Tensor-based approaches for learning flexible neural networks

Yassine Zniyed
CRAN, Université de Lorraine, CNRS (Nancy)

Joint work with Konstantin Usevich, Sebastian Miron, David Brie

28.06.2021, Journée Théorie du DL
Tensor decompositions

Flexible neural networks

Table of Contents

1. Tensor decompositions
   - Tensor decompositions
   - Tensor decompositions and neural networks

2. Flexible neural networks
   - Main idea
   - Jacobian tensor method
   - Flexible AFs
   - Proposed approach
What is a tensor?

**Figure** – A 3-order tensor $\mathbf{X}$. [Cichocki et al., 2014]
Matrix and tensor rank-one decomposition

- outer product: \([p_1 \circ p_2 \circ p_3]_{i,j,k} = p_1(i) \cdot p_2(j) \cdot p_3(k).\)

Figure – Left: Rank-1 matrix - Right: Rank-1 tensor.

- Matrix rank-1 decomposition: \(X = p_1 \circ p_2 = p_1 p_2^T.\)
- What about rank-\(R\) decomposition?
From matrix to tensor rank decomposition

**Figure** – Tensor rank decomposition [Hitchcock, 1927].

- Canonical polyadic decomposition (CPD):

\[
\mathbf{X} = \sum_{l=1}^{R} \mathbf{P}_1(:, l) \circ \mathbf{P}_2(:, l) \circ \mathbf{P}_3(:, l).
\]

- Often unique, if rank \( R \) is smaller than a bound.
Overview

Typical feed-forward convolutional network:
Overview

Typical feed-forward convolutional network:

1. Compression of tensors in convolutional layers.  

2. Interpreting tensor decompositions as product-sum units.  

3. Our proposed architecture: **flexible activation functions**
Main idea

Graphical representation of the basic one-layer flexible NN:

![Graphical representation of the basic one-layer flexible NN](image)

Example:

- Piecewise linear
- Linear
- Polynomial
- Exponential
- Sigmoid

![Example graphs](image)
Some motivation

- **Kolmogorov-Arnold representation theorem (1957):**
  Any continuous $f : [0, 1]^m \rightarrow \mathbb{R}$ can be represented as
  $$f(u) = \sum_{k=1}^{2m+1} \chi_k \left( \sum_{j=1}^{m} \psi_{k,j}(u_j) \right),$$
  with different continuous $\chi_k, \psi_{k,j}$.

- Fixed AF with bias $(g(v_k^T u + b))$.

- Existing architectures (e.g. polynomial NN) are difficult to train from scratch.
Factorization: a tensor-based approach

- Our framework: compression of pre-trained NN.
- Goal: decompose (approximate) given $f$ by

$$f(u) = V^T g(V^T u),$$

with $g(t_1, \cdots, t_r) = [g_1(t_1) \cdots g_r(t_r)]^T$. 

**Borrow ideas from system identification ([Dreesen et al, 2015])**

**Key idea:** by chain rule,

$$J_f(u) = W \cdot \text{diag}(g'_1(v^T_1 u) \cdots g'_r(v^T_r u)) \cdot V^T.$$
Factorization: a tensor-based approach

- Our framework: compression of pre-trained NN.
- Goal: decompose (approximate) given $f$ by

$$f(u) = Wg(V^T u), \text{ with } g(t_1, \ldots, t_r) = [g_1(t_1) \cdots g_r(t_r)]^T.$$ 

Borrow ideas from system identification ([Dreesen et al, 2015])

**Key idea**: by chain rule

$$J_f(u) := \begin{bmatrix} \frac{\partial f_1}{\partial u_1}(u) & \cdots & \frac{\partial f_1}{\partial u_m}(u) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1}(u) & \cdots & \frac{\partial f_n}{\partial u_m}(u) \end{bmatrix} = W \cdot \text{diag}(g'_1(v_1^T u) \cdots g'_r(v_r^T u)) \cdot V^T.$$
Factorization: a tensor-based approach

Algorithm.

