# Tensor Train polynomial model via Riemannian optimization

**Alexander Novikov** NOVIKOV@BAYESGROUP.RU
Skolkovo Institute of Science and Technology, Moscow, Russia
Institute of Numerical Mathematics, Moscow, Russia

**Mikhail Trofimov** MIKHAIL.TROFIMOV@PHYSTECH.EDU
Moscow Institute of Physics and Technology, Moscow, Russia

**Ivan Oseledets** I.OSELEDETS@SKOLTECH.RU
Skolkovo Institute of Science and Technology, Moscow, Russia
Institute of Numerical Mathematics, 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 version of Riemannian optimization, which allows us to fit tensors with $2^{30}$ entries. We show that the model achieves state-of-the-art performance on synthetic data with high-order interactions.

## 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). Representing the tensor of parameters in the TT-format leads to better generalization, a linear number of underlying parameters $O(d)$, and linear inference time (Sec. 5).
- We develop a stochastic Riemannian optimization learning algorithm (Sec. 7). It significantly outperforms a stochastic gradient descent baseline (Sec. 8.1) that is often used for models parametrized by a tensor decomposition (see related works, Sec. 9). To the best of our knowledge, this paper is the first to use Riemannian optimization in the stochastic regime.
- We show that the proposed method outperforms other machine learning algorithms on synthetic data with high-order interactions (Sec. 8.2).

## 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 a $d$-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.

Learning the parameters $\boldsymbol{w}, b$ of the models corresponds to minimizing the following loss

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

where $\lambda$ is the regularization parameter.

A few 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)

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

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

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

Here we assume that $0^0 = 1$ for simplicity of the notation. 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

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

Note that there is no need in a separate bias term, since it is already included in the model as the weight tensor element $\mathcal{W}_{0\ldots0}$ (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

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

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 $r \times 1$ vector. 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 $\boldsymbol{\mathcal{A}}$.

Any tensor $\boldsymbol{\mathcal{A}}$ can be robustly approximated in the TT-format with a given TT-rank $r$. Another attractive property is the ability to efficiently perform algebraic operations on tensors without materializing them, i.e. by working with the TT-cores instead of the tensors themselves. TT-format supports computing the norm of a tensor and the dot product between tensors; element-wise sum and multiplication of two tensors (the result is a tensor in the TT-format with increased TT-rank), and some others (Oseledets, 2011).

## 5. Inference

In this section, we 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 tensor $\boldsymbol{\mathcal{X}}$ has low TT-rank:

**Theorem 1.** *For any $d$-dimensional vector $\boldsymbol{x}$, the TT-rank of the corresponding tensor $\boldsymbol{\mathcal{X}}$ defined as (5) equals 1.*

*Proof.* The following cores give the exact representation of the tensor $\boldsymbol{\mathcal{X}}$

$$G_k[i_k] = x_k^{i_k} \in \mathbb{R}^{1\times 1}, \quad k = 1, \ldots, d. \qquad (7)$$

The $k$-th core $G_k[i_k]$ is a $1 \times 1$ 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 2.** *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 $\boldsymbol{\mathcal{W}}$.*

*Proof.*

$$\begin{aligned}\widehat{y}(\boldsymbol{x}) &= \sum_{i_1,\ldots,i_d} \mathcal{W}_{i_1\ldots i_d}\, \mathcal{X}_{i_1\ldots i_d} \\ &= \sum_{i_1,\ldots,i_d} G_1[i_1]\ldots G_d[i_d] \left(\prod_{k=1}^{d} x_k^{i_k}\right) \\ &= \sum_{i_1,\ldots,i_d} x_1^{i_1} G_1[i_1]\ldots x_d^{i_d} G_d[i_d] \\ &= \left(\sum_{i_1=0}^{1} x_1^{i_1} G_1[i_1]\right)\ldots\left(\sum_{i_d=0}^{1} x_d^{i_d} G_d[i_d]\right) \\ &= \underbrace{A_1}_{1\times r}\underbrace{A_2}_{r\times r}\ldots\underbrace{A_d}_{r\times 1},\end{aligned}$$

where the matrices $A_k$ for $k = 1, \ldots, d$ are defined as follows: $A_k = \sum_{i_k} x_k^{i_k} G_k[i_k] = G_k[0] + x_k G_k[1]$. The final value $\widehat{y}(\boldsymbol{x})$ can be computed via $d-1$ matrix-by-vector multiplications and 1 vector-by-vector multiplication. □

The proof of theorem 2 is constructive and corresponds to an implementation of the inference algorithm.

