Marchuk Institute of Numerical Mathematics
of the Russian Academy of Sciences

as a manuscript

Alexander Novikov

Tensor methods for machine learning

PhD Dissertation Summary
for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Moscow — 2021

The PhD Dissertation was prepared at Marchuk Institute Of Numerical Mathematics of the Russian Academy of Sciences.

Academic Supervisors: Ivan V. Oseledets, Doctor of Sciences, Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences, Skolkovo Institute of Science and Technology.

Dmitry P. Vetrov, Candidate of Sciences, National Research University Higher School of Economics

Thesis topic

Relevance of the research topic. Machine learning is experiencing a period of rapid development and in recent years, significant progress has been made in many important practical tasks, such as image recognition, image segmentation, machine translation, speech recognition and synthesis, etc. However, algorithms, showing the best quality in many tasks (such as neural networks and Markov random fields) are computationally expensive both during training and during the use of the trained models. For example, a language model[1] may require a computer cluster with petabytes of RAM and a capacity of up to hudreds of petaflops for training [7]. While many corporations can afford to train such (or somewhat less expensive) models, such a high computational complexity slows down progress in machine learning and stops researchers with insufficient computational resources from working on advanced models.

Another important barrier to the spread of machine learning is computational expensivnes of applying an already trained model to new objects. This problem is especially relevant in the context of using machine learning models in mobile and embedded devices (such as smartphones and computers on board self-driving vehicles), on which significantly less computing power is available for applying the model compared to traditional computers. For example, a modern machine translation model may require relatively little resources to translate one sentence into another language: hundreds of megabytes of RAM and tens of teraflops [8]. However, such complexity turns out to be unacceptable for use on mobile devices, both in terms of speed and memory requirements, and in terms of battery use. In particular, the size of the required memory is one of the limiting factors in reducing the power consumption of the models. For example, using 45 nm CMOS technology[2] requires 0.9 pJ of power to add two 32-bit floating point numbers; accessing one 32-bit number in the SRAM cache requires 5 pJ of energy; and accessing one 32-bit number in RAM requires 640 pJ of energy, which is three orders of magnitude more than the addition operation. Large artificial neural networks do not fit into the processor's cache and thus require resource-intensive access to RAM.

This dissertation is focused on using low-rank tensor decomposition algorithms to develop faster methods for training models, methods for compressing and accelerating models, and creating less resource-intensive machine learning models from scratch.

Purpose of the thesis. The purpose of this dissertation is to develop new machine learning models based on the apparatus of tensor decompositions, which are not inferior to their counterparts in prediction accuracy, but surpass analogs in terms of speed

[1] A language model allows us to estimate the probability of the appearance of various sentences in a natural language, which in turn is required to solve machine translation and create chat bots.

[2] Complementary metal-oxide-semiconductor.

and (or) compactness, as well as to develop new methods for learning and applying existing models that are ahead of analogues in terms of speed. An additional goal is to develop a software package that simplifies further research at the intersection of machine learning and tensor methods.

Scientific novelty. A new method for estimating the partition function and marginal distributions of a Markov random field (statistics used both for training Markov random fields and for applying them to new objects) is proposed, which significantly exceeds its analogues in terms of accuracy. Theoretical results have been obtained on evaluating the accuracy of the proposed methods.

A neural network architecture based on tensor decompositions is proposed, which is orders of magnitude more compact (has less adjustable parameters with comparable predictive accuracy) than alternatives. A tensor based model is proposed that can take into account high order interactions of features and allows for training using stochastic Riemannian optimization, which leads to faster training of the model compared to training by classical methods such as stochastic gradient descent applied to parameters. The developed Riemannian optimization method uses the structure of the model and the optimized function to achieve high speed.

The first method for the automatic calculation of the Riemannian gradient and the product of the approximate Riemannian Hessian by a vector for functions depending on tensors in the Tensor Train format with an asymptotic complexity of work equal to the complexity of calculating the value of a given function is proposed. The developed method can be used to train neural network models proposed in this work.

Practical value. The methods developed in this dissertation can be applied to solve a wide range of problems (such as training models for image classification and segmentation, speech analysis and synthesis, machine translation, etc.), obtaining models that are faster and more compact compared to alternatives, thus allowing to use a wider range of models on mobile and embedded devices. Also, the use of the developed software package makes it possible to significantly simplify the development of new methods of Riemannian optimization for a variety of tensors of a fixed TT-rank due to the support of Riemannian automatic differentiation.

Key aspects/ideas to be defended. The main results of the work are methods for effective training and application of trained machine learning models, and the application of the developed methods to several applied problems, as well as a library that simplifies further work at the intersection of tensor methods and machine learning.

1. Library for working with Tensor Train decomposition in Python with support for GPUs;
2. A method for the automatic calculation of the Riemannian gradient and the product of the Riemannian Hessian (of a function given in the form of a computer program) by a vector with asymptotic complexity of work equal to the complexity of calculating the value of the function in a single point;

3. A computational method for estimating the partition function and marginal distributions of a Markov random field, and a theoretical estimate of the accuracy of its operation;
4. A neural network architecture with parameter layers represented with tensor decompositions. The model was tested on the problem of image classification and it was shown that it allows one to significantly outperform analogues in the ratio of the number of parameters to the quality of work;
5. Machine learning model that can take into account high-order interactions of features is proposed and that allows using Riemannian optimization for faster training.

Publications and probation of the work.
First-tier publications

[1] Novikov A. V., Izmailov P. A., Khrulkov V. A., Figurnov M. V., Oseledets, I. V. Tensor Train Decomposition on TensorFlow (T3F). // Journal of Machine Learning Research, Machine Learning Open Source Software. – 2020. – № 21. – С. 1–7. Q2 Web of Science.

[2] Rakhuba M. V., Novikov A. V., Oseledets I. V. Low-rank Riemannian eigensolver for high-dimensional Hamiltonians // Journal of Computational Physics. – 2019. – № 396. – С. 718–737. In this work, the author's contribution to the defense consists in writing code using the developed library and conducting experiments. Q1 Web of Science.

[3] Novikov, A. V., Podoprikhin D. A., Osokin A. A., Vetrov D. P. Tensorizing neural networks // Advances in Neural Information Processing Systems (NeurIPS). – 2015. – № 28. – С. 442–450. A^*CORE.

[4] Novikov, A. V., Rodomanov, A. O., Osokin, A. A., Vetrov D. P. Putting MRFs on a Tensor Train // International Conference on Machine Learning (ICML). – 2014. A^* CORE.

Second-tier publications

[5] Novikov A. V., Trofimov M. I., Oseledets, I. V. Exponential machines // Bulletin of the Polish Academy of Sciences Technical Sciences, special issue on Deep Learning: Theory and Practice. – 2018. – № 6. – С. 789–797. Q3 Web of Science.

Third-tier publications

[6] Novikov, A. V., Rodomanov, A. O., Osokin, A. A., Vetrov D. P. Putting MRFs on a Tensor Train // Intelligent Systems, theory and practice, pp. 293–318, 2014.

Personal contribution of the author. This work is an independent work of the author. The author personally implemented the T3F library, conducted experiments and wrote a paper describing the program complex [1], proposed and implemented

a method of automatic Riemannian differentiation, proposed and implemented a machine learning method which takes into account high-order feature interactions [5], implemented and tested in numerical experiments a model of an artificial neural network that uses tensor decomposition to compress the weights of the network [3]. The idea of a method for estimating the partition function of a Markov random field [4; 6] was developed jointly with D. P. Vetrov and I. V. Oseledts, while the author personally implemented the developed method and proved theoretical guaranties. The formulation of the problem in [3; 4; 6] was performed by D. P. Vetrov. The formulation of the problem in [2] and in the work on automatic Riemanian differentiation was performed by I. V. Oseledts. Also, the author has implemented a method for finding eigenvalues of matrices in the TT-format in the application to the search for the vibration spectrum of molecules [2].