1. Evaluate $J_f(u)$ at $N$ “operating points” $u_1, \ldots, u_N \in \mathbb{R}^m$
2. Stack them into a tensor:

$$
N \quad J \quad m
n \quad = \quad J(u_1) \quad J(u_2) \quad \ldots \quad J(u_N)
$$

3. Joint matrix diagonalization $\leftrightarrow$ CPD

$$
J_f(u^{(1)}) = WD^{(1)}V^T,
$$

$$
\vdots
$$

$$
J_f(u^{(N)}) = WD^{(N)}V^T
$$

$J = \sum_{i=1}^r w_i \circ v_i \circ h_i$

4. Retrieve $v_i, w_i$ from factors of the CPD

Issue: assumptions needed to estimate $g_i$
The AFs are expressed as

$$g_l(t) = c_{0,l} + c_{1,l}\phi_1(t) + \cdots + c_{d,l}\phi_d(t).$$

Examples:
The flexible NN model function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is expressed as:

$$f(u) = w_1 \cdot g_1(v_1^T u) + \cdots + w_r \cdot g_r(v_r^T u).$$ (1)

Worth to mention: [Comon, Qi, Usevich, 2017]

- The above decomposition is associated with some interesting uniqueness results, when $\phi_k(t) = t^k$.
- The decomposition (1) is unique, when $d \geq 3$, $m \geq 2$ and $r$ is not too large.
- The decomposition (1) is partially identifiable, when $r > mn$. 
Constrained Coupled Matrix-Tensor Factorization (CMTF)

For a base $g_l(t) = c_{0,l} + c_{1,l} \phi_1(t) + \cdots + c_{d,l} \phi_d(t)$, we solve

$$\min_{w_l, v_l, h_l, z_l} \left\| J - [W, V, H] \right\|^2 + \lambda \cdot \left\| F - WZ^T \right\|^2$$

subject to $h_l = X_l \cdot c_l$, $z_l = Y_l \cdot c_l$, with

$$X_l = \begin{bmatrix} 0 & \phi'_1(v_l^T u^{(1)}) & \cdots & \phi'_d(v_l^T u^{(1)}) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \phi'_1(v_l^T u^{(N)}) & \cdots & \phi'_d(v_l^T u^{(N)}) \end{bmatrix}, \quad Y_l = \begin{bmatrix} 1 & \phi_1(v_l^T u^{(1)}) & \cdots & \phi_d(v_l^T u^{(1)}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(v_l^T u^{(N)}) & \cdots & \phi_d(v_l^T u^{(N)}) \end{bmatrix}.$$

- We use an alternating least squares (block-coordinate descent) with projection
Neural networks compression

**Figure** – Graphical representation of the compression process: (left) the original pretrained NN with fixed AFs - (right) the approximated NN with flexible AFs.
ICDAR and CharNet

ICDAR 2003 dataset: \( \approx 163000 \) train, \( \approx 5300 \) test, 36 classes

CharNet [Jaderberg et al., 2014] (in MatConvNet from Oxford)

<table>
<thead>
<tr>
<th>layer</th>
<th>type</th>
<th>filter size</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>convolutional</td>
<td>9x9</td>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>maxout (/2)</td>
<td></td>
<td>96</td>
<td>48</td>
</tr>
<tr>
<td>conv2</td>
<td>convolutional</td>
<td>9x9</td>
<td>48</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>maxout (/2)</td>
<td></td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>conv3</td>
<td>convolutional</td>
<td>8x8</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>maxout (/4)</td>
<td></td>
<td>512</td>
<td>128</td>
</tr>
<tr>
<td>fc</td>
<td>fully connected</td>
<td>1x1</td>
<td>128</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>maxout (/4)</td>
<td></td>
<td>192</td>
<td>48</td>
</tr>
<tr>
<td>softmax</td>
<td>softmax</td>
<td></td>
<td>48</td>
<td>36</td>
</tr>
</tbody>
</table>
Some results

- we compress the conv3 layer of CharNet (viewed as fully connected layer $\mathbb{R}^{4096} \rightarrow \mathbb{R}^{128}$)
- use only 360 points (10 per class), without fine-tuning
- 4x compression
Thank you!

References: