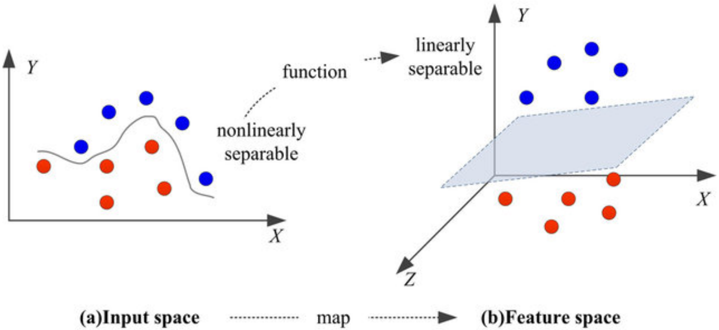
Machine learning with tensor decompositions

Nick Vannieuwenhoven
KU Leuven, Department of Computer Science, NUMA
June 2, 2022

Product feature maps



[1]

A fundamental idea in machine learning is **nonlinearly mapping** low-dimensional inputs in $\mathbb{R}^m$ to a **high-dimensional feature vector space** $\mathbb{R}^N$ and computing a feature vector in $\mathbb{R}^f$ by taking **inner products** with f vectors from $\mathbb{R}^N$.

[1] Figure 3 from Cheng, Feng, Niu, Liao, Water 7(8):4477–4495, 2015.

Mapping a low-dimensional vector $x \in \mathbb{R}^m$ nonlinearly to $\mathbb{R}^N$ with Φ can be accomplished

- **Globally:** One nonlinear map $\Phi : \mathbb{R}^m \to \mathbb{R}^N$.
  For example, a *fully connected layer* in a neural network.
- **Locally:** Several (nonlinear) maps $\phi_i : \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$ combined into a global map Φ.
  For example, a *convolutional layer* in a convolutional neural network.

Mapping a low-dimensional vector $x \in \mathbb{R}^m$ nonlinearly to $\mathbb{R}^N$ with Φ can be accomplished

- **Globally:** One nonlinear map $\Phi : \mathbb{R}^m \to \mathbb{R}^N$.
  For example, a *fully connected layer* in a neural network.
- **Locally:** Several (nonlinear) maps $\phi_i : \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$ combined into a global map Φ.
  For example, a *convolutional layer* in a convolutional neural network.

The usual way to combine local features is by **concatenation**, as in convolutional neural networks. That is, the full feature map is

$$\Phi(x) = \begin{bmatrix} \phi'_1(x) \\ \phi'_2(x) \\ \vdots \\ \phi'_k(x) \end{bmatrix},$$

where ϕ'_i is ϕ_i applied to the correct elements of the input x.

Mathematically, concatenation of the features is the **Cartesian product** of the local feature maps:

$$\Phi = \phi_1' \times \phi_2' \times \cdots \times \phi_k' : \mathbb{R}^m \to \mathbb{R}^N,$$

where $N = \sum_{i=1}^{k} n_i$.

Mathematically, concatenation of the features is the **Cartesian product** of the local feature maps:

$$\Phi = \phi_1' \times \phi_2' \times \cdots \times \phi_k' : \mathbb{R}^m \to \mathbb{R}^N,$$

where $N = \sum_{i=1}^{k} n_i$.

This interpretation suggests an interesting alternative way to combine the features. We can take the **tensor product** of the local feature maps:

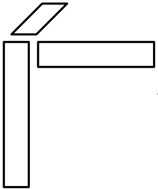
$$\Phi = \phi_1' \otimes \phi_2' \otimes \cdots \otimes \phi_k' : \mathbb{R}^m \to \mathbb{R}^N,$$

where now $N = \prod_{i=1}^{k} n_i$.

The tensor product

The **tensor product**[2] of vectors $\mathbf{f}_1 \in \mathbb{R}^{n_1}, \mathbf{f}_2 \in \mathbb{R}^{n_2}, \ldots, \mathbf{f}_k \in \mathbb{R}^{n_k}$ is

$$\mathbf{f}_1 \otimes \mathbf{f}_2 \otimes \cdots \otimes \mathbf{f}_k =$$



,

where the result is a $n_1 \times n_2 \times \cdots \times n_k$ k-array (or **tensor**).

---

[2] Also referred to as the Kronecker product and outer product, depending on the codomain of $\otimes$.

Note that it takes multiple low-dimensional vectors into an **exponentially large space**. For example,

$$\otimes : \underbrace{\mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{k \text{ factors}} \longrightarrow \mathbb{R}^{n \times n \times \cdots \times n}$$

That is, **the tensor product itself is a very special feature map**!

It seems this tensor product is much less useful than the Cartesian product. After all, the former suffers immensely from the **curse of dimensionality**. Indeed, we want to compute

$$L \circ \Phi = L \circ (\phi_1' \otimes \phi_2' \otimes \cdots \otimes \phi_k'),$$

where

- $\Phi : \mathbb{R}^m \to \mathbb{R}^N$ is a tensor product feature map, and
- $L : \mathbb{R}^N \to \mathbb{R}^f$ is a linear map.

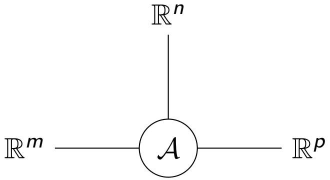
In the naive way, applying $L \circ \Phi$ requires at least $fn_1 n_2 \cdots n_k \geq f2^k$ operations.

To lower the cost, one trick is to **impose further constraints on the linear map** L such that $L \circ \Phi$ can be evaluated efficiently without computing $\phi_1' \otimes \phi_2' \otimes \cdots \otimes \phi_k'$ explicitly.

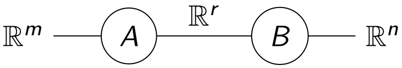
Tensor network decompositions

In the physics literature,[3] a **graphical language** was developed to represent various **tensor decompositions**.

A general third-order tensor $\mathcal{A}$:

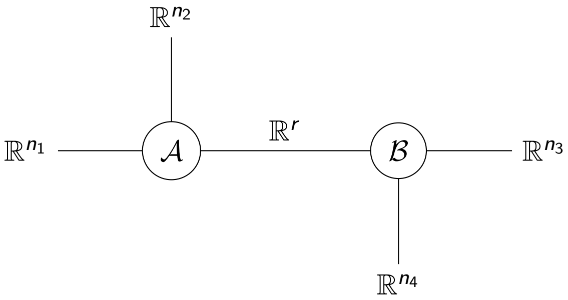


Matrices that have a factorization AB:



[3]See Orús, Ann. Phys. 349:117–158, 2014 for a good introduction.

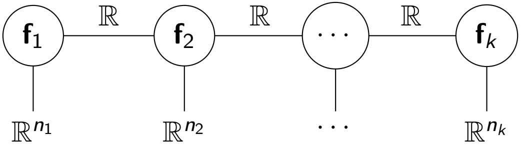
A **tensor network** is a graph of tensors. A tensor at a node lives in the tensor product of the vector spaces on its edges. The whole network represents a tensor, by **contracting over internal edges**, that lives in the tensor product of the **dangling edges**.



In the above example,

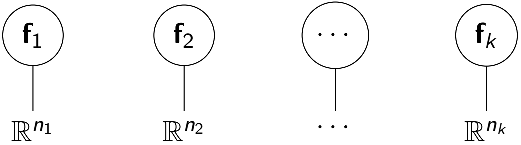
$$\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times r} \qquad \text{and} \qquad \mathcal{B} \in \mathbb{R}^{n_3 \times n_4 \times r}.$$

Example I: Tensor product of vectors (disconnected nodes)

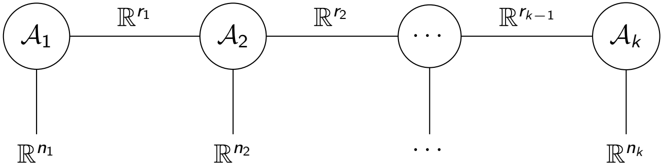


$$\mathcal{A} = \mathbf{f}_1 \otimes \mathbf{f}_2 \otimes \cdots \otimes \mathbf{f}_k$$

This is equivalent to



Example II: Tensor train decomposition (chain network)

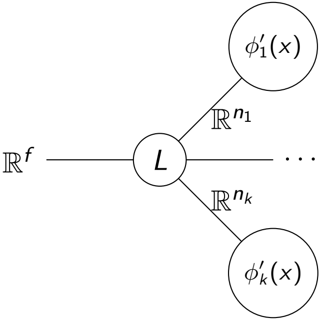


[4]

[4] Oseledets, SIAM J. Sci. Comput. 33(5):2295-2317, 2011.

