Tensorizing Neural Networks

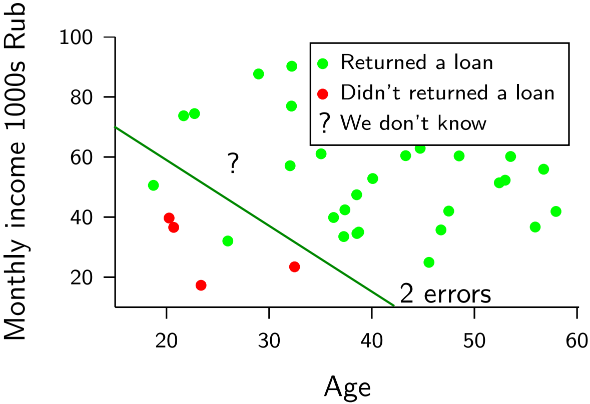
Alexander Novikov[1] Dmitry Podoprikhin[1] Anton Osokin[2]
Dmitry Vetrov[1,3]

[1]Skolkovo Institute of Science and Technology, Russia [2]SIERRA, INRIA, France
[3]Higher School of Economics, Russia

September 4, 2015

Machine learning

Lets consider a classification or regression problem.



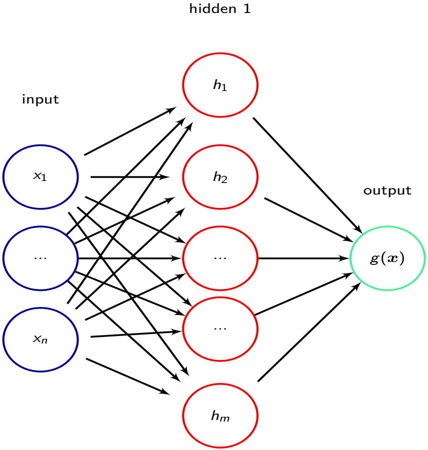
Neural networks

Lets use a composite function:

$$g(\boldsymbol{x}) = f(\boldsymbol{W}_2 \cdot \underbrace{f(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)}_{h \in \mathbb{R}^m} + \boldsymbol{b}_2)$$

$$h_k = f\left((\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)_k\right)$$

$$f(x) = \frac{1}{1 + \exp(-x)}$$



Neural networks learning

$$g(\boldsymbol{x}) = f(\boldsymbol{W}_2 \cdot f(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)$$

To teach the neural network minimize its' error on the training set:

$$\boldsymbol{W}_1^*, \boldsymbol{b}_1^*, \boldsymbol{W}_2^*, \boldsymbol{b}_2^* = \underset{\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2}{\arg\min} \sum_q (y_q - g(\boldsymbol{x}_q))^2$$

Motivation

We will compress fully-connected layers.

Why do we care about memory?

- State-of-the-art deep networks doesn't fit to mobile devices;
- Up to 95% percent of the parameters are in the fully connected layers;
- A shallow network with a huge fully connected layer can achieve almost the same accuracy, as an ensemble of deep CNNs (Ba and Caruana 2014).

Compact formats for a matrix

- Constant: $W(k, \ell) = a$.
    - Too restrictive
- Zip archive.
    - You need to uncompress the matrix before the multiplication, RAM requirement is not reduced
- Sparse matrix (most of the elements are zero).
    - How to choose which elements are zero?
    - A little bit hard to implement efficiently (especially on a GPU)

Matrix rank decomposition

Lets consider an $M \times N$ matrix $\boldsymbol{W}$ with the rank equals r. We can use $(M+N)r$ parameters instead of MN:

$$\underbrace{\boldsymbol{W}}_{M \times N} = \underbrace{\boldsymbol{A}}_{M \times \mathrm{r}} \underbrace{\boldsymbol{B}}_{\mathrm{r} \times N}$$

It works, but we want to do better.

Tensor Train summary

Tensor Train (TT) decomposition (Oseledets 2011):

- A compact representation for vectors, matrices and tensors;
- Allows for efficient application of linear algebra operations.

TT-format

Tensor $\boldsymbol{A}$ is said to be in the *TT-format*, if

$$\boldsymbol{A}(i_1, \ldots, i_d) = \boldsymbol{G}_1[i_1]\, G_2[i_2] \cdots \boldsymbol{G}_d[i_d], \quad i_k \in \{1, \ldots, n\},$$

where $\boldsymbol{G}_k[i_k]$ — is a matrix of size $\mathrm{r}_{k-1} \times \mathrm{r}_k$, $\mathrm{r}_0 = \mathrm{r}_d = 1$.
Notation & terminology:

- $\boldsymbol{G}_k$ — *TT-cores*;
- r_k — *TT-ranks*;
- $r = \max\limits_{k=0,\ldots,d} r_k$ — *the maximal TT-rank*.

The TT-format uses $O\left(ndr^2\right)$ memory to store n^d elements. Efficient only if the TT-rank is small.

TT-format: example

$$A(i_1, i_2, i_3) = i_1 + i_2 + i_3,$$

$$i_1 \in \{1, 2, 3\},\ i_2 \in \{1, 2, 3, 4\},\ i_3 \in \{1, 2, 3, 4, 5\}.$$

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

TT-format: example

$$\boldsymbol{A}(i_1, i_2, i_3) = i_1 + i_2 + i_3,$$

$$i_1 \in \{1, 2, 3\},\ i_2 \in \{1, 2, 3, 4\},\ i_3 \in \{1, 2, 3, 4, 5\}.$$

$$\boldsymbol{A}(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

$$G_1[i_1] = \begin{bmatrix} i_1 & 1 \end{bmatrix} \quad G_2[i_2] = \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix} \quad G_3[i_3] = \begin{bmatrix} 1 \\ i_3 \end{bmatrix}$$

Lets check:

$$A(i_1, i_2, i_3) = \begin{bmatrix} i_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} =$$

$$= \begin{bmatrix} i_1 + i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} = i_1 + i_2 + i_3.$$

TT-format: example

$$A(i_1, i_2, i_3) = i_1 + i_2 + i_3,$$
$$i_1 \in \{1, 2, 3\},\ i_2 \in \{1, 2, 3, 4\},\ i_3 \in \{1, 2, 3, 4, 5\}.$$

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

$$G_1 = \left(\begin{bmatrix} 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 \end{bmatrix} \right)$$
$$G_2 = \left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \right)$$
$$G_3 = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 5 \end{bmatrix} \right)$$

The tensor has $3 \cdot 4 \cdot 5 = 60$ elements.

The TT-format use 32 parameters to describe it.

Mapping example: a vector

How to store a vector $\boldsymbol{b}$ in the TT-format?
Build a mapping from a vector $\boldsymbol{b}$ indices to tensor's elements:
$t \leftrightarrow \boldsymbol{i} = (i_1, \ldots, i_d)$

Example (Matlab `reshape`):

$$\begin{aligned}
\boldsymbol{B}(1,1,1) &= \boldsymbol{b}(t(1,1,1)) &= \boldsymbol{b}(1), \\
\boldsymbol{B}(2,1,1) &= \boldsymbol{b}(t(2,1,1)) &= \boldsymbol{b}(2), \\
&\ldots, \\
\boldsymbol{B}(2,3,3) &= \boldsymbol{b}(t(2,3,3)) &= \boldsymbol{b}(18).
\end{aligned}$$

Now lets use the TT-format for the tensor $\boldsymbol{B}$.

Matrices in the TT-format

Build a mapping from row / column indices of matrix $\boldsymbol{W} = [\boldsymbol{W}(t, \ell)]$ to vectors $\boldsymbol{i}$ and $\boldsymbol{j}$: $t \leftrightarrow \boldsymbol{i} = (i_1, \dots, i_d)$ and $\ell \leftrightarrow \boldsymbol{j} = (j_1, \dots, j_d)$.

TT-format for matrix $\boldsymbol{W}$:

$$\boldsymbol{W}(i_1, \dots, i_d; j_1, \dots, j_d) = \boldsymbol{W}(t(\boldsymbol{i}), \ell(\boldsymbol{j})) = \underbrace{\boldsymbol{G}_1[i_1, j_1]}_{1 \times \mathrm{r}} \underbrace{\boldsymbol{G}_2[i_2, j_2]}_{\mathrm{r} \times \mathrm{r}} \dots \underbrace{\boldsymbol{G}_d[i_d, j_d]}_{\mathrm{r} \times 1}$$

The TT-format exists for any matrix $\boldsymbol{W}$ and uses $O(dmnr^2)$ memory to store $m^d n^d$ elements. Efficient only if the TT-rank is small.

Tensor Train layer

Input is a N-dimensional vector $\boldsymbol{x}$, output is a M-dimensional vector $\boldsymbol{h}$:

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}.$$

$\boldsymbol{W}$ is represented in the TT-format:

$$\boldsymbol{h}(i_1, \dots, i_d) = \sum_{j_1, \dots, j_d} \boldsymbol{G}_1[i_1, j_1] \dots \boldsymbol{G}_d[i_d, j_d]\, \boldsymbol{x}(j_1, \dots, j_d) + \boldsymbol{b}(i_1, \dots, i_d)$$

The parameters are the vector $\boldsymbol{b}$ and the TT-cores $\{\boldsymbol{G}_k\}_{k=1}^{d}$

Tensor Train layer learning

$$g(\boldsymbol{x}) = \sum_{j_1,\dots,j_d} \boldsymbol{G}_1[i_1, j_1] \dots \boldsymbol{G}_d[i_d, j_d]\, \boldsymbol{x}(j_1, \dots, j_d) + \boldsymbol{b}(i_1, \dots, i_d)$$

Instead of optimizing over all possible matrices $\boldsymbol{W}$:

$$\boldsymbol{W}^*, \boldsymbol{b}^* = \arg\min_{\boldsymbol{W}, \boldsymbol{b}} \sum_q (y_q - g(\boldsymbol{x}_q))^2,$$

we optimize over the matrices representable in the TT-format with rank r:

$$\boldsymbol{G}_1^*, \dots, \boldsymbol{G}_d^*, \boldsymbol{b}^* = \arg\min_{\boldsymbol{G}_1, \dots, \boldsymbol{G}_d, \boldsymbol{b}} \sum_q (y_q - g(\boldsymbol{x}_q))^2.$$

Tensor Train layer: the Jacobian

The Jacobian of the linear transformation:

$$\boldsymbol{h}(\boldsymbol{i}) = \sum_{\boldsymbol{j}} \boldsymbol{G}_1[i_1, j_1] \dots \boldsymbol{G}_k[i_k, j_k] \dots \boldsymbol{G}_d[i_d, j_d]\, \boldsymbol{x}(\boldsymbol{j}) + \boldsymbol{b}(\boldsymbol{i}).$$

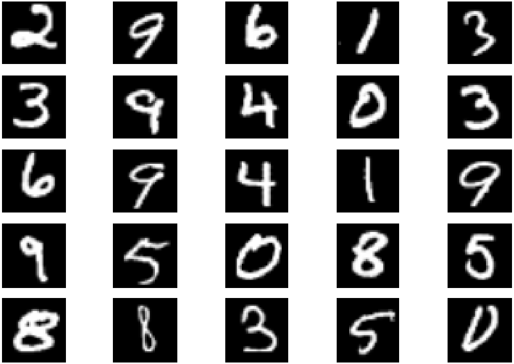
$$\frac{\partial \boldsymbol{h}(\boldsymbol{i})}{\partial \boldsymbol{G}_k[i_k, j_k]} = \sum_{\boldsymbol{j}^{\setminus k}} \overbrace{\boldsymbol{G}_1[i_1, j_1] \dots}^{\mathrm{r} \times 1} \cancel{\boldsymbol{G}_k[i_k, j_k]} \overbrace{\dots \boldsymbol{G}_d[i_d, j_d]}^{1 \times \mathrm{r}}\, \boldsymbol{x}(\boldsymbol{j}) =$$

$$\sum_{\boldsymbol{j}^{\setminus k,d}} \boldsymbol{G}_1[i_1, j_1] \dots \cancel{\boldsymbol{G}_k[i_k, j_k]} \dots \boldsymbol{G}_{d-1}[i_{d-1}, j_{d-1}] \underbrace{\sum_{j_d} \boldsymbol{G}_d[i_d, j_d]\, \boldsymbol{x}(\boldsymbol{j})}_{\mathrm{r} \times m n^{d-1}}$$

The complexity is $O(d^2\, \mathrm{r}^4\, m \max\{M, N\})$.

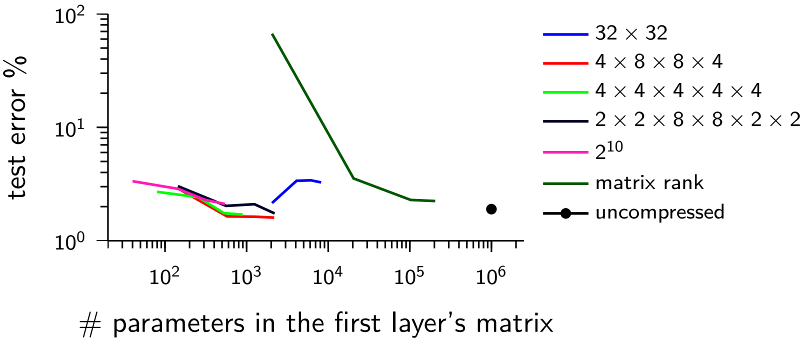
Mnist

Hadwritten digits recognition



Mnist cont'd

A two layered neural network with 1024×1024 and 1024×10 matrices.



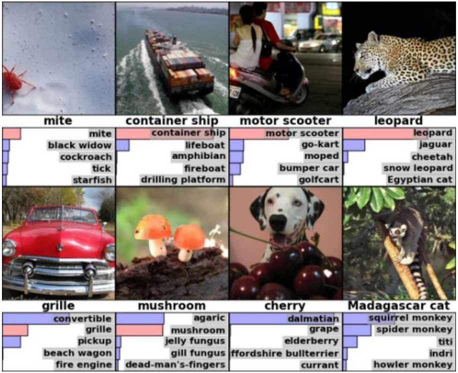
Speed

Type	1 im. time (ms)	100 im. time (ms)
CPU fully-connected layer	16.09	97.2
CPU TT-layer	1.24	94.7
GPU fully-connected layer	2.7	33
GPU TT-layer	1.92	12.86

$25\,088 \times 4\,096$ layer, the fully-connected layer uses 392MB and the TT-layer uses 0.766MB.

Image classification

Cifar & ImageNet



We used a $262\,144 \times 4\,096$ TT-matrix to outperform other non-convolutional neural networks on Cifar.

ImageNet

Architecture	Matrices compr.	Network compr.	Error
FC FC FC	1	1	11.2
TT4 FC FC	50 972	3.9	11.2
TT2 FC FC	194 622	3.9	11.5
TT1 FC FC	713 614	3.9	12.8
TT4 TT4 FC	37 732	7.4	12.3
LR1 FC FC	3 521	3.9	97.6
LR5 FC FC	704	3.9	53.9
LR50 FC FC	70	3.7	14.9

A 3-layered network, FC stands for the traditional layer; TT□ stands for the TT-layer with all the TT-ranks equal "□"; LR□ stands for the rank decomposition with the rank equal "□".

Future work

- Get rid of the convolutions.
    - Convolution matrix can be represented in the TT-format with small ranks
- Fully tensorial network.
    - Use the TT-format for all the intermediate calculations
    - Allows billions of hidden neurons
- Riemannian optimization.
    - Highly efficient for the optimization with the ranks constraints

References I

Ba, Jimmy and Rich Caruana (2014). "Do Deep Nets Really Need to be Deep?" In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2654–2662.

Oseledets, I. V. (2011). "Tensor-Train Decomposition". In: *SIAM J. Scientific Computing* 33.5, pp. 2295–2317.