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:

$$\begin{aligned}&\underset{\boldsymbol{\mathcal{W}}}{\text{minimize}} \quad L(\boldsymbol{\mathcal{W}}), \\ &\text{subject to} \quad \text{TT-rank}(\boldsymbol{\mathcal{W}}) = r_0,\end{aligned} \tag{8}$$

where the loss is defined as follows

$$L(\boldsymbol{\mathcal{W}}) = \sum_{f=1}^{N} \ell\left(\langle \boldsymbol{\mathcal{X}}^{(f)}, \boldsymbol{\mathcal{W}}\rangle,\, y^{(f)}\right) + \frac{\lambda}{2}\|\boldsymbol{\mathcal{W}}\|_F^2, \tag{9}$$

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

We consider two approaches to solving problem (8). 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 $\boldsymbol{\mathcal{W}}$.

To improve upon the baseline, we exploit the geometry of the set of tensors that satisfy the TT-rank constraint (8) to build a Riemannian optimization procedure (Sec. 7). We experimentally show the advantage of this approach over the baseline in Sec. 8.1.

## 7. Riemannian optimization

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

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

forms a Riemannian manifold (Holtz et al., 2012). This observation allows us to use Riemannian optimization to solve problem (8). A typical Riemannian optimization procedure consists of repeating the following steps until convergence: project the gradient $\frac{\partial L}{\partial \boldsymbol{\mathcal{W}}}$ on the tangent space of $\mathcal{M}_r$ taken at a point $\boldsymbol{\mathcal{W}}$ to get the direction $\boldsymbol{\mathcal{G}}$; follow along $\boldsymbol{\mathcal{G}}$ with some step $\alpha$ (this operation increases the TT-rank); and retract back to the manifold $\mathcal{M}_r$ (decrease the TT-rank to $r$). In the following few paragraphs, we describe how to implement each of the steps outlined above.

The complexity of projecting a tensor $\boldsymbol{\mathcal{Z}}$ on a tangent space of $\mathcal{M}_r$ at a point $\boldsymbol{\mathcal{W}}$ is $O(dr^2(r + \text{TT-rank}(\boldsymbol{\mathcal{Z}})^2))$ (Lubich et al., 2015). We denote this projection operator as $P_{T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r}(\boldsymbol{\mathcal{Z}})$. The TT-rank of the projection is less than $2r$: $\text{TT-rank}(P_{T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r}(\boldsymbol{\mathcal{Z}})) \leq 2\,\text{TT-rank}(\boldsymbol{\mathcal{W}}) = 2r$.

Let us consider the gradient of the loss function (9)

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

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

$$P_{T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r}\left(\frac{\partial L}{\partial \boldsymbol{\mathcal{W}}}\right) = \sum_{f=1}^{N} \frac{\partial \ell}{\partial \widehat{y}} P_{T_{\boldsymbol{\mathcal{W}}}\mathcal{M}_r}(\boldsymbol{\mathcal{X}}^{(f)}) + \lambda \boldsymbol{\mathcal{W}}. \tag{11}$$

We can project all data tensors $\boldsymbol{\mathcal{X}}^{(f)}$ in parallel. Since the TT-rank of each of them equals to 1 (Theorem 1), all $N$ projections cost $O(dr^2(r+N))$. The TT-rank of the projected gradient equals $2r$ regardless of the dataset size.

To choose the step size $\alpha$, we use the generalization of Armijo rule for Riemannian optimization (Sato & Iwai, 2015). 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.

