and R. Sepulchre, "Regression on fixedrank positive semidefinite matrices: a Riemannian approach", *The Journal of Machine Learning Research*, 593–625, 2011.

[20] A. Novikov, P. Izmailov, V. Khrulkov, M. Figurnov, and I. Oseledets, "Tensor train decomposition on tensorflow (t3f)", *arXiv preprint arXiv:1801.01928*, 2018.

[21] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks", In *Advances in Neural Information Processing Systems 28 (NIPS)*. 2015.

[22] I. V. Oseledets, "Tensor-train decomposition", *SIAM J. Scientific Computing*, 33(5), 2295–2317, 2011.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python", *Journal of Machine Learning Research*, 12, 2825–2830, 2011.

[24] S. Rendle. "Factorization machines", In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000, 2010.

[25] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states", *Annals of Physics*, 326(1), 96–192, 2011.

[26] E. Stoudenmire and D. J. Schwab, "Supervised learning with tensor networks", In *Advances in Neural Information Processing Systems 29 (NIPS)*. 2016.

[27] M. Tan, I.W. Tsang, L. Wang, B. Vandereycken, and S.J. Pan, "Riemannian pursuit for big matrix recovery", In *Proceedings of The 31st International Conference on Machine Learning (ICML)*, 2014.

[28] S. Wahls, V. Koivunen, H.V. Poor, and M. Verhaegen. "Learning multidimensional fourier series with tensor trains", In *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*, pages 394–398. IEEE, 2014.

[29] Z. Xu and Y. Ke. "Stochastic variance reduced Riemannian eigensolver", *arXiv preprint arXiv:1605.08233*, 2016.

[30] J. Yang and A. Gittens, "Tensor machines for learning targetspecific polynomial features", *arXiv preprint arXiv:1504.01697*, 2015.

[31] H. Zhang, S.J. Reddi, and S. Sra. Riemannian, "SVRG: Fast stochastic optimization on riemannian manifolds", *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.