Novel tensor framework for neural networks and model reduction

Shashanka Ubaru[1] Lior Horesh[1]

Misha Kilmer[2] Elizabeth Newman[2] Haim Avron[3] Osman Malik[4]

[1]IBM TJ Watson Research Center

[2]Tufts University [3]Tel Aviv University [4]University of Colorado, Boulder

ICERM Workshop on Algorithms for Dimension and Complexity Reduction

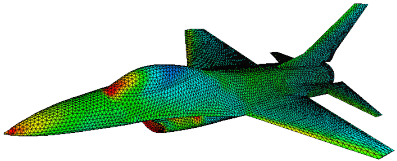
Outline

- Brief introduction to tensors
- Tensor based graph neural networks
- Tensor neural networks
- Numerical Results
- Model reduction for NNs?

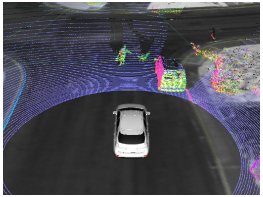
Introduction

- Much of real-world **data** is inherently **multidimensional**

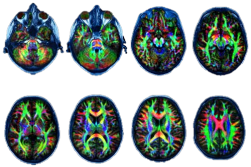
- Many **operators** and **models** are natively **multi-way**



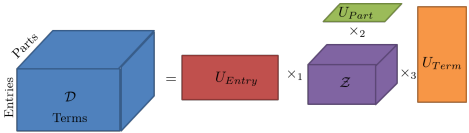
Tensor Applications



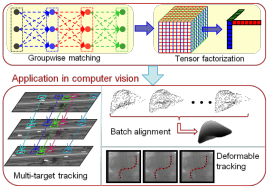
Machine vision



Medical imaging



Latent semantic tensor indexing



Video surveillance, streaming

Ivanov, Mathies, Vasilescu, **Tensor subspace analysis for viewpoint recognition**, ICCV, 2009

Shi, Ling, Hu, Yuan, Xing, **Multi-target tracking with motion context in tensor power iteration**, CVPR, 2014

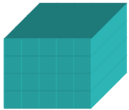
Background and Notation

- **Notation** : $\mathcal{A}^{n_1 \times n_2 \ldots, \times n_d}$ - d^{th} order tensor
  - 0th order tensor - scalar

  - 1st order tensor - vector

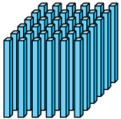
  - 2nd order tensor - matrix

  - 3rd order tensor ...

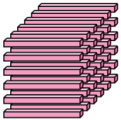


Inside the Box

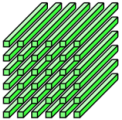
- *Fiber* - a vector defined by fixing all but one index while varying the rest



mode-1
$\boldsymbol{\mathcal{A}}_{:,j,k}$

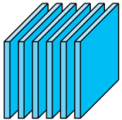


mode-2
$\boldsymbol{\mathcal{A}}_{i,:,k}$

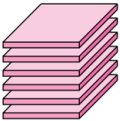


mode-3
$\boldsymbol{\mathcal{A}}_{i,j,:}$, $\boldsymbol{a}_{ij}$

- *Slice* - a matrix defined by fixing all but two indices while varying the rest



lateral
$\boldsymbol{\mathcal{A}}_{:,j,:}$, $\vec{\boldsymbol{\mathcal{A}}}_j$



horizontal
$\boldsymbol{\mathcal{A}}_{i,:,:}$

frontal
$\boldsymbol{\mathcal{A}}_{:,:,k}$, $\boldsymbol{A}^{(k)}$

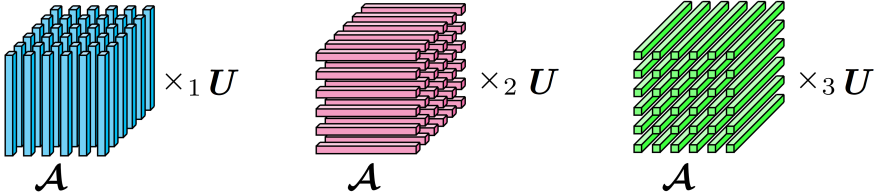
Tensor Multiplication

Definition

- The k - mode multiplication of a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ with a matrix $U \in \mathbb{R}^{j \times n_k}$ is denoted by $\mathcal{A} \times_k U$ and is of size $n_1 \times \cdots \times n_{k-1} \times j \times n_{k+1} \times \cdots \times n_d$
- Element-wise

$$(\mathcal{A} \times_k U)_{i_1 \cdots i_{k-1} j i_{k+1} \cdots i_d} = \sum_{i_k=1}^{n_d} a_{i_1 i_2 \cdots i_d} u_{j i_k}$$

- k-mode multiplication

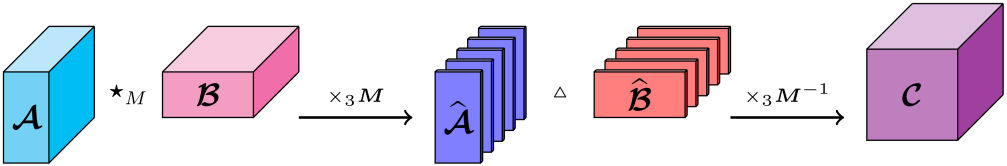


The $\star_M$-Product

Given $\mathcal{A} \in \mathbb{R}^{\ell \times p \times n}$, $\mathcal{B} \in \mathbb{R}^{p \times m \times n}$, and an invertible $n \times n$ matrix M, then

$$\mathcal{C} = \mathcal{A} \star_M \mathcal{B} = \left(\hat{\mathcal{A}} \triangle \hat{\mathcal{B}}\right) \times_3 M^{-1}$$

where $\mathcal{C} \in \mathbb{R}^{\ell \times m \times n}$, $\hat{\mathcal{A}} = \mathcal{A} \times_3 M$, and $\triangle$ multiplies the frontal slices in parallel



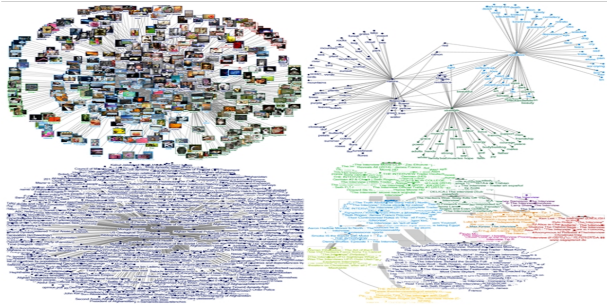
Spatial domain Transform domain Spatial domain

Useful properties: tensor transpose, identity tensor, connection to Fourier transform, invariance to circulant shifts, ...

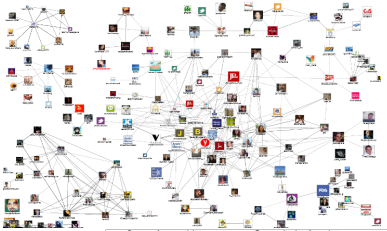
Tensor Graph Convolutional Networks

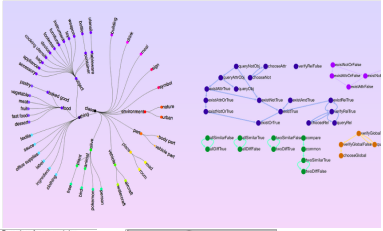
Dynamic Graphs

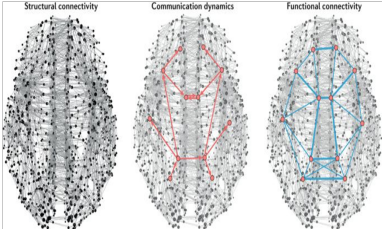
- Graphs are ubiquitous data structures - represent interactions and structural relationships.
- In many real-world applications, underlying graph changes over time.
- Learning representations of dynamic graphs is essential.

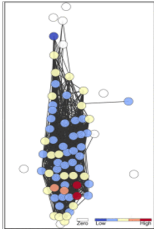


Dynamic Graphs - Applications





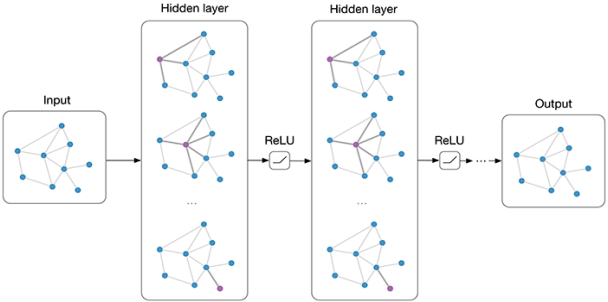




Corporate/financial networks, Natural Language Understanding (NLU), Social networks, Neural activity networks, Traffic predictions.

Graph Convolutional Networks

- ***Graph Neural Networks*** (GNN) popular tools to explore graph structured data.
- ***Graph Convolutional Networks*** (GCN) - based on graph convolution filters - extend convolutional neural networks (CNNs) to irregular graph domains.
- These GNN models operate on a given, static graph.



Courtesy: Image by (Kipf & Welling, 2016).

Graph Convolutional Networks

Motivation:

- Convolution of two signals $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{x} \otimes \mathbf{y} = \mathbf{F}^{-1}(\mathbf{F}\mathbf{x} \odot \mathbf{F}\mathbf{y}),$$

  $\mathbf{F}$ is Fourier transform (DFT matrix).

- Convolution of two node signals $\mathbf{x}$ and $\mathbf{y}$ on a graph with Laplacian $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^T$:

$$\mathbf{x} \otimes \mathbf{y} = \mathbf{U}(\mathbf{U}^T\mathbf{x} \odot \mathbf{U}^T\mathbf{y}).$$

- Filtered convolution:

$$\mathbf{x} \otimes_{filt} \mathbf{y} = h(\mathbf{L})\mathbf{x} \odot h(\mathbf{L})\mathbf{y},$$

  with matrix filter function $h(\mathbf{L}) = \mathbf{U}h(\Lambda)\mathbf{U}^T$.

Graph Convolutional Neural Networks

- Layer of initial convolution based GNNs (Bruna et. al, 2016):
  Given graph Laplacian $\mathbf{L} \in \mathbb{R}^{N \times N}$ and node features $\mathbf{X} \in \mathbb{R}^{N \times F}$:

$$\mathbf{H}_{i+1} = \sigma(h_\theta(\mathbf{L})\mathbf{H}_i\mathbf{W}^{(i)}),$$

  h_θ filter function parametrized by θ, σ a nonlinear function (e.g., RELU), and $\mathbf{W}^{(i)}$ a weight matrix with $\mathbf{H}_0 = \mathbf{X}$.
- Defferrard et al., (2016) used Chebyshev approximation:

$$h_\theta(\mathbf{L}) = \sum_{k=0}^{K} \theta_k T_k(\mathbf{L}).$$

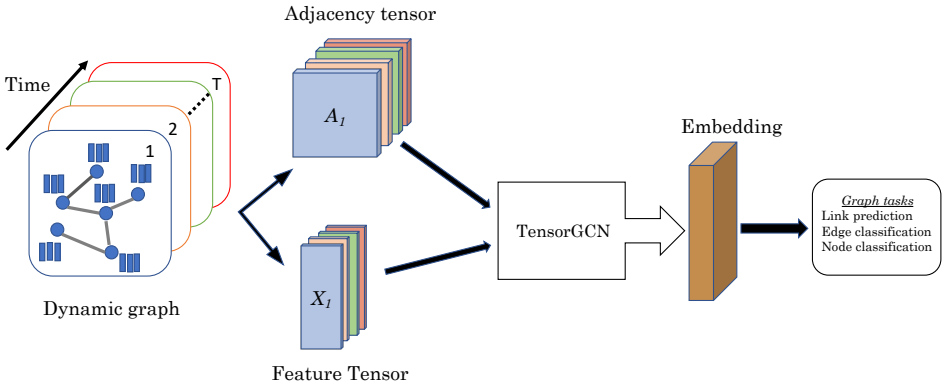
- *GCN* (Kipf & Welling, 2016): Each layer takes form: $\sigma(\mathbf{LXW})$.
- 2-layer example:

$$\mathbf{Z} = \text{softmax}(\mathbf{L}\ \sigma(\mathbf{LXW}^{(0)})\ \mathbf{W}^{(1)})$$

GCN for dynamic graphs

- We consider *time varying*, or *dynamic*, graphs
- **Goal:** Extend GCN framework to the dynamic setting for tasks such as node and edge classification, link prediction.
- **Our approach:** Use the tensor framework
- T adjacency matrices $\mathbf{A}_{::t} \in \mathbb{R}^{N \times N}$ stacked into tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times T}$
- T node feature matrices $\mathbf{X}_{::t} \in \mathbb{R}^{N \times F}$ stacked into tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times F \times T}$

TensorGCN


Adjacency tensor
Time
T
2
1
A_I
Embedding
TensorGCN
Graph tasks
Link prediction
Edge classification
Node classification
X_I
Dynamic graph
Feature Tensor

TensorGCN

- We use the $\star_M$-Product to extend the std. GCN to dynamic graphs.
- We propose tensor GCN model $\sigma(\mathcal{A} \star_M \mathcal{X} \star_M \mathcal{W})$.
- 2-layer example:

$$\mathcal{Z} = \text{softmax}(\mathcal{A} \star_M \sigma(\mathcal{A} \star_M \mathcal{X} \star_M \mathcal{W}^{(0)}) \star_M \mathcal{W}^{(1)}) \tag{1}$$

- We choose $\mathbf{M}$ to be lower triangular and banded:

$$\mathbf{M}_{tk} = \begin{cases} \frac{1}{\min(b,t)} & \text{if } \max(1, t-b+1) \leq k \leq t, \\ 0 & \text{otherwise,} \end{cases}$$

- Can be shown to be consistent with a spatio-temporal message passing model.

O. Malik, S. Ubaru, L. Horesh, M. Kilmer, and H. Avron, **Tensor graph convolutional networks for prediction on dynamic graphs**, 2020

Tensor Neural Networks

Neural Networks

Let $\boldsymbol{a}_0$ be a **feature vector** with an associated **target vector** $\boldsymbol{c}$
Let f be a function which propagates $\boldsymbol{a}_0$ though connected layers:

$$\boldsymbol{a}_{j+1} = \sigma(W_j \cdot \boldsymbol{a}_j + \boldsymbol{b}_j) \text{ for } j = 0, \ldots, N-1,$$

where σ is some **nonlinear, monotonic activation** function

Neural Networks

Let $\boldsymbol{a}_0$ be a **feature vector** with an associated **target vector** $\boldsymbol{c}$
Let f be a function which propagates $\boldsymbol{a}_0$ though connected layers:

$$\boldsymbol{a}_{j+1} = \sigma(W_j \cdot \boldsymbol{a}_j + \boldsymbol{b}_j) \text{ for } j = 0, \ldots, N-1,$$

where σ is some **nonlinear, monotonic activation** function

Goal: Learn the function f which optimizes:

$$\min_{f \in \mathcal{H}} E(f) \equiv \frac{1}{m} \sum_{i=1}^{m} \underbrace{V(\boldsymbol{c}^{(i)}, f(\boldsymbol{a}_0^{(i)}))}_{\text{loss function}} + \underbrace{R(f)}_{\text{regularizer}}$$

$\mathcal{H}$ - **hypothesis space** of functions
rich, restrictive, efficient

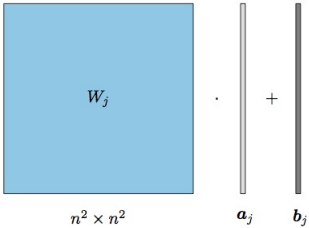
Reduced Parameterization

Given an $n \times n$ image A_0, stored as $\boldsymbol{a}_0 \in \mathbb{R}^{n^2 \times 1}$ and $\vec{\mathcal{A}}_0 \in \mathbb{R}^{n \times 1 \times n}$.

Matrix:

$$\boldsymbol{a}_{j+1} = \sigma(W_j \cdot \boldsymbol{a}_j + \boldsymbol{b}_j)$$

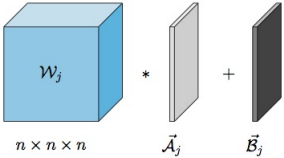
$\boxed{\boldsymbol{n^4} + \boldsymbol{n^2} \text{ parameters}}$



Tensor:

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

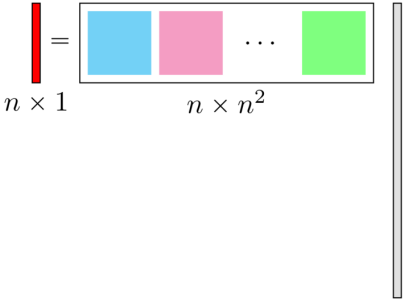
$\boxed{\boldsymbol{n^3} + \boldsymbol{n^2} \text{ parameters}}$



Improved Parametrization

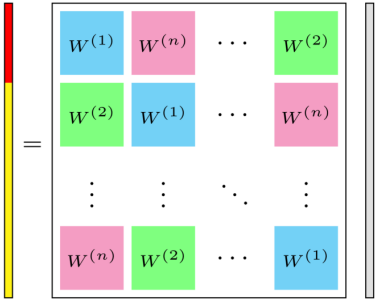
Given an $n \times n$ image A_0, stored as $\boldsymbol{a}_0 \in \mathbb{R}^{n^2 \times 1}$ and $\vec{\mathcal{A}}_0 \in \mathbb{R}^{n \times 1 \times n}$.

Matrix:



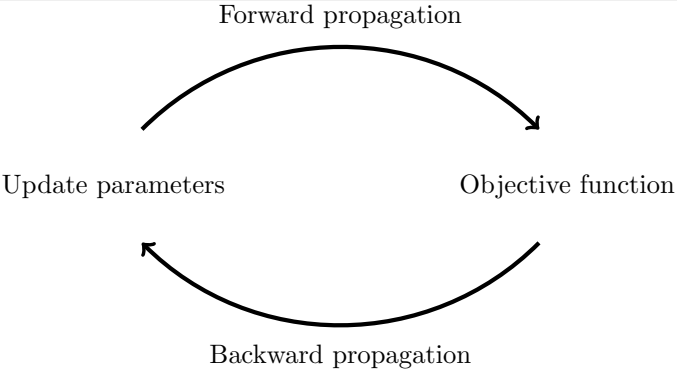
$\boldsymbol{n}$ features

Tensor:



$\boldsymbol{n^2}$ features

Tensor Neural Networks (tNNs)



M. Nielsen, **Neural networks and deep learning**, 2017

Tensor Neural Networks (tNNs)

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

Update parameters

Objective function

Backward propagation

M. Nielsen, **Neural networks and deep learning**, 2017

Tensor Neural Networks (tNNs)

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

Update parameters

$$E = \frac{1}{2}||W_N \cdot \texttt{unfold}(\vec{\mathcal{A}}_N) - \boldsymbol{c}||_F^2$$

