Tensor networks and deep learning

I. Oseledets

Skoltech, INM RAS, Moscow

3 November 2017

Skoltech
Сколковский институт науки и технологий

Deep, Deep Trouble

Michael Elad, SIAM News, 01.05.2017

I am really confused. I keep changing my opinion on a daily basis, and I cannot seem to settle on one solid view of this puzzle. I am talking about ... deep learning.

For the sake of brevity, consider the classic image processing task of denoising — removing noise from an image. Researchers developed beautiful and deep mathematical ideas with tools .... In 2012, Harold Burger, Christian Schuler, and Stefan Harmeling decided to throw deep learning into this problem. The end result was a network that performed better than any known image denoising algorithm at that time.

Deep learning

- The general paradigm is to parametrize your favourite algorithm as a (deep) neural network
- Prepare a training set of true answers
- The problem is non-convex and it is difficult to lie mathematical foundations, i.e. “guarantees” in any form

Approximation of multivariate functions

Given a training set (x_i, y_i), $i = 1, \dots, N$, $x_i \in \mathbb{R}^d$

we want to build a model $f(x, w)$

Such that

$$f(x_i, w) = \hat{y}_i \approx y_i$$

in the sense of a certain loss function

$$\sum_{i=1}^{N} \mathcal{L}(\hat{y}_i, y_i) \to \min .$$

Classification and regression are multivariate function interpolation

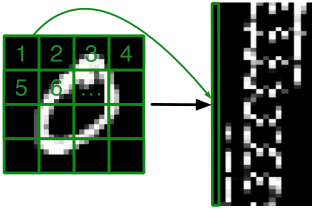
Skoltech
Сколковский институт науки и технологий

Classification

Suppose that we have a classification problem and a dataset of pairs $\{(X^{(b)}, y^{(b)})\}_{b=1}^{N}$. Let us assume that each object $X^{(b)}$ is represented as a sequence of vectors

$$X^{(b)} = (\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_d), \quad \mathbf{x}_k \in \mathbb{R}^n,$$

Example



Example

Consider $\{f_{\theta_\ell} : \mathbb{R}^n \to \mathbb{R}\}_{\ell=1}^m$, which are organized into a representation map

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^m.$$

A typical choice for such a map is

$$f_\theta(\mathbf{x}) = \sigma(A\mathbf{x} + b),$$

Connection between tensors and deep learning

Score functions considered in Cohen et. al, 2016 can be written in the form

$$l_y(X) = \langle \mathcal{W}_y, \Phi(X) \rangle,$$

where

$$\Phi(X)^{i_1 i_2 \ldots i_d} = f_{\theta_{i_1}}(\mathbf{x}_1) f_{\theta_{i_2}}(\mathbf{x}_2) \ldots f_{\theta_{i_d}}(\mathbf{x}_d),$$

Connection between tensors and deep learning

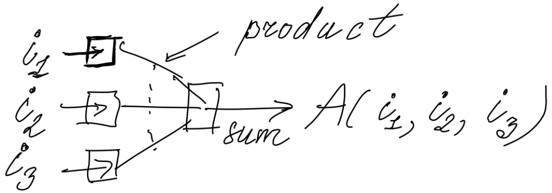
$$l_y(X) = \langle \mathcal{W}_y, \Phi(X) \rangle,$$

We put low-rank constraints on $\mathcal{W}$ and get different neural networks.

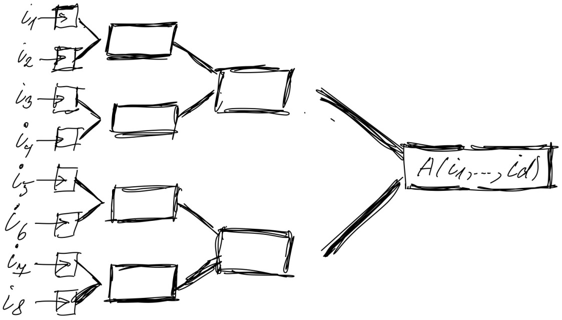
Canonical format and shallow network

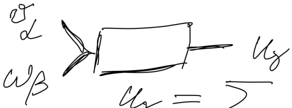
N. Cohen, A. Shashua et. al provided an interpretation of the canonical format as a shallow neural network with a product pooling

$$A(i_1, \dots, i_d) \approx \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha).$$



H-Tucker as a deep neural network with product pooling





$$u_\gamma = \sum_{\alpha\beta} M_{\alpha\beta\gamma} v_\alpha w_\beta$$

Tensor-train

TT-decomposition is defined as

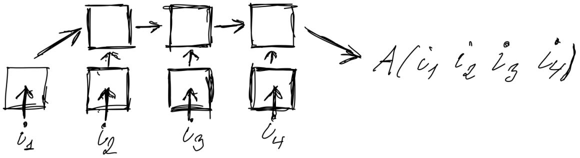
$$A(i_1, \dots, i_d) = G_1(i_1) \dots G_d(i_d),$$

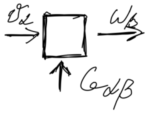
$G_k(i_k)$ is $r_{k-1} \times r_k$, $r_0 = r_d$.

Known for a long time as matrix product state in solid state physics.

Tensor-train as recurrent neural network

$$A(i_1, \dots, i_d) = G_1(i_1) \dots G_d(i_d),$$





$$w_\beta = \sum_\alpha v_\alpha G_{\alpha\beta}$$

Expressive power result

We prove that given a random d-dimensional tensor in the TT format with ranks **r** and modes n, with probability 1 this tensor will have exponentially large CP-rank.

Lemma

Let $\mathcal{X}^{i_1 i_2 \ldots i_d}$ and

$$\operatorname{rank}_{CP} \mathcal{X} = r.$$

Then for any matricization $\mathcal{X}^{(s,t)}$

$$\operatorname{rank} \mathcal{X}^{(s,t)} \leq r,$$

where ordinary matrix rank is assumed.

Theorem

Suppose that $d = 2k$ is even. Define the following set

$$B = \{\mathcal{X} \in \mathcal{M}_{\mathbf{r}} : \operatorname{rank}_{CP} \mathcal{X} < q^{\frac{d}{2}}\},$$

where $q = \min\{n, r\}$.

Then

$$\mu(B) = 0,$$

where μ is the standard Lebesgue measure on $\mathcal{M}_{\mathbf{r}}$.

Idea of the proof

We would like to show that that for $s = \{1, 3, \dots d-1\}$, $t = \{2, 4, \dots d\}$ the following set

$$B^{(s,t)} = \{\mathcal{X} \in \mathcal{M}_{\mathbf{r}} : \operatorname{rank} \mathcal{X}^{(s,t)} \leq q^{\frac{d}{2}} - 1\},$$

has measure 0.

We have

$$B \subset B^{(s,t)},$$

so if $\mu(B^{(s,t)}) = 0$ then $\mu(B) = 0$ as well.

Idea of the proof

- To show that $\mu(B^{(s,t)}) = 0$ it is sufficient to find at least one $\mathcal{X}$ such that *rank* $\mathcal{X}^{(s,t)} \geq q^{\frac{d}{2}}$.
- This follows from the fact that $B^{(s,t)}$ is an algebraic subset of the irreducible algebraic variety $\mathcal{M}_{\mathbf{r}}$, so it is either equal to $\mathcal{M}_{\mathbf{r}}$ or has measure 0.

Example of a tensor

$$
\begin{aligned}
&G_1^{i_1 \alpha_1} = \delta_{i_1 \alpha_1}, \quad G_1 \in \mathbb{R}^{1 \times n \times r} \\
&G_k^{\alpha_{k-1} i_k \alpha_k} = \delta_{i_k \alpha_{k-1}}, \quad G_k \in \mathbb{R}^{r \times n \times 1}, k = 2, 4, 6, \dots, d-2 \\
&G_k^{\alpha_{k-1} i_k \alpha_k} = \delta_{i_k \alpha_k}, \quad G_k \in \mathbb{R}^{1 \times n \times r}, k = 3, 5, 7, \dots, d-1 \\
&G_d^{\alpha_{d-1} i_d} = \delta_{i_d \alpha_{d-1}}, \quad G_d \in \mathbb{R}^{r \times n \times 1}
\end{aligned}
\tag{1}
$$

Connection

Table: Correspondence between languages of Tensor Analysis and Deep Learning.

Tensor Decompositions	**Deep Learning**
CP-decomposition	shallow network
TT-decomposition	RNN
HT-decomposition	CNN
rank of the decomposition	width of the network

Expressive power

Table: Comparison of the expressive power of various networks. Given a network, specified in a column, rows correspond to the upper bound on the width of the equivalent network of other type.

	TT-Network (r)	**HT-Network** (r)	**CP-Network** $(r$
TT-Network	r	$r^{log_2 d}$	r
HT-Network	r^2	r	r
CP-Network	$\geq r^{\frac{d}{2}}$	$\geq r^{\frac{d}{2}}$	r

Why tensor networks are good

Low-rank tensor decompositions (and problems with them) can be solved using efficient linear algebra tools.

Simplest tensor network

The simplest tensor network is matrix factorization:

$$A = UV^\top.$$

Why matrix factorization is great

$$A \approx UV^\top$$

- Best factorization by SVD
- Riemannian manifold structure
- Nice convex relaxation (nuclear norm)
- Cross approximation / skeleton decomposition

Cross approximation / skeleton decomposition

One of underestimated matrix facts:

If a matrix is rank r, it can be represented as

$$A = C\widehat{A}^{-1}R,$$

where C are some r columns of A, R are some rows of A, $\widehat{A}$ is a submatrix on the intersection.

Maximum-volume principle

Goreinov, Tyrtyshnikov, 2001 have shown:

If $\widehat{A}$ has maximal volume, then

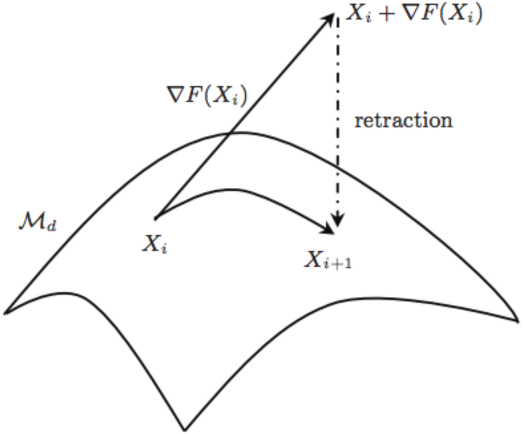
$$\|A - A_{skel}\|_C \leq (r+1)\sigma_{r+1}.$$

Way to compare submatrices!

Riemannian framework

Low-rank matrices form a manifold

Standard: $F(X) = F(UV^\top) \to min$



Riemannian:

Riemannian word embedding

Example: Riemannian Optimization for Skip-Gram Negative Sampling A Fonarev, O Hrinchuk, G Gusev, P Serdyukov arXiv:1704.08059, ACL 2017.

We treated SGNS as implicit matrix factorization and solved in using Riemannian optimization.

Negative sampling

Words and context (nearby words)

$$X = WC^T = (x_{wc}),\ x_{wc} = \langle \mathrm{w}, \mathrm{c} \rangle$$

$$\mathcal{M}_d = \{X \in \mathbb{R}^{n \times m} : \mathsf{rank}(X) = d\}$$

$$F(X) = \sum_{w \in V_W} \sum_{c \in V_C} (\#(w,c)(\mathit{log}\, \sigma(x_{wc}) +$$

$$+ k \frac{\#(w)\#(c)}{|D|} \mathit{log}\, \sigma(-x_{wc}))) \to \mathit{max}_{X \in \mathcal{M}_d}$$

Results

usa					
SGD-SGNS		SVD-SPPMI		RO-SGNS	
Neighbors	Dist.	Neighbors	Dist.	Neighbors	Dist.
akron	0.536	**wisconsin**	0.700	**georgia**	0.707
midwest	0.535	**delaware**	0.693	**delaware**	0.706
burbank	0.534	**ohio**	0.691	**maryland**	0.705
nevada	0.534	northeast	0.690	**illinois**	0.704
arizona	0.533	cities	0.688	madison	0.703
uk	0.532	southwest	0.684	**arkansas**	0.699
youngstown	0.532	places	0.684	**dakota**	0.690
utah	0.530	counties	0.681	**tennessee**	0.689
milwaukee	0.530	**maryland**	0.680	northeast	0.687
headquartered	0.527	**dakota**	0.674	**nebraska**	0.686

Table: Examples of the semantic neighbors for "usa".