Content of work

The **introduction** explains the relevance of the research carried out in this disseratation, provides a review of the scientific literature on the problem under study, formulates the goal of the study, lists the tasks of the work, and describes the scientific novelty and the practical relevance of the work presented.

The **first chapter** describes the concepts used in the dissertation work: tensor decompositions, Tensor Train (TT) decomposition, Riemannian optimization, machine learning and automatic differentiation.

We call multidimensional array of numbers indexed by the vector-index $(i_1, \ldots, i_d)$, $i_k \in \{1, \ldots, n_k\}$, $k = 1, \ldots, d$ as *d-dimensional tensor* and denote it as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$. The length d of the vector-index is called the *tensor dimensionality*. Tensors are denoted with bold uppercase Gothic letters (eg $\boldsymbol{\mathcal{A}}$).

A d-dimensional tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ is said to be represented in the *TT-format* [9] if for all $k = 1, \ldots, d$ and all for each possible value of the index $i_k = 1, \ldots, n_k$ ($n = \max k = 1 \ldots, dn_k$) there are exist matrices $\boldsymbol{G}_k^{\boldsymbol{\mathcal{A}}}[i_k]$, such that each element of the tensor $\boldsymbol{\mathcal{A}}$ can be represented as a product of these matrices:

$$\mathcal{A}_{i_1,\ldots,i_d} = \boldsymbol{G}_1^{\boldsymbol{\mathcal{A}}}[i_1]\boldsymbol{G}_2^{\boldsymbol{\mathcal{A}}}[i_2] \cdots \boldsymbol{G}_d^{\boldsymbol{\mathcal{A}}}[i_d]. \tag{1}$$

All matrices related to the same index k are constrained to have the same size $\mathrm{r}_{k-1}(\boldsymbol{\mathcal{A}}) \times \mathrm{r}_k(\boldsymbol{\mathcal{A}})$. To make the resulting matrix product (1) a number, (the corresponding element of the tensor $\boldsymbol{\mathcal{A}}$), we set $\mathrm{r}_0(\boldsymbol{\mathcal{A}}) = \mathrm{r}_d(\boldsymbol{\mathcal{A}}) = 1$.

To improve the efficiency of working with vectors and matrices, the concepts of TT-format of a vector and TT-format of a matrix are introduced in a special way. The *TT-representation of a vector $\boldsymbol{b}$* is defined as the TT-representation of the tensor $\boldsymbol{\mathcal{B}}$, which contains all the elements of the vector $\boldsymbol{b}$ in lexicographic order. In other words, let there be a one-to-one mapping f between the indices of the vector $\boldsymbol{b} \in \mathbb{R}^{n^d}$ and the d-dimensional vectors $\boldsymbol{j} = (j_1, \ldots, j_d)$. Then the tensor $\boldsymbol{\mathcal{B}}$ can be defined as $\mathcal{B}_{j_1,\ldots,j_d} = b_{f^{-1}(j_1,\ldots,j_d)}$.

Let's define the concept of TT-format for matrices. Let there exist one-to-one mappings g and f between the indices of rows and columns of the matrix $\boldsymbol{M}$ into d-dimensional vectors $\boldsymbol{i}$ and $\boldsymbol{j}$, respectively. We denote as $\boldsymbol{\mathcal{R}}$ the $2d$-dimensional tensor containing all elements of the matrix $\boldsymbol{M}$ in the order determined by the one-to-one mappings: $\mathcal{R}(i_1, \ldots, i_d, j_1, \ldots, j_d) = M(g^{-1}(i_1, \ldots, i_d), f^{-1}(j_1, \ldots, j_d))$. Lets reorder the indices of the tensor $\boldsymbol{\mathcal{R}}$ as follows: $(i_1, j_1, i_2, j_2, \ldots, i_d, j_d)$, group the adjacent pairs of indices $(i_1 j_1, i_2 j_2, \ldots, i_d j_d)$ and represent the resulting d-dimensional tensor $\boldsymbol{\mathcal{W}}$ in TT-format:

$$\mathcal{R}(i_1, \ldots, i_d, j_1, \ldots, j_d) = \mathcal{W}(i_1 j_1, \ldots, i_d j_d) = \boldsymbol{G}_1^{\boldsymbol{\mathcal{W}}}[i_1, j_1] \ldots \boldsymbol{G}_d^{\boldsymbol{\mathcal{W}}}[i_d, j_d],$$

where $\boldsymbol{G}_k^{\boldsymbol{\mathcal{W}}}$, $k = 1, \ldots, d$ are called *TT-cores* and $\boldsymbol{G}_k^{\boldsymbol{\mathcal{M}}}[i_k, j_k]$ are matrices for any value of the indices $i_k = 1, \ldots, m_k$, $j_k = 1, \ldots, n_k$. *The TT-representation of the matrix* $\boldsymbol{M}$ is defined as the TT-representation of the tensor $\boldsymbol{\mathcal{W}}$. Note that a matrix represented in the TT-format does not have to be square, since i_k and j_k can take a different number of possible values.

For a matrix $\boldsymbol{M}$ and a vector $\boldsymbol{b}$ represented in the TT-format, it is possible to efficiently calculate the product $\boldsymbol{c} = \boldsymbol{M}\boldsymbol{b}$. The result of this operation is a vector $\boldsymbol{c}$ in the TT-format with TT-ranks equal to the product of the TT-ranks of the matrix $\boldsymbol{M}$ and the vector $\boldsymbol{b}$: $\mathrm{r}_k(\boldsymbol{c}) = \mathrm{r}_k(\boldsymbol{M})\, \mathrm{r}_k(\boldsymbol{b})$.

Section D1.4 is devoted to the Riemannian optimization methods.

The set of d-dimensional tensors of fixed size and with fixed TT-ranks r

$$\mathcal{M}_{\mathrm{r}} = \{\boldsymbol{\mathcal{W}} \in \mathbb{R}^{n_1 \times \ldots \times n_d} \colon \text{TT-rank}(\boldsymbol{\mathcal{W}}) = \mathrm{r}\}$$

forms a Riemannian manifold [10].

Lets focus on the following optimization problem

$$\min_{\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}} f(\boldsymbol{\mathcal{X}}).$$

Lets assume, that there is a priori information that the solution of this problem can be approximated by a tensor in the TT-format, i.e. an element of the Riemannian manifold $\boldsymbol{\mathcal{X}}_* \in \mathcal{M}_r$. Then we can reformulate the optimization problem as

$$\min_{\boldsymbol{\mathcal{X}} \in \mathcal{M}_r} f(\boldsymbol{\mathcal{X}}).$$

In Riemannian optimization problems, the domain $\mathbb{R}^{n_1 \times \cdots \times n_d}$ is replaced by the Riemannian manifold $\mathcal{M}_r$. Riemannian optimization methods usually work at each iteration with the linearized version of the manifold around the current point $\boldsymbol{\mathcal{X}}$, which is called the *tangent space*. Intuitively, the tangent space can be thought of as a plane orthogonal to the surface $\mathcal{M}_r$ at a given point $\boldsymbol{\mathcal{X}}$.

Riemannian optimization algorithms usually compute the *Riemannian gradient* $\operatorname{grad} f(\boldsymbol{\mathcal{X}})$, which for a smooth submanifold $\mathcal{M}_{\boldsymbol{r}} \subset \mathbb{R}^{n_1 \times \cdots \times n_d}$ can be represented as the projection of the Euclidean gradient $\nabla f(\boldsymbol{\mathcal{X}})$ onto the tangent space $T_{\boldsymbol{\mathcal{X}}}\mathcal{M}_r$ of the manifold $\mathcal{M}_{\boldsymbol{r}}$:

$$\operatorname{grad} f(\boldsymbol{\mathcal{X}}) = \mathrm{P}_{\boldsymbol{\mathcal{X}}} \nabla f(\boldsymbol{\mathcal{X}}), \tag{2}$$

where $\mathrm{P}_{\boldsymbol{\mathcal{X}}} : \mathbb{R}^{n_1 \times \cdots \times n_d} \to T_{\boldsymbol{\mathcal{X}}}\mathcal{M}_{\boldsymbol{r}}$ denotes the orthoprojection operator onto the tangent space $T_{\boldsymbol{\mathcal{X}}}\mathcal{M}_{\boldsymbol{r}}$. Section D3.4.2 proposes a method for automatically calculating the Riemannian gradient of a given function.

Also, some Riemannian optimization methods use second-order information, i.e. work with a Riemannian Hessian. Riemannian Hessian $\operatorname{Hess} f(\boldsymbol{\mathcal{X}}) : T_{\boldsymbol{\mathcal{X}}}\mathcal{M}_{\boldsymbol{r}} \to T_{\boldsymbol{\mathcal{X}}}\mathcal{M}_{\boldsymbol{r}}$ can be defined in the following way

$$\operatorname{Hess} f(\boldsymbol{\mathcal{X}}) = \mathrm{P}_{\boldsymbol{\mathcal{X}}} \left(\nabla^2 f(\boldsymbol{\mathcal{X}})\right) \mathrm{P}_{\boldsymbol{\mathcal{X}}} + \mathrm{P}_{\boldsymbol{\mathcal{X}}} \dot{\mathrm{P}}_{\boldsymbol{\mathcal{X}}} (\nabla f(\boldsymbol{\mathcal{X}})) \mathrm{P}_{\boldsymbol{\mathcal{X}}}, \tag{3}$$

where $\nabla^2 f(\boldsymbol{\mathcal{X}})$ is the Euclidean Hessian, and $\dot{\mathrm{P}}_{\boldsymbol{\mathcal{X}}}$ denotes the Frechet derivative of the operator $\mathrm{P}_{\boldsymbol{\mathcal{X}}}$.

The second term of the Riemannian Hessian (3) for the low-rank matrix manifold depends on the inverse eigenvalues of the matrix and its calculation can be numerically unstable when using an overestimated rank. Therefore, the second term is often discarded and the *lianerized Riemannian Hessian* is considered:

$$\mathrm{P}_{\boldsymbol{\mathcal{X}}} \left(\nabla^2 f(\boldsymbol{\mathcal{X}})\right) \mathrm{P}_{\boldsymbol{\mathcal{X}}}. \tag{4}$$

Effectively implementing (2) and (4) for various functions f can be challenging. Note that calculating the Euclidean gradient $\nabla f(\boldsymbol{\mathcal{X}})$ and then projecting it onto the tangent space $\mathrm{P}_{\boldsymbol{\mathcal{X}}}$ does not lead to an efficient algorithm in most cases.

Section D1.5 introduces basic machine learning concepts.

Let a dataset of independent identically distributed training pairs $\{(\boldsymbol{x}^{(f)}, y^{(f)})\}_{f=1}^{N} \sim p(\boldsymbol{x}, y)$ be given, where object number f is described by a vector $\boldsymbol{x}^{(f)}$ of length d, and also a target variable $y^{(f)}$ is associated with each object. Let also the loss function $\ell(\widehat{y}, y) : \mathbb{R}^2 \to \mathbb{R}$ be given, which maps the predicted target variable $\widehat{y}$ and the correct answer y (taken from the training dataset) into a real value representing the error made during the prediction. The goal of the supervised learning problem is to find a function h that maps feature descriptions $\boldsymbol{x}$ into predictions y, approximating the minimum of the loss function averaged over the training distribution:

$$h \approx \arg\min_{g(\boldsymbol{x})} \mathbb{E}_{p(\boldsymbol{x},y)} \ell(g(\boldsymbol{x}), y) \tag{5}$$

Since in the problem statement the distribution $p(\boldsymbol{x}, y)$ is not given, but only the finite training set $\{(\boldsymbol{x}^{(f)}, y^{(f)})\}_{f=1}^{N}$ is given, the problem (5) is often replaced by the problem of *empirical risk minimization* among some subclass of functions $\mathcal{F}$

$$h \approx \arg\min_{g(x) \in \mathcal{F}} \frac{1}{N} \sum_{f=1}^{N} \ell(g(\boldsymbol{x}^{(f)}), y^{(f)}) \tag{6}$$

The need to introduce a subclass of functions $\mathcal{F}$ is due to the *overfitting problem*: the set of global empirical risk minimizers (6) contains functions that memorize the dataset, that is, they return correct answers on the objects of the training dataset, and random

ones on all other (new objects). The class of functions $\mathcal{F}$ is chosen so that it does not contain such undesirable solutions.

In practice, $\mathcal{F}$ is often used as a class of functions *artificial neural networks* (ANN or NN). Artificial neural networks are usually defined as the composition of a certain class of basic functions that are continuous and (almost everywhere) differentiable. A popular chose is using the class of linear mappings and element-by-element application of the sigmoid function $\sigma(x) = \frac{1}{1+\exp(x)}$ or the ReLU function $f(x) = \max\{0, x\}$ are considered as the base class of functions. An example of a neural network can be the following class of functions, specified up to adjustable parameters $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{w}_3, \boldsymbol{b}_1, \boldsymbol{b}_2, b_3$

$$h(\boldsymbol{x}) = \langle \boldsymbol{w}_3, \sigma(\boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)\rangle + b_3$$

Section D1.6 introduces the concept of automatic differentiation.

Automatic differentiation [11] is a technique for computing the gradient value of a function $f : \mathbb{R}^d \to \mathbb{R}$ given as a software implementation. In particular, suppose that function f can be implemented as a sequence of elementary operations, such as additions and multiplications, trigonometric operations, taking the logarithm, etc. The implementation of the function f can also include more complex operations, such as calculating matrix decompositions, if the derivatives of these operations are known. When this assumption is satisfied, automatic differentiation allows calculating the gradient of the function f with machine precision and with a computational complexity only a constant number of times greater than the complexity of calculating the value of the function f at a single point (i.e., the gradient is calculated with the same asymptotic complexity as the calculation of the function value at a single point). In the machine learning community, automatic differentiation is sometimes referred to as *backpropagation* [12].

Note that automatic differentiation is not the same as numerical differentiation by the finite difference method, where the components of the gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ are approximated by the difference in the value of the function at two points:

$$\frac{\partial f(x_1, \ldots, x_d)}{\partial x_k} \approx \frac{f(x_1, \ldots, x_{k-1}, x_k + h, x_{k+1}, \ldots, x_d) - f(x_1, \ldots, x_d)}{h}, \quad (7)$$

where h is chosen so that the approximation error is small enough. Numerical differentiation is computationally more expensive than automatic differentiation. Indeed, let us denote the complexity of calculating the value of a function at a single point by $\mathcal{O}(F)$. Expression (7) requires calculating the value of the function at $d+1$ points in order to approximate the value ∇f, which means that the asymptotic complexity of the work is equal to $\mathcal{O}(dF)$. Moreover, automatic differentiation is numerically more robust than numerical differentiation. Due to round-off errors that occur when subtracting numbers that are close in value, numerical differentiation (7) suffers from accumulation of errors, which does not allow calculating the gradient with an accuracy greater than the

square root of the machine precision, while the automatic differentiation maintains the machine precision.

In Section D3.4, a new method of Riemannian automatic differentiation is proposed, which makes it possible to automatically calculate the Riemannian gradient of a given function with optimal asymptotic complexity on the manifold of low rank matrices and low TT-rank tensors.

The second chapter is devoted to the development of new methods for working with Markov random fields (MRF) – a machine learning method designed to work with distributions on structured objects (an example of a structured object is an image, while an example of an unstructured one is a real number). For example, for a semantic segmentation problem, Markov random fields allow you to define and train a model that, for a given image, returns a probability distribution over all possible segmentation (the number of which is equal to the number of possible classes in the power of the number of image pixels) and allows you to efficiently perform some operations on this distribution, for example find the most likely segmentation. However, due to the curse of dimensionality, there are no efficient algorithms for many operations such as calculating the partition function (the normalization constant). After reviewing the basic concepts required for the introduction of Markov random fields (Section D2.1), the author proposes a new method for estimating the statistics of a Markov random field required for its training and further use, based on the idea of interpreting the probability distribution as a tensor and using the structure of the TT-decomposition of this tensor (section D2.3).

Consider a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a finite set of vertices $\mathcal{V}$ and hyperedges $\mathcal{E}$. Suppose that all vertices are numbered from 1 to n, and all hyperexes from 1 to m.

Let us assign to each vertex $k = 1, \ldots, d$ a variable y_k taking values from the set $\mathcal{Y}_k = \{1, \ldots, n_k\}$. Let us denote the set of vertices included in the hyperedge $\ell = 1, \ldots, m$ by $\boldsymbol{y}^\ell = \{y_{N_v}\}_{v=1}^{q_\ell}$. To each hyperedge $\ell = 1, \ldots, m$ we associate a real-valued function Θ_ℓ defined on the common domain of variables from $\boldsymbol{y}^\ell$, that is, $\Theta_\ell : \prod_{v=1}^{q_\ell} \mathcal{Y}_{N_v} \to \mathbb{R}$. The function Θ_ℓ will be called *potential*.

A Markov random field (MRF) on a hypergraph $\mathcal{G}$ has an associated *energy function*, defined as the sum of all potentials: $E(\boldsymbol{y}) = \sum_{\ell=1}^{m} \Theta_\ell(\boldsymbol{y}^\ell)$.

The exponent of minus energy is called *unnormalized Gibbs distribution*: $\widehat{P}(\boldsymbol{y}) = \exp(-E(\boldsymbol{y}))$. The symbol Z, $Z = \sum_{\boldsymbol{y}} \widehat{P}(\boldsymbol{y})$ will be used to denote the *partition function* (normalization constant). The functions $\Psi_\ell(\boldsymbol{y}^\ell) = \exp(-\Theta_\ell(\boldsymbol{y}^\ell))$ will be called *factors* of a Markov random field.

Both the energy and the unnormalized probability are d-dimensional tensors. The potentials and the factors become d-dimensional tensors as well if we add non-essential variables for non-existing dimensions: $\boldsymbol{\Theta}_\ell(\boldsymbol{y}^\ell) = \boldsymbol{\Theta}_\ell(\boldsymbol{y})$. Energy and probability tensors can be defined as follows: $\boldsymbol{\mathcal{E}} = \sum_{\ell=1}^{m} \boldsymbol{\Theta}_\ell$ and $\widehat{\boldsymbol{\mathcal{P}}} = \bigodot_{\ell=1}^{m} \boldsymbol{\Psi}_\ell$.

In the example of semantic image segmetation, the potential function of the MRF can be defined in the following way: *unary potentials* (that is, the potentials depending on only one variable for each pixel) $\boldsymbol{\Theta}_k(y_k)$ set the logarithm of confidence of the base

classifier that the k-th pixel belongs to the class y_k; the binary potentials defined on the edges of the graph $\boldsymbol{\Theta}_{k_i,k_j}(y_{k_i}, y_{k_j})$ set a priori knowledge about the correlation of labels in neighboring pixels (for example, the Potts potential $\boldsymbol{\Theta}_{k_i,k_j}(y_{k_i}, y_{k_j}) = [y_{k_i} \neq y_{k_j}]$ penalizes the model for too many boundaries between classes [13]). Having specified the model of a Markov random field, one can, for example, use the maximum likelihood method for training it, that is, to maximize the likelihood of the training dataset $(\boldsymbol{x}^{(f)}, \boldsymbol{y}^{(f)})_{f=1}^{N}$ w.r.t. parameters $\boldsymbol{w}$

$$\max_{\boldsymbol{w}} \prod_{f=1}^{N} \frac{1}{Z(\boldsymbol{w}, \boldsymbol{x}^{(f)})} \exp\left(- \sum_{\ell=1}^{m} \boldsymbol{\Theta}(\boldsymbol{y}^{(f)}, \boldsymbol{x}^{(f)}, \boldsymbol{w}) \right)$$

where the partition function $Z(\boldsymbol{w}, \boldsymbol{x}^{(f)})$ is defined as the sum over an exponentially large number of possible segmentation

$$Z(\boldsymbol{w}, \boldsymbol{x}) = \sum_{\boldsymbol{y}} \exp\left(- \sum_{\ell=1}^{m} \boldsymbol{\Theta}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) \right)$$

Thus, one of the main problems in the theory and practice of using Markov random fields is the approximate calculation of the partition function Z. Note that the exact calculation of the partition function in the general case is a #P-hard problem.

After training, the MRF can be applied to a new image by finding the most probable segmentation $\boldsymbol{x}$

$$\arg\max_{\boldsymbol{y}} \frac{1}{Z(\boldsymbol{w}, \boldsymbol{x})} \exp\left(- \sum_{\ell=1}^{m} \boldsymbol{\Theta}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) \right) = \arg\min_{\boldsymbol{y}} \sum_{\ell=1}^{m} \boldsymbol{\Theta}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w})$$

Another way to apply a trained MRF to new images is to calculate the *marginal distributions*, that is, the distributions for segmentation of individual pixels obtained from the conditional probability for the segmentation of all pixels by summing over all pixels except for the given one.

$$p(y_k \mid \boldsymbol{x}, \boldsymbol{w}) = \sum_{y_1=1}^{n} \dots \sum_{y_{k-1}=1}^{n} \sum_{y_{k+1}=1}^{n} \dots \sum_{y_d=1}^{n} p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})$$

Thus, the problems of discrete minimization of energy given in the form of a sum of potentials, as well as the problem of calculating marginal distributions, are the among the most important ones for using MRFs in practice.

Section D2.3 proposes a new algorithm for calculating the partition function and marginal distributions based on the decomposition of each factor of the MRF in the TT-format and then combining the TT-cores of the decomposed factors in a special way. For the proposed algorithm, the following estimate of the accuracy is shown.

***Theorem* 1.** *For any set of factors $\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_m$ and any value of the rounding accuracy parameter $\varepsilon \geq 0$, the absolute error of the estimation of the partition function calculated by the algorithm D1 does not exceed:*

$$\begin{aligned}\left|Z - \widehat{Z}\right| \leq \|\boldsymbol{B}_1\|_2 \cdots \|\boldsymbol{B}_{n-2}\|_2 \cdot \left\|\boldsymbol{B}_{d-1}\boldsymbol{f}_d - \boldsymbol{f}_{d-1}\right\|_2 + \\ + \|\boldsymbol{B}_1\|_2 \cdots \|\boldsymbol{B}_{d-3}\|_2 \cdot \left\|\boldsymbol{B}_{d-2}\boldsymbol{f}_{d-1} - \boldsymbol{f}_{d-2}\right\|_2 + \cdots + \\ + \|\boldsymbol{B}_1\boldsymbol{f}_2 - \boldsymbol{f}_1\|_2\end{aligned}$$

where the matrices $\boldsymbol{B}_k$ represent the results of intermediate calculations arising in the process of executing the algorithm D1.