Supervised learning with tensor trains decompositions

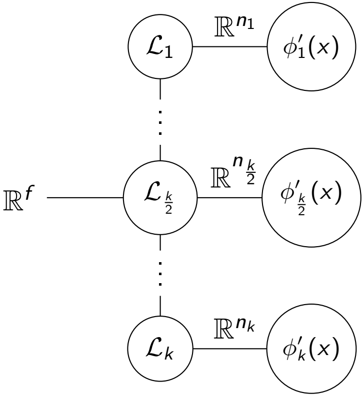
It is known[5] that $L \circ (\phi'_1 \otimes \phi'_2 \otimes \cdots \otimes \phi'_k)$ applied to x is represented by the network



Herein, $L \in \mathbb{R}^{f \times n_1 \times n_2 \times \cdots \times n_k}$ and the result of the computation is a vector in $\mathbb{R}^f$.

[5] This is the universal property of the tensor product in reverse; See Greub, Springer, 1978.

The trick is now to **impose a suitable tensor network structure** on L. For example, with a tensor trains decomposition, we get:



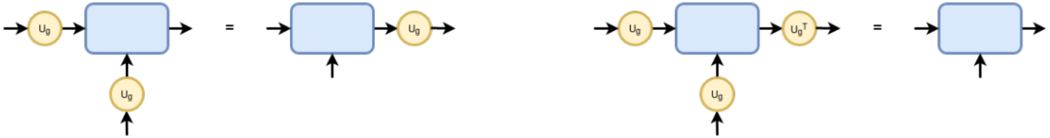
The tensors $\mathcal{L}_i \in \mathbb{R}^{r_{i-1} \times r_i \times n_i}$ are all small-scale if the r_i are small. This mitigates the curse of dimensionality!

This tensor (trains) technology can be plugged into existing machine learning pipelines:

- **Standalone**[6]
- **Neural networks**[7]
- **Support vector machines**[8],[9]

[6]Stoudenmire, Schwab, Supervised learning with tensor networks, NeurIPS, 2016.
[7]Novikov, Podoprikhin, Osokin, Vetrov, Tensorizing Neural Networks, NeurIPS, 2015.
[8]Chen, Batselier, Suykens, Wong, IEEE Trans. Neural Netw. Learn. Sys. 29(10):4621–4632, 2018.
[9]Chen, Batselier, Yu, Wong, Pattern Recognition 122(108337), 2022.

We extended the standalone setup with an efficient scheme to build tensor networks that are **equivariant** under the action of a representation of a **finite group**:[10]



As the **translation equivariance** of convolutional neural networks, this can be viewed as an **inductive bias** for tensor train networks.

It turns out[11] the global equivariance of the represented tensor is guaranteed by local equivariance of the tensors at the nodes of a (loop-free) tensor network.

[10] Sprangers, Vannieuwenhoven, in preparation, 2022.
[11] Singh, Pfeifer, Vidal, Phys. Rev. A 82(050301), 2010.

We applied this equivariant tensor trains network to the **supervised binary classification problem** from [12]:

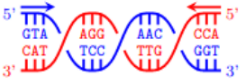


Figure: Reverse compliment symmetry in DNA

- A ↔ T, G ↔ C : $\mathbb{Z}_2$
- Global mirror symmetry
- e.g.: GCTCA ⇔ TGAGC

Problem:

- Predict binding of transcription factor to DNA sequence.

Setup:

- Kronecker product one-hot encoding
- SGD with Nesterov and momentum
- Regularization

Table: Test set results RC-equivariant networks and best benchmark results from [4].

Tasks	Model	**AUROC**
CTCF	Ours	94.10%
	Benchmark	$\mathbf{98.84\%}$
SPI1	Ours	96.53%
	Bechmark	$\mathbf{99.26\%}$
MAX	Ours	$\mathbf{97.06\%}$
	Benchmark	92.80%

[12] Benchmark problems from Mallet, Vert, Reverse-Complement Equivariant Networks for DNA Sequences, In: Adv. Neural Inf. Process. Sys. 34 (NeurIPS 2021)

Thanks for your attention!