Tensor factorization

Tensor factorization: we want numerical tools of the same quality

Classical attempt

Matrix case:

$$A(i,j) = \sum_{\alpha=1}^{r} U(i,\alpha)V(j,\alpha).$$

CP-decomposition:

$$A(i,j,k) = \sum_{\alpha=1}^{r} U(i,\alpha)V(j,\alpha)W(k,\alpha)$$

Tucker decomposition:

$$A(i,j,k) = \sum_{\alpha,\beta,\gamma=1}^{r} G(\alpha,\beta,\gamma)U(i,\alpha)V(j,\beta)W(k,\gamma)$$

CP-decomposition has bad properties!

- Best rank-r approximation may not exist
- Algorithms may converge very slowly (swamp behaviour)
- No finite-step completion procedure.

Example where CP decomposition is not known

Consider a $9 \times 9 \times 9$ tensor A with slices

$$A_i = E_i \otimes I_3, \quad i = 1, \dots, 9,$$

and E_3 has only one identity element.

It is known that CP-rank of A is ≤ 23 and ≥ 20.

Example where CP decomposition does not exist

Consider

$$T = a \otimes b \otimes \ldots \otimes b + \ldots + b \otimes \ldots \otimes a.$$

Then,

$$P(t) = \otimes_{k=1}^{d} (b + ta), \quad P'(0) = T = \frac{P(h) - P(0)}{h} + \mathcal{O}(h).$$

Can be approximated with rank-2 with any accuracy, but no exact decomposition of rank less than d exist!

Our idea

Our idea was to build tensor decompositions using well-established matrix tools.

Reshaping tensor into matrix

Let reshape an $n \times n \times \ldots \times n$ tensor into a $n^{d/2} \times n^{d/2}$ matrix A:

$$\mathbb{A}(\mathcal{I}, \mathcal{J}) = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$$

and compute low-rank factorization of $\mathbb{A}$:

$$\mathbb{A}(\mathcal{I}, \mathcal{J}) \approx \sum_{\alpha=1}^{r} U(\mathcal{I}, \alpha) V(\mathcal{J}, \alpha).$$

Recursion

If we do it recursively, we get $r^{log\, d}$ complexity

If we do it smart, we get dnr^3 complexity:

- Tree-Tucker format (Oseledets, Tyrtyshnikov, 2009)
- H-Tucker format (Hackbusch, Kuhn, Grasedyck, 2011)
- Simple but powerful version: Tensor-train format (Oseledets, 2009)

Properties of the TT-format

- TT-ranks are ranks of matrix unfoldings
- We can do basic linear algebra
- We can do rounding
- We can recover a low-rank tensor from $\mathcal{O}(dnr^2)$ elements
- Good for rank-constrained optimization
- There are classes of problems where $r_k \sim log^s \varepsilon^{-1}$
- We have MATLAB, Python and Tensorflow toolboxes!

TT-ranks are matrix ranks

Define unfoldings:

$A_k = A(i_1 \dots i_k; i_{k+1} \dots i_d)$, $n^k \times n^{d-k}$ matrix

TT-ranks are matrix ranks

Define unfoldings:
$A_k = A(i_1 \dots i_k; i_{k+1} \dots i_d)$, $n^k \times n^{d-k}$ matrix   Theorem: there exists a TT-decomposition with TT-ranks

$$r_k = \mathit{rank}\, A_k$$

TT-ranks are matrix ranks

The proof is constructive and gives the TT-SVD algorithm!

TT-ranks are matrix ranks

No exact ranks in practice – stability estimate!

TT-ranks are matrix ranks

Physical meaning of ranks of unfoldings is entanglement: we split the system into two halves, and if rank is 1, they are independent.