Using this theorem, it is possible to estimate the accuracy of the value obtained by the proposed algorithm, as well as to select the rounding accuracy ε required to achieve the required accuracy of the estimate of the partition function by using binary search.

The third chapter is devoted to the development of new neural network models based on the factorization of the tensors of parameters into the TT-format, as well as the development of methods for their training.

As discussed in section D1.5, artificial neural networks (NN) can be parameterized by millions and in some cases billions [7] real-valued parameters organized into tensors corresponding to individual layers. In addition to a large number of parameters, modern NNs are trained on huge datasets sometimes consisting of terabytes of data of various modalities (for example, a film can be simultaneously described by a video sequence and a textual description of the plot). This multi-modal data is often naturally represented in the form of tensors. Thus, when teaching and using NNs in various contexts, tensors of large dimensions emerge. This chapter deals with the tasks of compressing, accelerating (and therefore reducing the power consumption), and training artificial neural networks. Section D3.2 proposes a tensor approach to the compression of neural networks; Section D3.3 proposes a machine learning model that takes into account the interactions of high-order features that can be trained using Riemannian optimization, which allows achieving faster training compared with analogs, and in Section D3.4, I propose the first method for automatic Riemannian differentiation for the manifold of tensors of fixed TT-rank, which simplifies the further development of Riemannian optimization methods in the context of machine learning.

Section D3.3 proposes a machine learning model that takes into account the interactions of high-order features, all parameters of which are specified as a single tensor in the TT-format. Suppose that a training example is represented as a d-dimensional vector $\boldsymbol{x}$. Let us associate each subset of features with a d-dimensional binary vector $(i_1, \ldots, i_d)$, where $i_k = 1$ if and only if the k-th feature belongs to this subset. Using these notation, the proposed model is given by the following equation:

$$\widehat{y}(\boldsymbol{x}) = \sum_{i_1=0}^{1} \cdots \sum_{i_d=0}^{1} \mathcal{W}_{i_1 \ldots i_d} \prod_{k=1}^{d} x_k^{i_k}. \tag{8}$$

Further in section D3.3 it is shown that the proposed model allows fast inference (computing the value (8) predicted by the model on an object) and allows training using Riemannian optimization. A method is proposed for efficient calculation of the Riemannian gradient of empirical risk using the structure of the equation of the model (8). Also in section D3.3.3 a method for initializing the model is proposed that accelerates the convergence of training.

Section D3.4 is devoted to the development of a method for automatic Riemannian differentiation.

As can be seen from section D3.3.3, even for relatively simple models, Riemannian optimization requires non-trivial calculations to derive and implement an effective optimization method. The situation becomes much more complicated when using second-order optimization methods, which require an efficient implementation of the product of the Riemannian Hessian by a given vector.

On the other hand, in the deep learning community, the question of implementing an optimization method for a particular model rarely arises, since the overwhelming majority of researchers use automatic differentiation. Automatic differentiation is increasingly used in numerical methods and is considered one of the fundamental tools that led to the success of deep learning, allowing one to combine artificial neural networks from increasingly complex building blocks without worrying about calculating derivatives or implementing optimization methods.

In Section D3.4, I propose a method for automatic differentiation on the manifold of matrices of fixed rank and on the manifold of tensors of fixed TT-rank. The proposed method makes it possible to significantly simplify the implementation of Riemannian optimization methods, since it allows to automatically calculate the Riemannian gradient and the product of the linearized Riemannian Hessian by a given vector. The asymptotic complexity of the proposed method for calculating the Riemannian gradient and the product of the linearized Hessian and a vector equals to the asymptotic complexity of calculating the value of the given function at a single point.

Let us illustrate the idea of the proposed method using the example of a manifold of low-rank matrices (in Section D3.4.2 this result is generalized to the case of tensors of fixed TT-rank).

Let us denote the Euclidean gradient of a function $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ at a point $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\intercal$ as

$$\frac{\partial f}{\partial \boldsymbol{X}} = \left\{ \frac{\partial f}{\partial X_{ij}} \right\}_{i,j=1}^{m,n}.$$

The problem under consideration is to efficiently calculate the Riemannian gradient $\operatorname{grad} f(\boldsymbol{X})$

$$\operatorname{grad} f(\boldsymbol{X}) = P_{\boldsymbol{X}} \frac{\partial f}{\partial \boldsymbol{X}} \in T_{\boldsymbol{X}}\mathcal{M}_r \subset \mathbb{R}^{m\times n}$$

A naive way to calculate this gradient is to first calculate the Euclidean gradient $\partial f/\partial \boldsymbol{X}$ using classical automatic differentiation and then project it onto the tangent

space built around the current point using the following formula.

$$P_{\boldsymbol{X}} \frac{\partial f}{\partial \boldsymbol{X}} = \frac{\partial f}{\partial \boldsymbol{X}} \boldsymbol{V}\boldsymbol{V}^\intercal + \boldsymbol{U}\boldsymbol{U}^\intercal \frac{\partial f}{\partial \boldsymbol{X}} (\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^\intercal). \tag{9}$$

However, this approach requires the calculation of the Euclidean gradient $\partial f / \partial \boldsymbol{X}$ which has the size $m \times n$ and therefore cannot be calculated faster than with asymptotic complexity $\mathcal{O}(mn)$, which is unacceptable in many practical problems. The author proposes a method for calculating the Riemannian gradient without explicetly forming the Euclidean gradient $\partial f / \partial \boldsymbol{X}$. Namely, note that the Riemannian gradient (9) requires the calculation of the following quantities

$$\left(\frac{\partial f}{\partial \boldsymbol{X}} \boldsymbol{V} \right) \in \mathbb{R}^{m \times r}, \quad \left(\boldsymbol{U}^\intercal \frac{\partial f}{\partial \boldsymbol{X}} \right) \in \mathbb{R}^{r \times n},$$

which can be calculated as partial derivatives of the following auxiliary function

$$g(\boldsymbol{A}, \boldsymbol{B}) \triangleq f(\boldsymbol{A}\boldsymbol{V}^\intercal + \boldsymbol{U}\boldsymbol{B}^\intercal).$$

Indeed, since $\boldsymbol{X}$ can be represented as

$$\boldsymbol{X} = (\boldsymbol{U}\boldsymbol{S}) \cdot \boldsymbol{V}^\intercal + \boldsymbol{U} \cdot \mathbf{O}_{n \times r}^\intercal,$$

we get

$$\left. \frac{\partial g}{\partial \boldsymbol{A}} \right|_{\substack{\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}, \\ \boldsymbol{B} = \mathbf{O}_{n \times r}}} = \sum_{i,j} \frac{\partial f}{\partial X_{ij}} \frac{\partial X_{ij}}{\partial \boldsymbol{A}} = \frac{\partial f}{\partial \boldsymbol{X}} \boldsymbol{V},$$

$$\left. \frac{\partial g}{\partial \boldsymbol{B}^\intercal} \right|_{\substack{\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}, \\ \boldsymbol{B} = \mathbf{O}_{n \times r}}} = \sum_{i,j} \frac{\partial f}{\partial X_{ij}} \frac{\partial X_{ij}}{\partial \boldsymbol{B}} = \boldsymbol{U}^\intercal \frac{\partial f}{\partial \boldsymbol{X}}.$$

Thus, the low-rank representation of the sought Riemannian gradient can be constructed as

$$P_{\boldsymbol{X}} \frac{\partial f}{\partial \boldsymbol{X}} = \begin{bmatrix} \boldsymbol{U} & \mathbf{U}_\delta \end{bmatrix} \begin{bmatrix} \mathbf{V}_\delta & \boldsymbol{V} \end{bmatrix}^\intercal,$$

where

$$\begin{aligned} \mathbf{U}_\delta &= \left. \frac{\partial g}{\partial \boldsymbol{A}} \right|_{\substack{\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}, \\ \boldsymbol{B} = \mathbf{O}_{n \times r}}} \\ \mathbf{V}_\delta^\intercal &= \left. \frac{\partial g}{\partial \boldsymbol{B}^\intercal} \right|_{\substack{\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}, \\ \boldsymbol{B} = \mathbf{O}_{n \times r}}} \cdot (\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^\intercal) \end{aligned} \tag{10}$$

For the general case of tensors of fixed TT-rank, a method for automatic Riemannian differentiation is proposed and the following estimate of the complexity of the proposed method is formulated in Section D3.4.2.

***Statement* 1.** *Let a function $f : \mathbb{R}^{n_1 \times \ldots \times n_d} \to \mathbb{R}$ be given in the form of a program p, which takes as input the TT-cores of a tensor $\boldsymbol{\mathcal{X}}$ and calculates the value $f(\boldsymbol{\mathcal{X}})$ with asymptotic complexity $\mathcal{O}(F)$, which polynomially depends on the TT-ranks of the tensor $\boldsymbol{\mathcal{X}}$ (in other words, the program p belongs to the complexity class P). Then the complexity of computing the TT cores of the Riemannian gradient $P_{\boldsymbol{\mathcal{X}}} \frac{\partial f}{\partial \boldsymbol{\mathcal{X}}}$ using the algorithm D5 is equal to $\mathcal{O}(F + dn\,\mathrm{r}^3)$, where $n = \max_{k=1,\ldots,d} n_k$, $\mathrm{r} = \max_{k=1,\ldots,d-1} r_k$.*

In practice, for most functions, the complexity of calculating the value of the function dominates the additional complexity $\mathcal{O}(dn\,\mathrm{r}^3)$ from Statement 1. Lemma D6 formulates the class of functions for which this is true. Examples of such functions are:

- Frobenius norm of a tensor $f(\boldsymbol{\mathcal{X}}) = \|\boldsymbol{\mathcal{X}}\|_F$;
- Scalar product of two tensors $f(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{A}}) = \langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{A}} \rangle$ for $\mathcal{O}(\mathrm{r}(\boldsymbol{\mathcal{X}})) \leq \mathcal{O}(\mathrm{r}(\boldsymbol{\mathcal{A}}))$;
- The function arising when solving systems of linear equations $f(\boldsymbol{x}) = \boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^\intercal \boldsymbol{f}$ for $\mathcal{O}(\mathrm{r}(\boldsymbol{x})) \leq \mathcal{O}(\mathrm{r}(\boldsymbol{A}))$;
- Function arising when solving eigenvalue problems $f(\boldsymbol{x}) = \frac{\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\intercal \boldsymbol{x}}$;
- A function that arises when solving the *tensor completion problem*, that is recovering a low TT-rank tensor given only a fraction of the tensor values $f(\boldsymbol{\mathcal{X}}) = \|P_\Omega(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}})\|_F^2$, where the operator P_Ω vanishes the tensor values in elements where the true value is not known.

Thus, we can formulate the following corollary

***Corollary* 1.** *Let a function $f : \mathbb{R}^{n_1 \times \ldots \times n_d} \to \mathbb{R}$ be given in the form of the program p, which takes as input the TT-cores of a tensor $\boldsymbol{\mathcal{X}}$ and calculates the value $f(\boldsymbol{\mathcal{X}})$ with asymptotic complexity $\mathcal{O}(F)$, which polynomially depends on the TT-ranks of the tensor $\boldsymbol{\mathcal{X}}$. Suppose also that the function f satisfies the conditions of Lemma D6. Then, the complexity of computing the TT cores of the Riemannian gradient $P_{\boldsymbol{\mathcal{X}}} \frac{\partial f}{\partial \boldsymbol{\mathcal{X}}}$ using the algorithm D5 is equal to $\mathcal{O}(F)$, that is, is equal to the complexity of calculating the value of the function at a single point.*

Note that even in cases where the conditions of the corollary D2 are not satisfied and the asymptotic complexity of calculating the value of the function at a single point is less than $\mathcal{O}(dn\,\mathrm{r}^3)$, the algorithm D5 still works with the optimal asymptotic complexity, since by the definition of an element of the tangent space D(1.8), its constructive representation requires the computation of left- and right-orthogonal TT-cores, the complexity of which already equals $\mathcal{O}(dn\,\mathrm{r}^3)$.

In Section D3.4.3, a new method is proposed for automatically calculating the product of a linearized Riemannian Hessian by a given vector from the tangent space, that is

$$P_{\boldsymbol{\mathcal{X}}} \left(\nabla^2 f(\boldsymbol{\mathcal{X}}) \right) P_{\boldsymbol{\mathcal{X}}} \boldsymbol{\mathcal{Z}} \tag{11}$$

The main idea of the proposed method is as follows. Using the first order Riemannian automatic differentiation proposed in Section D3.4.2 we can calculate the Riemannian gradient $P_{c(\boldsymbol{\mathcal{X}})} \frac{\partial f}{\partial \boldsymbol{\mathcal{X}}}$, where $c(\boldsymbol{\mathcal{X}})$ is a constant function equal to $\boldsymbol{\mathcal{X}}$ at any point (ie, the gradient of this function is equal to zero). This function is also called

«stop gradient» in the automatic differentiation community. Multiplying the Riemannian gradient by a given vector, we obtain

$$w(\boldsymbol{\mathcal{X}}) = \left\langle P_{c(\boldsymbol{\mathcal{X}})} \frac{\partial f}{\partial \boldsymbol{\mathcal{X}}}, P_{c(\boldsymbol{\mathcal{X}})} \boldsymbol{\mathcal{Z}} \right\rangle \tag{12}$$

We again apply the Riemannian automatic differentiation to the function (12) and obtain the desired product of the linearized Riemannian Hessian by the vector $H\boldsymbol{\mathcal{Z}} = P_{\boldsymbol{\mathcal{X}}} \frac{\partial}{\partial \boldsymbol{\mathcal{X}}} w(\boldsymbol{\mathcal{X}})$.

Similarly to the case of automatic first-order differentiation, it is possible to formulate the following estimate of the complexity of the proposed algorithm

***Statement* 2.** *Let a function $f : \mathbb{R}^{n_1 \times \ldots \times n_d} \to \mathbb{R}$ be given in the form of a program p, which takes as input the TT-cores of the tensor $\boldsymbol{\mathcal{X}}$ and calculates the value $f(\boldsymbol{\mathcal{X}})$ with asymptotic complexity $\mathcal{O}(F)$, which polynomially depends on the TT-ranks of the tensor $\boldsymbol{\mathcal{X}}$ (otherwise in words, the given program belongs to the complexity class P). Then the complexity of computing the TT kernels of the product of the linearized Riemannian Hessian and the vector $\boldsymbol{\mathcal{Z}}$ given in the form of delta terms $\{\delta \boldsymbol{S}_k^{\boldsymbol{\mathcal{Z}}}\}_{k=1}^d$ is equal to $\mathcal{O}(F + dn\,\mathrm{r}^3)$, where $n = \max_{k=1,\ldots,d} n_k$, $\mathrm{r} = \max_{k=1,\ldots,d-1} r_k$.*

And, similarly to the case of automatic first-order differentiation, we can formulate the following corollary

***Следствие* 1.** *Let the function $f : \mathbb{R}^{n_1 \times \ldots \times n_d} \to \mathbb{R}$ be given in the form of the program p, which takes as input the TT-cores of the tensor $\boldsymbol{\mathcal{X}}$ and calculates the value $f(\boldsymbol{\mathcal{X}})$ with asymptotic complexity $\mathcal{O}(F)$, which polynomially depends on the TT-ranks of the tensor $\boldsymbol{\mathcal{X}}$. Suppose also that the function f satisfies the conditions of Lemma D6. Then the complexity of computing the TT-cores of the product of the linearized Riemannian Hessian and a vector $\boldsymbol{\mathcal{Z}}$ given in the form of the delta terms $\{\delta \boldsymbol{S}_k^{\boldsymbol{\mathcal{Z}}}\}_{k=1}^d$ equals to $\mathcal{O}(F)$, that is, equals to the complexity of calculating the value of the function at a single point.*

Note that, as in the case of calculating the Riemannian gradient, even in those cases when the conditions of Corollary D3 are not satisfied and the asymptotic complexity of calculating the value of the function at a single point is less than $\mathcal{O}(dn\,\mathrm{r}^3)$, the proposed algorithm still works with the optimal asymptotic complexity.