Since we aim for big datasets, we use a stochastic version of the Riemannian gradient descent: on each iteration we sample a random batch of objects from the dataset, compute the stochastic gradient for this batch, make a step

**Algorithm 1** Riemannian optimization

**Input:** Dataset $\{(\boldsymbol{x}^{(f)}, y^{(f)})\}_{f=1}^{N}$, desired TT-rank $r_0$, number of iterations $T$, batch size $M$, $0 < c_1 < 0.5$, $0 < \rho < 1$, $\beta > 1$
**Output:** $\boldsymbol{\mathcal{W}}$ that approximately minimizes (8)
Train a linear model (2) and get the parameters $\boldsymbol{w}$ and $b$
Initialize $\boldsymbol{\mathcal{W}}_0$ with a rank $r_0$ tensor built from $\boldsymbol{w}$ and $b$
Initialize $\alpha_0 = 1$
**for** $t := 1$ **to** $T$ **do**
  $\alpha_t := \beta\alpha_{t-1}$
  Sample $M$ indices $h_1, \ldots, h_M \sim \mathcal{U}(\{1, \ldots, N\})$
  $\boldsymbol{\mathcal{D}}_t := \sum_{j=1}^{M} \frac{\partial \ell}{\partial \hat{y}} \boldsymbol{\mathcal{X}}^{(h_j)} + \lambda \boldsymbol{\mathcal{W}}_{t-1}$
  $\boldsymbol{\mathcal{G}}_t := P_{T_{\boldsymbol{\mathcal{W}}_{t-1}}\mathcal{M}_r}(\boldsymbol{\mathcal{D}}_t)$ (11)
  $\boldsymbol{\mathcal{W}}_t := \text{TT-round}(\boldsymbol{\mathcal{W}}_{t-1} - \alpha_t \boldsymbol{\mathcal{G}}_t, r_0)$
  **while** $L(\boldsymbol{\mathcal{W}}_t) > L(\boldsymbol{\mathcal{W}}_{t-1}) - c_1\alpha_t \|\boldsymbol{\mathcal{G}}_t\|_F^2$ **do**
    $\alpha_t := \rho\alpha_t$
    $\boldsymbol{\mathcal{W}}_t := \text{TT-round}(\boldsymbol{\mathcal{W}}_{t-1} - \alpha_t \boldsymbol{\mathcal{G}}_t, r_0)$
  **end while**
**end for**

along the projection of the stochastic gradient, and retract back to the manifold (Alg. 1).

## 8. Experiments

We implemented the proposed algorithm in Python[1] and used the following parameters: $\rho = 0.5$, $c_1 = 0.1$, $\beta = 1.2$ for Algorithm 1. For the operations related to the TT-format, we used a Python version of the TT-Toolbox[2].

### 8.1. Riemannian optimization

In this experiment, we compared two approaches to learning the model: Riemannian optimization (Sec. 7) vs. the baseline (Sec. 6). We experimented on the Car dataset from UCI repository (Lichman, 2013), which is a binary classification problem with 1728 objects and 21 binary features (after one-hot encoding). We report that Riemannian optimization converges faster and achieves better final point than the baseline (Fig. 1).

### 8.2. Synthetic data

In this experiment, we tested our method on a dataset with high-order interactions. We generated 100 000 train and 100 000 test objects with 30 features. Each entry of the data matrix $X$ was independatly sampled from $\{-1, +1\}$ with equal probabilities 0.5. We also uniformly sampled 20 subsets of features (interactions) of order 6: $j_1^1, \ldots, j_6^1, \ldots, j_1^{20}, \ldots, j_6^{20} \sim \mathcal{U}\{1, \ldots, 30\}$. We set the ground truth target variable to a deterministic function of the input: $y(\boldsymbol{x}) = \sum_{z=1}^{20} \varepsilon_z \prod_{h=1}^{6} x_{j_h^z}$, where the weights of the interactions were sampled from a uniform distribution: $\varepsilon_1, \ldots, \varepsilon_{20} \sim \mathcal{U}(-1, 1)$.

[1] https://github.com/bihaqo/exp-machines
[2] https://github.com/oseledets/ttpy

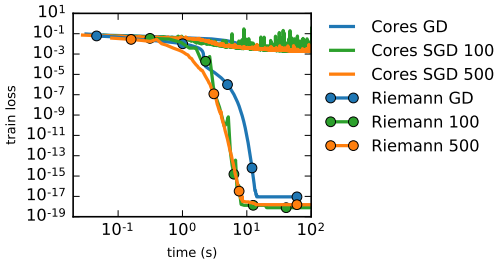


*Figure 1.* Compare Riemannian optimization with SGD applied to the underlying parameters of the TT-format (the baseline). Numbers in the legend stand for the batch size. All methods were initialized from the solution of ordinary linear logistic regression.

| Method | Test AUC | Training time (s) | Inference time (s) |
|---|---|---|---|
| Log. reg. | $0.50 \pm 0.0$ | 0.4 | 0.0 |
| RF | $0.55 \pm 0.0$ | 21.4 | 1.3 |
| SVM RBF | $0.50 \pm 0.0$ | 2262.6 | 1076.1 |
| SVM poly. 2 | $0.50 \pm 0.0$ | 1152.6 | 852.0 |
| SVM poly. 6 | $0.56 \pm 0.0$ | 4090.9 | 754.8 |
| 2-nd order FM | $0.50 \pm 0.0$ | 638.2 | 0.1 |
| 6-th order FM | $0.57 \pm 0.05$ | 1412.0 | 0.2 |
| ExM rank 2 | $0.54 \pm 0.05$ | 198.4 | 0.1 |
| ExM rank 4 | $0.69 \pm 0.02$ | 443.0 | 0.1 |
| ExM rank 8 | $\mathbf{0.75} \pm 0.02$ | 998.3 | 0.2 |

*Table 1.* A comparison between models on synthetic data with high-order interactions (Sec. 8.2). We trained each model 5 times and report the mean and standard deviation AUC across the runs.

We used scikit-learn implementation (Pedregosa et al., 2011) of logistic regression, random forest, and kernel SVM; FastFM implementation (Bayer, 2015) of 2-nd order Factorization Machine; and our implementation of Exponential Machine (our method) and high-order Factorization Machine[3]. To improve the generalization of our method, we used the dropout technique (Srivastava et al., 2014). We evaluated the performance based on the Area Under the Curve (AUC) metric since it is applicable to all methods and is robust to unbalanced labels (Table 1).

## 9. Related work

Kernel SVM translates into many non-linear predictors and in particular SVM with a polynomial kernel can model interactions (Boser et al., 1992). As a downside, it scales

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

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 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.

The main difference between our approach and high-order FM is the choice of the decomposition: we use TT-decomposition, while FM uses CP-decomposition (Caroll & Chang, 1970; Harshman, 1970), which does not allow for Riemannian optimization. We found Riemnannian optimization superior to the 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).

Another block of related works uses tensor decompositions to parametrize neural networks (Lebedev et al., 2014; Novikov et al., 2015) and to recover parameters of models, e.g. latent variable models (Anandkumar et al., 2014) and neural networks (Janzamin et al., 2015).

## 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. We found that the proposed model outperforms other machine learning algorithms on synthetic data with high-order interaction. The model, however, does not support sparse data, so it cannot scale to hundreds of thousands of features. 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. The stochastic Riemannian optimization may suit other machine learning models parametrized by tensors in the TT-format.

## References


Anandkumar, A., Ge, R., Hsu, D., Kakade, S. M., and Telgarsky, M. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15:2773–2832, 2014.

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

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

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

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

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

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

Janzamin, M., Sedghi, H., and Anandkumar, A. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *CoRR abs/1506.08473*, 2015.

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

Lichman, M. UCI machine learning repository, 2013.

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

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

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

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

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

Sato, H. and Iwai, T. A new, globally convergent riemannian conjugate gradient method. *Optimization: A Journal of Mathematical Programming and Operations Research*, pp. 1011–1031, 2015.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, pp. 1929–1958, 2014.