EXPONENTIAL MACHINES

Alexander Novikov[1,2]
novikov@bayesgroup.ru

Mikhail Trofimov[3]
mikhail.trofimov@phystech.edu

Ivan Oseledets[2,4]
i.oseledets@skoltech.ru

[1]National Research University Higher School of Economics, Moscow, Russia
[2]Institute of Numerical Mathematics, Moscow, Russia
[3]Moscow Institute of Physics and Technology, Moscow, Russia
[4]Skolkovo Institute of Science and Technology, Moscow, Russia

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 2^{160} 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 100K.

1 INTRODUCTION

Machine learning problems with categorical data require modeling interactions between the 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 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 our experiments, it outperformed the stochastic gradient descent baseline (Sec. 8.2) that is often used for models parametrized by a tensor decomposition (see related works, Sec. 9).

- We show that the linear model (e.g. logistic regression) is a special case of our model with the TT-rank equal 2 (Sec. 8.3).

- We extend the model to handle interactions between functions of the features, not just between the features themselves (Sec. 7).

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 $\{(\boldsymbol{x}^{(f)}, y^{(f)})\}_{f=1}^{N}$, where $\boldsymbol{x}^{(f)}$ is a d-dimensional feature vector of f-th object, and $y^{(f)}$ is the corresponding target variable. Also fix a loss function $\ell(\widehat{y}, y) : \mathbb{R}^2 \to \mathbb{R}$, which takes as input the predicted value $\widehat{y}$ and the ground truth value y. We call a model *linear*, if the prediction of the model depends on the features $\boldsymbol{x}$ only via the dot product between the features $\boldsymbol{x}$ and the d-dimensional vector of parameters $\boldsymbol{w}$:

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

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

One of the approaches to learn the parameters $\boldsymbol{w}$ and b of the model is to minimize the following loss

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

where λ is the regularization parameter. For the linear model we can choose any regularization term instead of L_2, but later 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 $\ell(\widehat{y}, y)$: least squares regression (squared loss), Support Vector Machine (hinge loss), and logistic regression (logistic loss).

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)

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

Note that all permutations of features in a term (e.g. $x_1 x_2$ and $x_2 x_1$) 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

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

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

The model equation (4) is linear with respect to the weight tensor $\boldsymbol{\mathcal{W}}$. To emphasize this fact and simplify the notation we rewrite the model equation (4) as a tensor dot product $\widehat{y}(\boldsymbol{x}) = \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}} \rangle$, where the tensor $\boldsymbol{\mathcal{X}}$ 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 $\boldsymbol{\mathcal{W}}$ in the Tensor Train format (Oseledets, 2011).

4 TENSOR TRAIN

A d-dimensional tensor $\boldsymbol{\mathcal{A}}$ is said to be represented in the Tensor Train (TT) format (Oseledets, 2011), if each of its elements can be computed as the following product of $d - 2$ matrices and 2 vectors

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

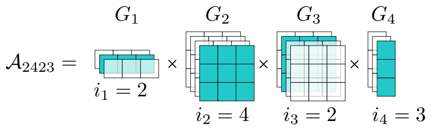


Figure 1: An illustration of the TT-format for a $3 \times 4 \times 4 \times 3$ tensor $\boldsymbol{\mathcal{A}}$ with the TT-rank equal 3.

where for any $k = 2, \dots, d-1$ 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 $r \times 1$ 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 $k = 1, \dots, d$. 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 $\boldsymbol{\mathcal{A}}$.

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 (Oseledets, 2011).

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 $\boldsymbol{\mathcal{W}}$ and the data tensor $\boldsymbol{\mathcal{X}}$ (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 $\boldsymbol{\mathcal{W}}$ in the TT-format and explicitly control its TT-rank. The TT-rank of the tensor $\boldsymbol{\mathcal{X}}$ always equals 1. Indeed, the following TT-cores give the exact representation of the tensor $\boldsymbol{\mathcal{X}}$

$$G_k[i_k] = x_k^{i_k} \in \mathbb{R}^{1 \times 1}, \;\; k = 1, \dots, d.$$

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 $\boldsymbol{\mathcal{X}}$ equals 1.

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

Theorem 1. *The model response $\widehat{y}(\boldsymbol{x})$ can be computed in $O(r^2 d)$, where r is the TT-rank of the weight tensor $\boldsymbol{\mathcal{W}}$.*

We refer the reader to Appendix A where we propose an inference algorithm with $O(r^2 d)$ complexity and thus prove Theorem 1.

The TT-rank of the weight tensor $\boldsymbol{\mathcal{W}}$ 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 possible values of the tensor $\boldsymbol{\mathcal{W}}$. A large TT-rank allows any value of the tensor $\boldsymbol{\mathcal{W}}$ 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:

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

where the loss is defined as follows

$$L(\boldsymbol{\mathcal{W}}) = \sum_{f=1}^{N} \ell\left(\langle \boldsymbol{\mathcal{X}}^{(f)}, \boldsymbol{\mathcal{W}} \rangle, \, y^{(f)}\right) + \frac{\lambda}{2} \|\boldsymbol{\mathcal{W}}\|_F^2, \quad \|\boldsymbol{\mathcal{W}}\|_F^2 = \sum_{i_1=0}^{1} \dots \sum_{i_d=0}^{1} \mathcal{W}_{i_1 \dots i_d}^2. \tag{8}$$

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

A simple alternative to the baseline is to perform gradient descent with respect to the tensor $\boldsymbol{\mathcal{W}}$, that is subtract the gradient from the current estimate of $\boldsymbol{\mathcal{W}}$ 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. 8.2.

6.1 RIEMANNIAN OPTIMIZATION

The set of all d-dimensional tensors with fixed TT-rank r

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

forms a Riemannian manifold (Holtz et al., 2012). 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):

1. Project the gradient $\frac{\partial L}{\partial \boldsymbol{\mathcal{W}}}$ on the tangent space of $\mathcal{M}_r$ taken at the point $\boldsymbol{\mathcal{W}}$. We denote the tangent space as $T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r$ and the projection as $\boldsymbol{\mathcal{G}} = P_{T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r}(\frac{\partial L}{\partial \boldsymbol{\mathcal{W}}})$.
2. Follow along $\boldsymbol{\mathcal{G}}$ with some step α (this operation increases the TT-rank).
3. Retract the new point $\boldsymbol{\mathcal{W}} - \alpha\boldsymbol{\mathcal{G}}$ 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.

Lubich et al. (2015) proposed an algorithm to project a TT-tensor $\boldsymbol{\mathcal{Z}}$ on the tangent space of $\mathcal{M}_r$ at a point $\boldsymbol{\mathcal{W}}$ which consists of two steps: preprocess $\boldsymbol{\mathcal{W}}$ in $O(dr^3)$ and project $\boldsymbol{\mathcal{Z}}$ in $O(dr^2 \, \text{TT-rank}(\boldsymbol{\mathcal{Z}})^2)$. Lubich et al. (2015) also showed that the TT-rank of the projection is bounded by a constant that is independent of the TT-rank of the tensor $\boldsymbol{\mathcal{Z}}$:

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

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

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

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

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

Since the resulting expression is a weighted sum of projections of individual data tensors $\boldsymbol{\mathcal{X}}^{(f)}$, we can project them in parallel. Since the TT-rank of each of them equals 1 (see Sec. 5), all N projections cost $O(dr^2(r+N))$ in total. The TT-rank of the projected gradient is less 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 (Oseledets, 2011). For a given tensor $\boldsymbol{\mathcal{W}}$ and rank r the TT-rounding procedure returns a tensor $\widehat{\boldsymbol{\mathcal{W}}} = \text{TT-round}(\boldsymbol{\mathcal{W}},\, r)$ such that its TT-rank equals r and the Frobenius norm of the residual $\|\boldsymbol{\mathcal{W}} - \widehat{\boldsymbol{\mathcal{W}}}\|_F$ is as small as possible. The computational complexity of the TT-rounding procedure is $O(dr^3)$.

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 (Alg. 1).

An iteration of the stochastic Riemannian gradient descent consists of inference $O(dr^2M)$, projection $O(dr^2(r+M))$, and retraction $O(dr^3)$, which yields $O(dr^2(r+M))$ total computational complexity.

Algorithm 1 Riemannian optimization

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

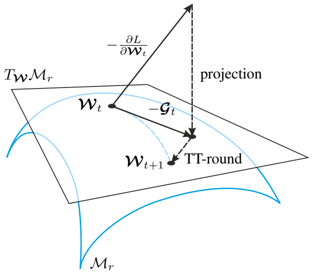

$-\frac{\partial L}{\partial \boldsymbol{\mathcal{W}}_t}$
projection
$T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r$
$\boldsymbol{\mathcal{W}}_t$
$-\boldsymbol{\mathcal{G}}_t$
$\boldsymbol{\mathcal{W}}_{t+1}$
TT-round
$\mathcal{M}_r$

Figure 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.

6.2 Initialization

We found that a random initialization for the TT-tensor $\boldsymbol{\mathcal{W}}$ sometimes freezes the convergence of optimization method (Sec. 8.3). We propose to initialize the optimization from the solution of the corresponding linear model (1).

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

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

The proof is provided in Appendix B.

7 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:

$$\begin{aligned}\widehat{y}^{\log}(\boldsymbol{x}) = & \mathcal{W}_{00} + \mathcal{W}_{01}x_1 + \mathcal{W}_{10}x_2 + \mathcal{W}_{11}x_1x_2 + \mathcal{W}_{20}\ \log(x_1) + \mathcal{W}_{02}\ \log(x_2) \\ & + \mathcal{W}_{12}\ x_1 \log(x_2) + \mathcal{W}_{21}\ x_2 \log(x_1) + \mathcal{W}_{22}\ \log(x_1) \log(x_2).\end{aligned}$$

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 $\boldsymbol{\mathcal{W}}$ and the object-tensor $\boldsymbol{\mathcal{X}}$ are now consist of $(n_g + 1)^d$ elements. After this change to the object-tensor $\boldsymbol{\mathcal{X}}$, 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.

8 Experiments

We release a Python implementation of the proposed algorithm and the code to reproduce the experiments[1]. For the operations related to the TT-format, we used the TT-Toolbox[2].

8.1 Datasets

The datasets used in the experiments (see details in Appendix C)

1. **UCI (Lichman, 2013) Car dataset** is a classification problem with 1728 objects and 21 binary features (after one-hot encoding). We randomly splitted the data into 1382 training and 346 test objects and binarized the labels for simplicity.
2. **UCI HIV dataset** is a binary classification problem with 1625 objects and 160 features, which we randomly splitted into 1300 training and 325 test objects.
3. **Synthetic data.** We generated 100 000 train and 100 000 test objects with 30 features and set the ground truth target variable to a 6-degree polynomial of the features.
4. **MovieLens 100K** is a recommender system dataset with 943 users and 1682 movies (Harper & Konstan, 2015). We followed Blondel et al. (2016a) in preparing 2703 one-hot features and in turning the problem into binary classification.

8.2 Riemannian optimization

In this experiment, we compared two approaches to training the model: Riemannian optimization (Sec. 6.1) vs. the baseline (Sec. 6). In this and later experiments we tuned the learning rate of both Riemannian and SGD optimizers with respect to the training loss after 100 iterations by the grid search with logarithmic grid.

On the Car and HIV datasets we turned off the regularization ($\lambda = 0$) and used rank $r = 4$. We report that on the Car dataset Riemannian optimization (learning rate $\alpha = 40$) converges faster and achieves better final point than the baseline (learning rate $\alpha = 0.03$) both in terms of the training and test losses (Fig. 3a, 5a). On the HIV dataset Riemannian optimization (learning rate $\alpha = 800$) converges to the value 10^{-4} around 20 times faster than the baseline (learning rate $\alpha = 0.001$, see Fig. 3b), but the model overfitts to the data (Fig. 5b).

The results on the synthetic dataset with high-order interactions confirm the superiority of the Riemannian approach over SGD – we failed to train the model at all with SGD (Fig. 6).

On the MovieLens 100K dataset, we have only used SGD-type algorithms, because using the one-hot feature encoding is much slower than using the categorical version (see Sec. 7), and we have yet to implement the support for categorical features for the Riemannian optimizer. On the bright side, prototyping the categorical version of ExM in TensorFlow allowed us to use a GPU accelerator.

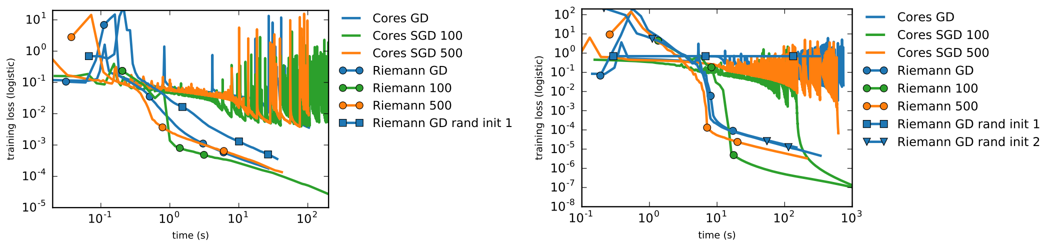
8.3 Initialization

In this experiment, we compared random initialization with the initialization from the solution of the corresponding linear problem (Sec. 6.2). We explored two ways to randomly initialize a TT-tensor: 1) filling its TT-cores with independent Gaussian noise; 2) initializing $\boldsymbol{\mathcal{W}}$ to represent a linear model with random coefficients (sampled from a standard Gaussian). We report that on the Car dataset type-1 random initialization slowed the convergence compared to initialization from the linear model solution (Fig. 3a), while on the HIV dataset the convergence was completely frozen (Fig. 3b).