In **the fourth chapter** a library for working with TT-decomposition which was developed in this thesis is described.

The main differences compared to the already existing implementations of the TT decomposition [3] [4] are:

1. Support for running on graphics processing units (GPUs) for all operations. The package automatically checks if a GPU device is available on a given machine and, if available, transfers the calculations to a graphics accelerator.

[3] `https://pypi.org/project/TensorToolbox/`

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

2. Extended support for Riemannian optimization. The only other library for working with the TT-decomposition that supports Riemannian optimization is TTPY, which has only a single Riemannian operation supported – projection onto the tangent space (which was implemented in the TTPY package by the author of this dissertation). In T3F package (implemented as a part of this dissertation work), many special cases are available that are required for the implementation of numerical methods of Riemannian optimization, which can be formally implemented using a combination of the projection operator and basic operations (such as matrix-vector multiplication), but can be implemented significantly faster when considering a particular special case. Examples of such operations are the projection of the matrix-vector product onto the tangent plane of the TT-vector $P_{\boldsymbol{b}}(\boldsymbol{A}\boldsymbol{c})$ and the calculation of the Gram matrix of the projections onto the tangent plane $G_{ij} = \langle P_{\boldsymbol{b}}\boldsymbol{x}_i, P_{\boldsymbol{b}}x_j\rangle$.
3. Support for working with multiple (a batch of) TT-objects simultaneously, implemented using vectorized operations. For example, the calculation of the Gram matrix of a set of TT-objects can be performed in one parallel operation.
4. Support for automatic differentiation of an arbitrary (differentiable) function $L(\cdot)$ which is a function of a tensor in TT-format. T3F allows you to automatically calculate the gradient of a function with respect to the TT-cores, the Riemannian gradient (the projection of the euclidian gradient onto the tangent space at a given point), and the product of the linearized Riemannian Hessian by a vector.
5. Documentation and tests (test coverage is 93% with a total code size of 11000 lines).

Section D4.1.3 provides the speed comparison between the developed T3F library and the existing TTPY library. From the results of numerical speed measurements, it is clear that T3F is faster than TTPY for most operations. The difference is especially visible when using vectorization (working with multiple objects at the same time) and using GPUs.

Section D4.1.4 describes the details of the implementation of new operations, effective formulas for which were derived as a part of this dissertation. These operations are: projection of matrix-by-vector multiplication, bilinear form and matrix of pairwise bilinear forms.

Section D4.1.5 presents the results of numerical experiments comparing the operating speed and memory consumption of the proposed Riemannian automatic differentiation method in comparison with the implementation of calculating the Riemannian gradient and the matrix-vector product of a given vector by a linearized Riemannian Hessian manually. Numerical experiments are carried out on the following functions

- $f(\boldsymbol{x}) = \langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x}\rangle$, where $\boldsymbol{A}$ is a TT matrix (matrix in matrix TT format) and $\boldsymbol{x}$ is a TT vector. This function arises when solving systems of linear equations.
- $f(\boldsymbol{x}) = \langle \boldsymbol{B}\boldsymbol{A}\boldsymbol{x}, \boldsymbol{x}\rangle$. This function occurs when solving systems of linear equations and using a preconditioner.

– the Rayleigh relation $f(\boldsymbol{x}) = \dfrac{\langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x} \rangle}{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$ arising when solving eigenvalue problems;
– $f(\boldsymbol{\mathcal{X}}) = \|\boldsymbol{P}_\Omega(\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{A}})\|^2$ where $\boldsymbol{P}_\Omega$ denotes the operator of projection onto the set of indices Ω, namely

$$\boldsymbol{P}_\Omega \boldsymbol{\mathcal{X}} = \begin{cases} \mathcal{X}_{i_1,\ldots,i_d} & (i_1,\ldots,i_d) \in \Omega, \\ 0 & \text{otherwise.} \end{cases}$$

This function arises when solving low TT-rank tensor compleation problems.
– $f(\boldsymbol{\mathcal{X}})$ specifying the empirical risk of the tensor model (8).

For each of the functions described above, a naive version of the algorithm for calculating the Riemannian gradient was implemented (i.e., calculating the Euclidean gradient in TT-format and then the projecting it onto the tangent space), as well as an optimized version that performs these two steps together to speed up the calculations. Comparison of the speed and memory consumption of two approaches to calculating the Riemannian gradient in comparison with the automatic method for calculating this value proposed in section D3.4 is given in table D5a. Similarly, three methods for calculating the product of the linearized Riemannian Hessian of a given function and a vector are compared in the table D5b. These tables show that the proposed automatic Riemannian method in many situations allows you to calculate the required values faster and using less memory than the optimized implementation, while avoiding the difficulties of manual derivation of formulas and implementation of methods.

Section D4.1.6 provides a brief description of other research projects produced with the participation of the author of this dissertation, in which the T3F library was used.

Namely, a project on developing a scalable Gaussian processes based on representing a certain tensor in the TT-format [14]; a project on finding eigenvalues of a multidimensional Hermitian linear operators [2]; and a project on evaluating the expressive power of recurrent neural networks [15].

Section D4.2 presents the results of numerical experiments comparing the accuracy and speed of the method for calculating the MRF partition function proposed in Section D2.3.2 with analogs.

The main model for experiments is the *Ising model*. The energy function of the Ising model is defined as follows:

$$\mathbf{E}(\boldsymbol{x}) = -\frac{1}{T}\left(\sum_{k=1}^{d} x_k h_k + \sum_{(i,j)\in\mathcal{E}} c_{ij} x_i x_j\right), \tag{13}$$

where the variables x_k, $k = 1,\ldots,d$ take values from the set $\{-1, 1\}$. The coefficients h_k will be called *unary weights*, the coefficients c_{ij} — *pairwise weights*, and T — *temperature*. Pairwise connections depend on the selected set of edges $\mathcal{E}$. Most experiments use a 4-connected lattice of size 10×10. We call a model *homogeneous* if all paired weights are equal to each other ($c_{ij} = c$), and *heterogeneous* otherwise.