Approximation theorem

If $A_k = R_k + E_k$, $||E_k|| = \varepsilon_k$

$$||\mathrm{A} - \mathrm{TT}||_F \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}.$$

TT-SVD

Suppose, we want to approximate:

$$A(i_1, \dots, i_d) \approx G_1(i_1)G_2(i_2)G_3(i_3)G_4(i_4)$$

1. A_1 is an $n_1 \times (n_2n_3n_4)$ reshape of A.
2. $U_1, S_1, V_1 = \mathrm{SVD}(A_1)$, U_1 is $n_1 \times r_1$ — first core
3. $A_2 = S_1V_1^*$, A_2 is $r_1 \times (n_2n_3n_4)$.
   Reshape it into a $(r_1n_2) \times (n_3n_4)$ matrix
4. Compute its SVD:
   $U_2, S_2, V_2 = \mathrm{SVD}(A_2)$,
   U_2 is $(r_1n_2) \times r_2$ — second core, V_2 is $r_2 \times (n_3n_4)$
5. $A_3 = S_2V_2^*$,
6. Compute its SVD:
   $U_3S_3V_3 = \mathrm{SVD}(A_3)$, U_3 is $(r_2n_3) \times r_3$, V_3 is $r_3 \times n_4$

Fast and trivial linear algebra

Addition, Hadamard product, scalar product, convolution
All scale linear in d

Fast and trivial linear algebra

$$C(i_1, \dots, i_d) = A(i_1, \dots, i_d) B(i_1, \dots, i_d)$$

$$C_k(i_k) = A_k(i_k) \otimes B_k(i_k),$$

ranks are multiplied

Tensor rounding

A is in the TT-format with suboptimal ranks.
How to reapproximate?

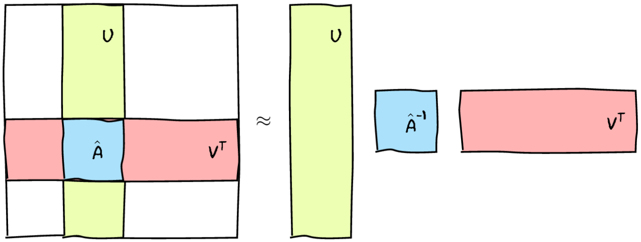
Tensor rounding

ε-rounding can be done in $\mathcal{O}(dnr^3)$ operations

Cross approximation

Recall the cross approximation

Rank-r matrix can be recovered from r columns and r rows



TT-cross approximation

Tensor with TT-ranks $r_k \leq r$ can be recovered from $\mathcal{O}(dnr^2)$ elements.

There are effective algorithms for computing those points in active learning fashion.

They are based on the computation of maximum-volume submatrices.

Making everything a tensor: the QTT

Let $f(x)$ be a univariate function (say, $f(x) = \mathit{sin}\, x$).

Let v be a vector of values on a uniform grid with 2^d points.

Transform v into a $2 \times 2 \times \ldots \times 2$ d-dimensional tensor.

Compute TT-decomposition of it!

And this is the QTT-format

Making everything a tensor: the QTT

If $f(x)$ is such that

$$f(x+y) = \sum_{\alpha=1}^{r} u_\alpha(x) v_\alpha(y),$$

then QTT-ranks are bounded by r

Corollary:

- $f(x) = \mathit{exp}(\lambda x)$
- $f(x) = \mathit{sin}(\alpha x + \beta)$
- $f(x)$ is a polynomial
- $f(x)$ is a rational function

Optimization with low-rank constraints

Tensors can be given implicitly as a solution of a certain optimization

$$F(X) \to \mathit{min}, \quad r_k \leq r.$$

The set of low-rank tensors is non-convex, but has efficient *Riemannian structure* and many fabulous unstudied geometrical properties.

Software

- http://github.com/oseledets/TT-Toolbox – MATLAB
- http://github.com/oseledets/ttpy – Python
- https://github.com/Bihaqo/t3f – Tensor Train in Tensorflow (Alexander Novikov)

Application of tensors

- High-dimensional, smooth functions
- Computational chemistry (electronic and molecular computations, spin systems)
- Parametric PDEs, high-dimensional uncertainty quantification
- Scale-separated multiscale problems
- Recommender systems
- Compression of convolutional layers in deep neural networks
- TensorNet (Novikov et. al) – very compact dense layers

Type of problems we can solve

- Active tensor learning by the cross method
- Solution of high-dimensional linear systems: $A(X) = F$
- Solution of high-dimensional eigenvalue problems $A(X) = \lambda X$
- Solution of high-dimensional time-dependent problems $\frac{dA}{dt} = F(A)$ (very efficient integrator).

Examples

Some examples of using tensor methods from different areas.

