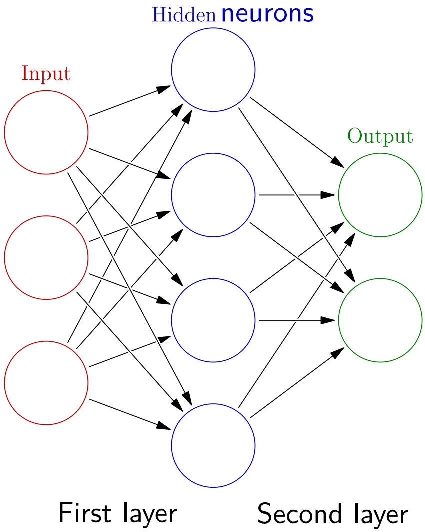
TensorNet: putting neural networks on a Tensor Train

A. Novikov D. Podoprikhin A. Osokin D. Vetrov

March 6, 2015

Outline

Neural networks



Motivation

Why do we care about memory?

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

Matrix rank decomposition

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

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

Drawbacks of the rank decomposition

Problems:

1. The low rank compression rate is limited (we want more);
2. There is no practical way to train low rank shallow networks.

Tensor Train summary

Tensor Train (TT) decomposition:

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

Mapping example: vector

Build a mapping from the vector $\boldsymbol{b}$ indices to tensor's elements:
$x \leftrightarrow \boldsymbol{i} = (i_1, \ldots, i_d)$

Example (Matlab reshape):

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

Matrices in the TT-format

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

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

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

Notation & terminology:

- $\boldsymbol{W} \in \mathbb{R}^{M \times N}$, $M = m^d$, $N = n^d$;
- $i_k \in \{1, \dots, m\}$, $j_k \in \{1, \dots, n\}$;
- $\boldsymbol{G}_k$ — TT-cores;
- r — TT-rank;

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

Tensor Train layer: feedforward

Input is a $N \times 1$ vector $\boldsymbol{x}$, output is a $M \times 1$ vector $\boldsymbol{y}$:

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

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

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

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

Backpropagation

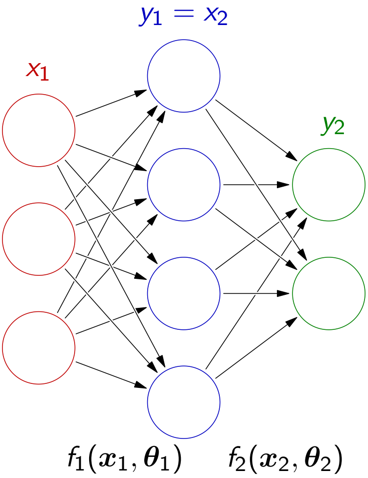
$$L = \frac{1}{2}\sum_{s=1}^{S} ||\boldsymbol{y}_2^s - \boldsymbol{y}^s||_2^2$$

$$\frac{\partial L}{\partial \boldsymbol{y}_2} = \sum_{s=1}^{S} (\boldsymbol{y}_2^s - \boldsymbol{y}^s)$$

$$\frac{\partial L}{\partial \boldsymbol{x}_2} = \sum_i \frac{\partial L}{\partial \boldsymbol{y}_2(i)} \frac{\partial \boldsymbol{y}_2(i)}{\partial \boldsymbol{x}_2}$$

$$= \frac{\partial f_2}{\partial \boldsymbol{x}_2} \frac{\partial L}{\partial \boldsymbol{y}_2}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_2} = \frac{\partial f_2}{\partial \boldsymbol{\theta}_2} \frac{\partial L}{\partial \boldsymbol{y}_2}$$

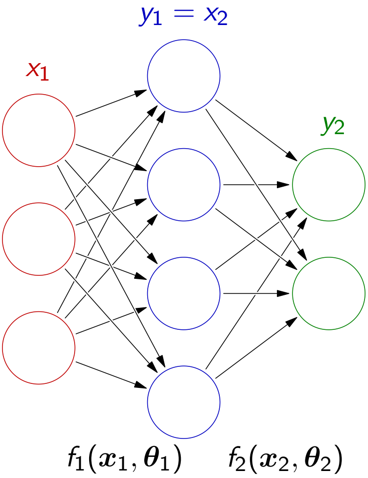


Backpropagation cont'd

From each layer we need only this:

$$\frac{\partial L}{\partial \boldsymbol{x}_k} = g_k^{\boldsymbol{x}}(\boldsymbol{x}_k, \frac{\partial L}{\partial \boldsymbol{y}_k})$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_k} = g_k^{\boldsymbol{\theta}}(\boldsymbol{x}_k, \frac{\partial L}{\partial \boldsymbol{y}_k})$$



Tensor Train layer: backpropagation

Input: vectors $\frac{\partial L}{\partial \boldsymbol{y}} \in \mathbb{R}^M$ and $\boldsymbol{x} \in \mathbb{R}^N$.

Output: $\frac{\partial L}{\partial \boldsymbol{x}}$, $\frac{\partial L}{\partial \boldsymbol{b}}$ and $\frac{\partial L}{\partial \boldsymbol{G}_k[i_k, j_k]}$.

Tensor Train layer: backpropagation

Input: vectors $\frac{\partial L}{\partial \boldsymbol{y}} \in \mathbb{R}^M$ and $\boldsymbol{x} \in \mathbb{R}^N$.
Output: $\frac{\partial L}{\partial \boldsymbol{x}}$, $\frac{\partial L}{\partial \boldsymbol{b}}$ and $\frac{\partial L}{\partial \boldsymbol{G}_k[i_k, j_k]}$.

$$\frac{\partial L}{\partial \boldsymbol{x}} = \boldsymbol{W}^\intercal \frac{\partial L}{\partial \boldsymbol{y}},$$
$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{y}}.$$

$$\underbrace{\frac{\partial L}{\partial \boldsymbol{G}_k[i_k, j_k]}}_{\mathrm{r} \times \mathrm{r}} = \sum_{\boldsymbol{i} \setminus k} \frac{\partial L}{\partial \boldsymbol{y}(\boldsymbol{i})} \frac{\partial \boldsymbol{y}(\boldsymbol{i})}{\partial \boldsymbol{G}_k[i_k, j_k]}$$

Tensor Train layer: Jacobian

We want to differentiate the following expression:

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

Tensor Train layer: Jacobian

We want to differentiate the following expression:

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

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

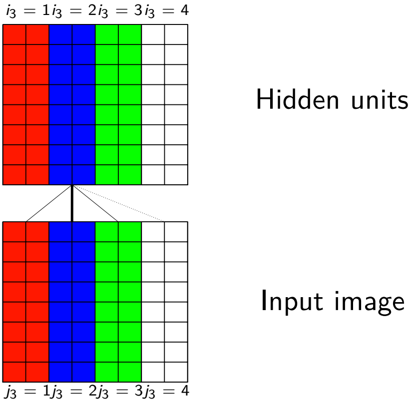
$$\sum_{\boldsymbol{j}^{\setminus k,d}} \boldsymbol{G}_1[i_1, j_1] \dots \cancel{\boldsymbol{G}_k[i_k, j_k]} \dots \boldsymbol{G}_{d-1}[i_{d-1}, j_{d-1}] \underbrace{\sum_{j_d} \boldsymbol{G}_d[i_d, j_d]\, \boldsymbol{x}(\boldsymbol{j})}_{\text{r} \times mn^{d-1}}$$

Intuition

$$\boldsymbol{W}(i_1, i_2, i_3;\ j_1, j_2, j_3) = \underbrace{G_1[i_1, j_1]}_{\in \mathbb{R}} \underbrace{G_2[i_2, j_2]}_{\in \mathbb{R}} \underbrace{G_3[i_3, j_3]}_{\in \mathbb{R}}$$

$\boldsymbol{W} \in \mathbb{R}^{64 \times 64}$
Input $\boldsymbol{x}$ and output $\boldsymbol{y}$ are reshaped to $4 \times 4 \times 4$ tensor.

To vanish all dashed line weights, set $G_3[i_3 = 2, j_3 = 4] = 0$.



Mnist

Mnist dataset, two layered neural network. The input 28×28 image is reshaped to $2 \times 2 \times 7 \times 2 \times 2 \times 7$ tensor.

Network	Error	Neurons	I layer params	II layer
Baseline	2.34%	500	400 000	5 000
TensorNet (one TT-layer)	2.26%	46 656	420	466 560
TensorNet (two TT-layer)	2.07%	15 625	3 360	1 380
TensorNet	2.6%	15 625	350	156 250
TensorNet (random order)	3.5%	15 625	350	156 250

Mnist cont'd

Mnist dataset, two layered neural network. The input 28×28 image is reshaped to $4 \times 7 \times 4 \times 7$ tensor.

Network	**Error**	**Neurons**	**I layer params**	**II layer**
Baseline	2.34%	500	400 000	5 000
TensorNet (one TT-layer)	1.68%	4 096	1 760	40 960

Cifar

Deep convolutional neural network for CIFAR-10 (image classification).
We compressed the last two fully connected layers x11 (the error increased from 23.25% to 23.74%).

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.