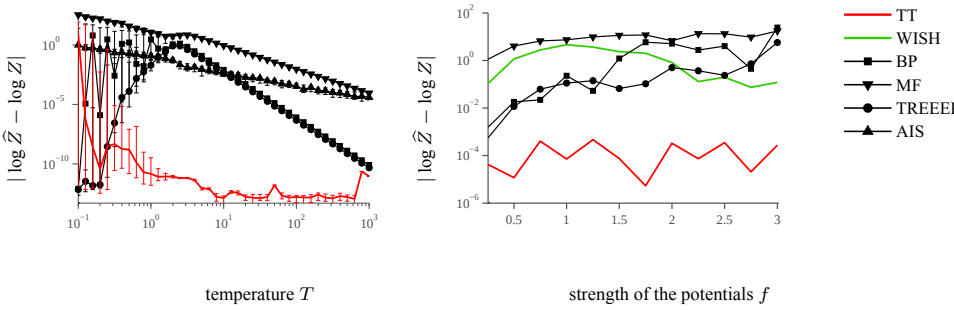

Fig. 1 — These figures show the results of experiments on calculating the partition function Z. In (a), the proposed method (TT) is compared with other methods on homogeneous Ising models of different temperatures. For each temperature value, 50 models are generated and the median and upper and lower quartiles are indicated. In (b), the proposed method is compared with the WISH method using heterogeneous Ising models. Both graphs show the errors of Z estimation (the lower the value - the better).

In the first experiment, the method based on TT decomposition (TT) is compared with the following methods from the LibDAI library: Belief Propagation (BP), Tree Expectation Propagation (TREEEP), and Mean Field (MF) method. A comparison is also made with the annealed importance sampling method (AIS) — a representative of MCMC methods. The results are shown in Fig. 1a. A set of Ising models with a 4-connected lattice of size 10×10 is considered. For each temperature value, 50 Ising models were generated, and the median and upper and lower quartiles of the absolute error in the estimate of the logarithm of the normalization constant are reported.

The second experiment compares the WISH method using data from the original paper [16]. The paper in which the WISH method was proposed considers a set of Ising models with a size lattice 10×10, unary weights h_i generated from a uniform distribution $U[-1, 1]$, a temperature T equal to 1, and paired weights c_{ij} generated from a uniform distribution $U[-f, f]$, where parameter f varies from 0.25 to 3. The comparison results are shown in Fig. 1b. Note that the WISH method was executed on a cluster and each computational core worked for at least 15 minutes. The proposed method was executed on a laptop and worked no more than 53 seconds in each example (32 seconds on average).

As can be seen from the experimental results, the proposed method significantly outperforms the alternatives in terms of accuracy with comparable or smaller running time.

Section D4.3 presents the results of numerical experiments on the compression of neural networks using the method proposed in Section D3.2. The basic properties

of neural networks are investigated when replacing a linear layer with a layer containing a matrix in TT-format and a comparison with analogs aimed at compressing neural networks is made. As shown in the numerical experiments, it is possible to compress the linear layer of the vgg-16 convolutional neural network trained for the ImageNet ILSVRC-2012 dataset (which consists of 1.2 million training images) more than 100000x with a quality loss of less than one percent, which significantly exceeds the results achieved by other approaches.

In section D4.4, an experimental comparison is made with alternatives to the proposed model, which allows one to take into account the interactions of high-order features (see section D3.3) and the proposed method of its training (see section D3.3.3). Using several different datasets, it is shown that the use of Riemannian optimization makes training more robust to initialization. It is also shown that the proposed model allows one to take into account the multiplicative interactions of high-order features, which cannot be achieved using most other methods (except for the recently proposed high-order factorization machines, which have singificantly higher training time in this regime).

Conclusion

This dissertation is devoted to the development of effective methods for solving machine learning problems based on the apparatus of tensor decompositions. The main results of the work are the developed methods for working with Markov random fields and their theoretical analysis, the proposed method for compressing neural networks, the proposed model that takes into account polynomial interactions of inputs and allows the Riemannian learning algorithm, and the method for automatically calculating the Riemannian gradient with optimal speed. Additional result developed as a part of this dissertation is the T3F library, which simplifies further work at the intersection of machine learning and tensor decompositions and supports such capabilities as running methods on GPUs and automatically calculating Riemannian gradients.

- A method for estimating the partition function and marginal distributions of a Markov random fields which singificantly outperfromed alternatives in terms of the accuracy given fixed time budget;
- A theoretical analysis of the proposed method which allows to upper bound the error of the proposed method and to approximately choose the hyperparameters of the method to achieve a given accuracy;
- A neural network that uses tensors in TT-format for parametrizing fully connected layers. As shown in the numerical experiments, it allows in some cases to compress fully connected layers of convolutional neural networks (such as VGG-16) hundreds of thousands of times with minor loss of quality;
- A machine learning model is proposed that allows taking into account the interactions of high-order features and parametrized by a single tensor in the TT format. Due to its structure, the model allows training by Riemannian optimization methods, which (as shown in numerical experiments) leads to

greater robustness to the choice of hyperparameters, compared to alternative training methods;

- A method is developed for the automatic Riemannian differentiation of a function on the manifold of tensors of a fixed TT-rank. The developed method makes it possible to automatically find the Riemannian gradient and the product of the linearized Riemannian Hessian by a given vector with machine accuracy and with asymptotic complexity that equals the complexity of calculating the original function at a single point;
- A library for working with tensors in TT-format, which supports the proposed method of automatic Riemannian differentiation, executing the methods on GPUs and vectorized computations.

References

1. Tensor Train Decomposition on TensorFlow (T3F) / A. Novikov [et al.] // Journal of Machine Learning Research, Machine Learning Open Source Software (JMLR MLOSS). — 2020. — Vol. 21, no. 30. — P. 1–7.
2. *Rakhuba M.* Low-rank Riemannian eigensolver for high-dimensional Hamiltonians / M. Rakhuba, A. Novikov, I. Oseledets // Journal of Computational Physics. — 2019. — Vol. 396. — P. 718–737.
3. Tensorizing neural networks / A. Novikov [et al.] // Advances in Neural Information Processing Systems. — 2015. — P. 442–450.
4. Putting MRFs on a Tensor Train / A. Novikov [et al.] // International Conference on Machine Learning (ICML). — 2014. — P. 811–819.
5. *Novikov A.* Exponential machines / A. Novikov, M. Trofimov, I. Oseledets // Bulletin of the Polish Academy of Sciences: Technical Sciences. — 2018. — Vol. 6.
6. Тензорный поезд в марковском случайном поле / А. Новиков [et al.] //. — 2014. — P. 293–318.
7. Language models are few-shot learners / T. B. Brown [et al.] // Advances in Neural Information Processing Systems 33 (NeurIPS). — 2020.
8. The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation / M. X. Chen [et al.] // arXiv preprint arXiv:1804.09849. — 2018.
9. *Oseledets I. V.* Tensor-Train Decomposition / I. V. Oseledets // SIAM J. Scientific Computing. — 2011. — Vol. 33, no. 5. — P. 2295–2317.
10. *Holtz S.* On manifolds of tensors of fixed TT-rank / S. Holtz, T. Rohwedder, R. Schneider // Numerische Mathematik. — 2012. — P. 701–731.
11. Automatic differentiation in machine learning: a survey / A. G. Baydin [et al.]. — 2018.
12. Learning representations by back-propagating errors / D. E. Rumelhart, G. E. Hinton, R. J. Williams, [et al.] // Cognitive modeling. — 1988. — Vol. 5, no. 3. — P. 1.
13. *Boykov Y.* Markov random fields with efficient approximations / Y. Boykov, O. Veksler, R. Zabih // Computer vision and pattern recognition, 1998. Proceedings. 1998 IEEE computer society conference on. — IEEE. 1998. — P. 648–655.
14. *Izmailov P.* Scalable Gaussian Processes with Billions of Inducing Inputs via Tensor Train Decomposition / P. Izmailov, A. Novikov, D. Kropotov // The 21st International Conference on Artificial Intelligence and Statistics (AISTATS 2018). — 2018.

15. *Khrulkov V.* Expressive power of recurrent neural networks / V. Khrulkov, A. Novikov, I. Oseledets // 6th International Conference on Learning Representations, ICLR. — 2018.

16. Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization / S. Ermon [et al.] // International Conference on Machine Learning (ICML). — 2013.