Vibrational states

Realistic Hamiltonian of CH_3CN
[P. Thomas, T. Carrington, 2015]

$$\mathcal{A} = \frac{1}{2}\sum_{i=1}^{12}\omega_i\left(-\frac{\partial^2}{\partial q_i^2} + q_i^2\right) + \frac{1}{6}\sum_{i=1}^{12}\sum_{j=1}^{12}\sum_{k=1}^{12}\phi^{(3)}_{ijk}q_iq_jq_k$$
$$+\frac{1}{24}\sum_{i=1}^{12}\sum_{j=1}^{12}\sum_{k=1}^{12}\sum_{l=1}^{12}\phi^{(4)}_{ijkl}q_iq_jq_kq_l.$$

$$\text{TT-rank} = [1, 5, 9, 14, 21, 25, 26, 24, 18, 15, 8, 5, 1]$$

Skoltech
Сколковский институт науки и технологий

Block eigensolvers in the the TT-format

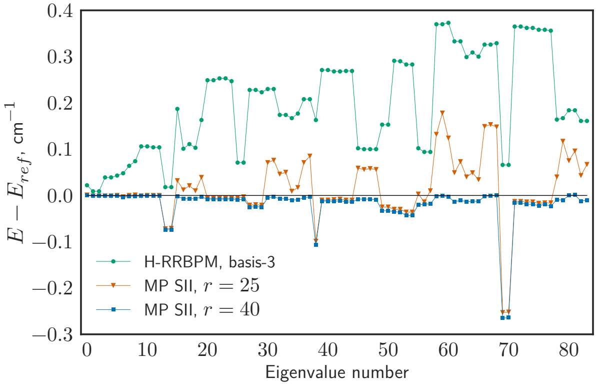


Figure: The E_{ref} are energies obtained by Smolyak quadratures [Carrington, 2011]. H-RRBPM method [Carrington, 2015].

M. Rakhuba and I. Oseledets. *Calculating vibrational spectra of molecules using tensor train decomposition* J. Chem. Phys. **145**, 124101 (2016)

Orbital-free DFT solver

Orbital-free DFT equation

Nonlinear eigenvalue problem:

$$\left(-\frac{1}{2}\Delta + C_F \phi^{4/3} - \sum_{\alpha} \frac{Z_\alpha}{|\mathrm{r} - \mathrm{R}_\alpha|} + \int_{\mathbb{R}^3} \frac{\phi^2(\mathrm{r}')}{|\mathrm{r} - \mathrm{r}'|}\, d\mathrm{r}' + V_{xc}(\phi^2)\right)\phi = \mu\phi$$

- For regular clusters of molecules density is of low QTT rank
- Standard iteration (SCF) in tensor formats is as difficult as initial problem $\Longrightarrow$ preconditioned gradient descent
- Derivative-free formulas to control accuracy

Some results about DL

- Generalization properties of DNN
- Adversarial examples

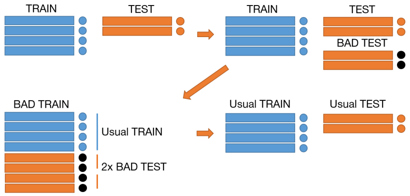
Generalization properties

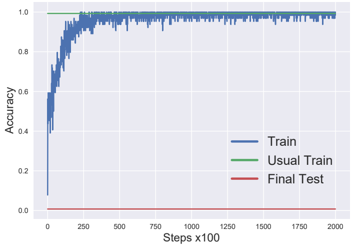
Work by Daniil Merkulov on experimental study of generalization properties of DNN.

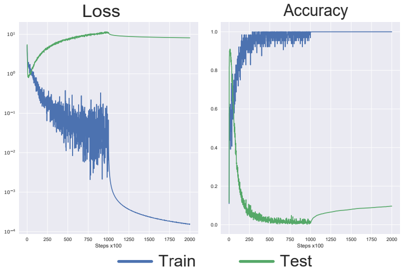
"Understanding Deep Learning requires rethinking generalization" - Chiyuan Zhang et al. 2016

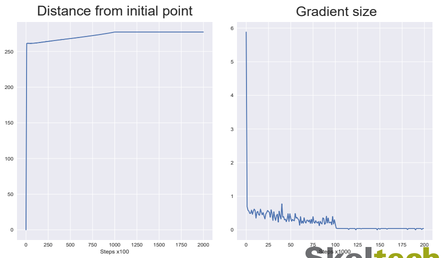
We have firstly investigated behaviour of our simple CNN in terms of robustness to shuffling of the labels. Here, we just corrupted part of labels by defining any random number between 0 and 9 randomly.

There are "sad" local minima on loss surfaces of NN



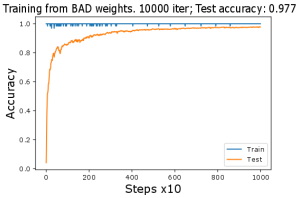


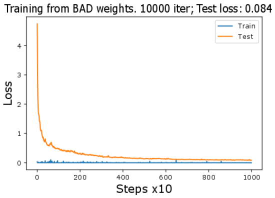




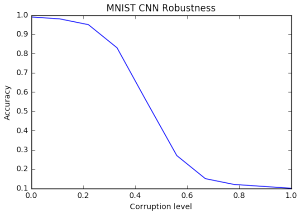
Properties of "sad" points (obtained so far)

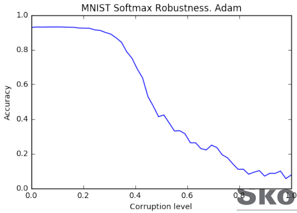
- Even l_2 regularization won't hide them if we try to find them relatively far from initial point.
- They are unstable (if we start stochastic algorithm with them, it will jump from such local minimum)





- Robustness of learning algorithms to the noise in labels have the same form for different learning models





Skoltech
Сколковский институт науки и технологий

Adversarial examples (joint with Valentin Khrulkov)

Suppose that we have a standard feed-forward DNN which takes a vector x as the input, and outputs a vector of probabilities $p(x)$ for class labels. Our goal is given parameters $q \geq 1$ and $L > 0$ produce a vector ε such that

$$\mathit{max}\, p(x) \neq \quad \mathit{max}\, p(x+\varepsilon), \quad \|\varepsilon\|_q = L,$$

for as many x in a dataset as possible. Efficiency of the given universal adversarial perturbation ε for the dataset X of the size N is called the *fooling rate* and is defined as

$$\frac{|\{x \in X : \quad \mathit{max}\, p(x) \neq \quad \mathit{max}\, p(x+\varepsilon)\}|}{N}.$$

Skoltech
Сколковский институт науки и технологий

Our construction

Let us denote the outputs of the i-th hidden layer of the network by $f_i(x)$. Then we have

$$f_i(x+\varepsilon) - f_i(x) \approx J_i(x)\varepsilon,$$

where

$$J_i(x) = \left.\frac{\partial f_i}{\partial x}\right|_x,$$

Thus, for any q

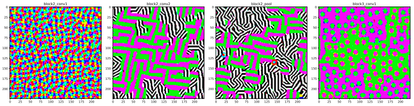
$$\|f_i(x+\varepsilon) - f_i(x)\|_q \approx \|J_i(x)\varepsilon\|_q, \tag{2}$$

Generalized singular values

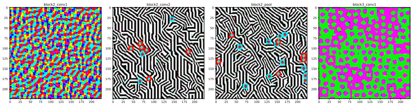
$$y = \quad \mathit{max} \frac{\|Ay\|_q}{\|y\|_p}.$$

We can use generalized power method to compute it.

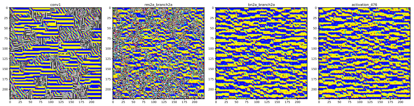
Singular vectors



(a) VGG-16



(b) VGG-19



(c) ResNet50

Figure: Universal adversarial perturbations constructed using various layers of various DNNs

Fooling rates

Layer name	block2_pool	block3_conv1	block3_conv2	block3_conv3
Singular value	1165.74	2200.08	3146.66	6282.64
Fooling rate	**0.52**	0.39	0.50	0.50

Table: Fooling rates for VGG-16

Layer name	block2_pool	block3_conv1	block3_conv2	block3_conv3
Singular value	784.82	1274.99	1600.77	3063.72
Fooling rate	**0.60**	0.33	0.50	0.52

Table: Fooling rates for VGG-19

Layer name	conv1	res3c_branch2a	bn5a_branch2c	activation_8
Singular value	59.69	19.21	138.81	15.55
Fooling rate	**0.44**	0.35	0.34	0.34

Table: Fooling rates for ResNet50