Two possible reasons for this effect are: a) the vanishing and exploding gradients problem (Bengio et al., 1994) that arises when dealing with a product of a large number of factors (160 in the case of the HIV dataset); b) initializing the model in such a way that high-order terms dominate we may force the gradient-based optimization to focus on high-order terms, while it may be more stable to start with low-order terms instead. Type-2 initialization (a random linear model) indeed worked on par with the best linear initialization on the Car, HIV, and synthetic datasets (Fig. 3b, 6).

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

[2] `https://github.com/oseledets/ttpy`



(a) Binarized Car dataset

(b) HIV dataset

Figure 3: A comparison between Riemannian optimization and SGD applied to the underlying parameters of the TT-format (the baseline) for the rank-4 Exponential Machines. Numbers in the legend stand for the batch size. The methods marked with ‘rand init’ in the legend (square and triangle markers) were initialized from a random TT-tensor from two different distributions (see Sec. 8.3), all other methods were initialized from the solution of ordinary linear logistic regression. Type-2 random initialization is ommited from the Car dataset for the clarity of the figure.

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 3	0.79	65	0.2
ExM rank 8	0.85	1831	1.3
ExM rank 16	**0.96**	48879	3.8

Table 1: A comparison between models on synthetic data with high-order interactions (Sec. 8.4). We report the inference time on 100000 test objects in the last column.

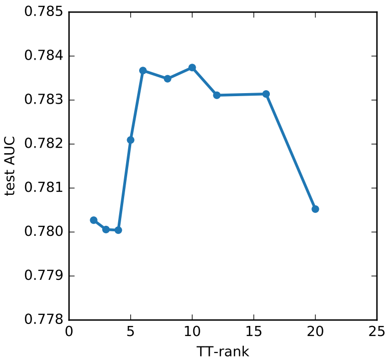


Figure 4: The influence of the TT-rank on the test AUC for the MovieLens 100K dataset.

8.4 Comparison to other approaches

On the synthetic dataset with high-order interactions we campared Exponential Machines (the proposed method) with scikit-learn implementation (Pedregosa et al., 2011) of logistic regression, random forest, and kernel SVM; FastFM implementation (Bayer, 2015) of 2-nd order Factorization Machines; our implementation of high-order Factorization Machines[3]; and a feed-forward neural network implemented in TensorFlow (Abadi et al., 2015). We used 6-th order FM with the Adam optimizer (Kingma & Ba, 2014) for which we'd chosen the best rank (20) and learning rate (0.003) based on the training 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 (Tbl. 1).

On the MovieLens 100K dataset we used the categorical features representation described in Sec. 7. Our model obtained 0.784 test AUC with the TT-rank equal 10 in 273 seconds on a Tesla K40 GPU (the inference time is 0.3 seconds per 78800 test objects); our implentation of 3-rd order FM obtained 0.782; logistic regression obtained 0.782; and Blondel et al. (2016a) reported 0.786 with 3-rd order FM on the same data.

[3] `https://github.com/geffy/tffm`

8.5 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 Sec. 9). 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 Tbl. 1).

9 RELATED WORK

Kernel SVM is a flexible non-linear predictor and, in particular, it can model interactions when used with the polynomial kernel (Boser et al., 1992). As a downside, it scales at least quadratically with the dataset size (Bordes et al., 2005) and overfits on highly sparse data.

With this in mind, Rendle (2010) developed Factorization Machine (FM), a general predictor that models pairwise interactions. To overcome the problems of polynomial SVM, FM restricts 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 (Caroll & Chang, 1970; Harshman, 1970) 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 (Rendle, 2010; Lebedev et al., 2014; Novikov et al., 2015). 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 (Meyer et al., 2011; Tan et al., 2014; Xu & Ke, 2016; Zhang et al., 2016). The last work (Zhang et al., 2016) 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.

In a concurrent work, Stoudenmire & Schwab (2016) proposed a model that is similar to ours but relies on the trigonometric basis $(\cos(\frac{\pi}{2}x), \sin(\frac{\pi}{2}x))$ in contrast to polynomials $(1, x)$ used in Exponential Machines (see Sec. 7 for an explanation on how to change the basis). They also proposed a different learning procedure inspired by the DMRG algorithm (Schollwöck, 2011), 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.

Other relevant works include the model that approximates the decision function with a multidimensional Fourier series whose coefficients lie in the TT-format (Wahls et al., 2014); and models that are similar to FM but include squares and other powers of the features: Tensor Machines (Yang & Gittens, 2015) and Polynomial Networks (Livni et al., 2014). Tensor Machines also enjoy a theoretical generalization bound. In another relevant work, Blondel et al. (2016b) 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. Note that Exponential Machines are inherently multi-convex.

10 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. However, the Riemannian learning algorithm does not support sparse data, so for dataset with hundreds of thousands of features we are forced to fall back on the baseline learning method. We found that training process is sensitive to initialization and proposed an initialization strategy based on the solution of the corresponding linear problem. The solutions developed in this paper for the stochastic Riemannian optimization may suit other machine learning models parametrized by tensors in the TT-format.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

I. Bayer. Fastfm: a library for factorization machines. *arXiv preprint arXiv:1505.00641*, 2015.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

M. Blondel, A. Fujino, N. Ueda, and M. Ishihata. Higher-order factorization machines. 2016a.

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)*. 2016b.

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.

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*, pp. 144–152, 1992.

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

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

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.

S. Holtz, T. Rohwedder, and R. Schneider. On manifolds of tensors of fixed TT-rank. *Numerische Mathematik*, pp. 701–731, 2012.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

M. Lichman. UCI machine learning repository, 2013.

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.

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

G. Meyer, S. Bonnabel, and R. Sepulchre. Regression on fixed-rank positive semidefinite matrices: a Riemannian approach. *The Journal of Machine Learning Research*, pp. 593–625, 2011.

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

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

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.

S. Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 995–1000, 2010.

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

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

M. Tan, I. W. Tsang, L. Wang, B. Vandereycken, and S. J. Pan. Riemannian pursuit for big matrix recovery. 2014.

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*, pp. 394–398. IEEE, 2014.

Z. Xu and Y. Ke. Stochastic variance reduced riemannian eigensolver. *arXiv preprint arXiv:1605.08233*, 2016.

J. Yang and A. Gittens. Tensor machines for learning target-specific polynomial features. *arXiv preprint arXiv:1504.01697*, 2015.

H. Zhang, S. J. Reddi, and S. Sra. Fast stochastic optimization on riemannian manifolds. *arXiv preprint arXiv:1605.07147*, 2016.

A Proof of Theorem 1

Theorem 1 states that the inference complexity of the proposed algorithm is $O(r^2 d)$, where r is the TT-rank of the weight tensor $\boldsymbol{\mathcal{W}}$. In this section, we propose an algorithm that achieve the stated complexity and thus prove the theorem.

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

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

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

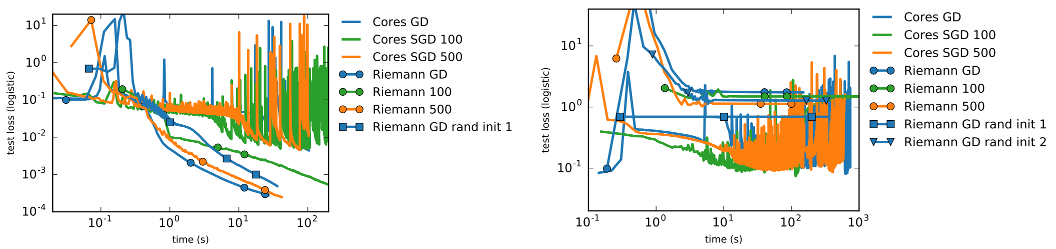
$$\begin{aligned}\widehat{y}(\boldsymbol{x}) &= \sum_{i_1,\dots,i_d} x_1^{i_1} G_1[i_1]\dots x_d^{i_d} G_d[i_d] = \left(\sum_{i_1=0}^{1} x_1^{i_1} G_1[i_1]\right)\dots\left(\sum_{i_d=0}^{1} x_d^{i_d} G_d[i_d]\right) \\ &= \underbrace{A_1}_{1\times r}\underbrace{A_2}_{r\times r}\dots\underbrace{A_d}_{r\times 1},\end{aligned}$$

where the matrices A_k for $k = 1,\dots,d$ 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 $\widehat{y}(\boldsymbol{x})$ can be computed from the matrices A_k via $d-1$ matrix-by-vector multiplications and 1 vector-by-vector multiplication, which yields $O(r^2 d)$ complexity.

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



(a) Binarized Car dataset

(b) HIV dataset

Figure 5: A comparison between Riemannian optimization and SGD applied to the underlying parameters of the TT-format (the baseline) for the rank-4 Exponential Machines. Numbers in the legend stand for the batch size. The methods marked with ‘rand init’ in the legend (square and triangle markers) were initialized from a random TT-tensor from two different distributions, all other methods were initialized from the solution of ordinary linear logistic regression. See details in Sec. 8.2 and 8.3

B PROOF OF THEOREM 2

Theorem 2 states that it is possible to initialize the weight tensor $\boldsymbol{\mathcal{W}}$ of the proposed model from the weights $\boldsymbol{w}$ of the linear model.

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

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

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

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

We start the proof with the following lemma:

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

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

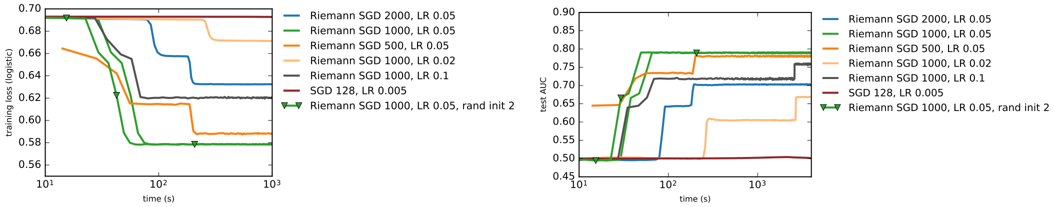
Proof. We prove the lemma by induction. Indeed, for $p = 1$ the statement of the lemma becomes

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

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 $p - 1 \ge 1$. If $i_p = 0$, then $G_p[i_p]$ is an identity matrix and $G_1[i_1] \dots G_p[i_p] = G_1[i_1] \dots G_{p-1}[i_{p-1}]$. 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:



(a) Training set

(b) Test set

Figure 6: A comparison between Riemannian optimization and SGD applied to the underlying parameters of the TT-format (the baseline) for the rank-3 Exponential Machines on the synthetic dataset with high order interactions. The first number in each legend enrty stands for the batch size. The method marked with 'rand init' in the legend (triangle markers) was initialized from a random linear model, all other methods were initialized from the solution of ordinary linear logistic regression. See details in Sec. 8.2 and 8.3

- 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 $\boldsymbol{\mathcal{W}}$ 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 $\mathcal{W}_{0\dots0} = b$.

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

C Detailed description of the datasets

We used the following datasets for the experimental evaluation

1. **UCI (Lichman, 2013) Car dataset** is a classification problem with 1728 objects and 21 binary features (after one-hot encoding). We randomly splitted the data into 1382 training and 346 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. **UCI (Lichman, 2013) HIV dataset** is a binary classification problem with 1625 objects and 160 features, which we randomly splitted into 1300 training and 325 test objects.
3. **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: $j_1^1, \dots, j_6^1, \dots, j_1^{20}, \dots, j_6^{20} \sim \mathcal{U}\{1, \dots, 30\}$. We set the ground truth target variable to a deterministic function of the input: $y(\boldsymbol{x}) = \sum_{z=1}^{20} \varepsilon_z \prod_{h=1}^{6} x_{j_h^z}$, and sampled the weights of the interactions from the uniform distribution: $\varepsilon_1, \dots, \varepsilon_{20} \sim \mathcal{U}(-1, 1)$.

4. **MovieLens 100K.** MovieLens 100K is a recommender system dataset with 943 users and 1682 movies (Harper & Konstan, 2015). We followed Blondel et al. (2016a) in preparing the features and in turning the problem into binary classification. For users, we converted age (rounded to decades), living area (the first digit of the zipcode), gender and occupation into a binary indicator vector using one-hot encoding. For movies, we used the release year (rounded to decades) and genres, also encoded. This process yielded $49+29 = 78$ additional one-hot features for each user-movie pair ($943 + 1682 + 78$ features in total). Original ratings were binarized using 5 as a threshold. This results in 21200 positive samples, half of which were used for traininig (with equal amount of sampled negative examples) and the rest were used for testing.