Backward propagation

M. Nielsen, **Neural networks and deep learning**, 2017

Tensor Neural Networks (tNNs)

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

Update parameters

$$E = \frac{1}{2}||W_N \cdot \texttt{unfold}(\vec{\mathcal{A}}_N) - \boldsymbol{c}||_F^2$$

$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top \star_M (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

where $\vec{\mathcal{Z}}_{j+1} = \mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j$ and $\odot$ is the pointwise product

$$\delta\vec{\mathcal{A}}_j := \frac{\partial E}{\partial \vec{\mathcal{A}}_j}$$

M. Nielsen, **Neural networks and deep learning**, 2017

Tensor Neural Networks (tNNs)

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

Update parameters

$$E = \frac{1}{2}||W_N \cdot \texttt{unfold}(\vec{\mathcal{A}}_N) - \boldsymbol{c}||_F^2$$

$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top \star_M (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

where $\vec{\mathcal{Z}}_{j+1} = \mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j$ and $\odot$ is the pointwise product

$$\delta\vec{\mathcal{A}}_j := \frac{\partial E}{\partial \vec{\mathcal{A}}_j} = \frac{\partial E}{\partial \vec{\mathcal{A}}_{j+1}} \frac{\partial \vec{\mathcal{A}}_{j+1}}{\partial \vec{\mathcal{Z}}_{j+1}} \frac{\partial \vec{\mathcal{Z}}_{j+1}}{\partial \vec{\mathcal{A}}_j}$$

M. Nielsen, **Neural networks and deep learning**, 2017

Tensor Neural Networks (tNNs)

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

$$\delta\mathcal{W}_j = (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1})) \star_M \vec{\mathcal{A}}_j^\top$$

$$\delta\vec{\mathcal{B}}_j = \delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1})$$

$$E = \frac{1}{2}||W_N \cdot \texttt{unfold}(\vec{\mathcal{A}}_N) - \boldsymbol{c}||_F^2$$

$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top \star_M (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

where $\vec{\mathcal{Z}}_{j+1} = \mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j$ and $\odot$ is the pointwise product

M. Nielsen, **Neural networks and deep learning**, 2017

Tensor Neural Networks (tNNs)

$$\vec{\mathcal{A}}_{j+1} = \sigma(\mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j)$$

$$\delta\mathcal{W}_j = (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1})) \star_M \vec{\mathcal{A}}_j^\top$$

$$\delta\vec{\mathcal{B}}_j = \delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1})$$

$$E = \frac{1}{2}||W_N \cdot \texttt{unfold}(\vec{\mathcal{A}}_N) - \boldsymbol{c}||_F^2$$

$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top \star_M (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

where $\vec{\mathcal{Z}}_{j+1} = \mathcal{W}_j \star_M \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j$ and $\odot$ is the pointwise product

Update parameters = Gradient descent!

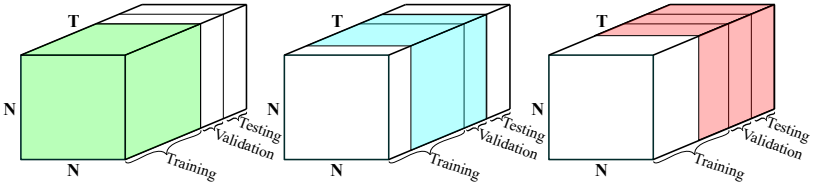
M. Nielsen, **Neural networks and deep learning**, 2017

Numerical Results

TensorGCN - Datasets

Table: Dataset statistics. By partitioning the data into windows of the specified length results in the given number of graphs.

						Partitioning		
Dataset	Nodes	Edges	No. graphs	Window length	Classes	S_{train}	S_{val}	S_{test}
Bitcoin OTC	6,005	35,569	135	14 days	2	95	20	20
Bitcoin Alpha	7,604	24,173	135	14 days	2	95	20	20
Reddit	3,818	163,008	86	14 days	2	66	10	10
Chess	7,301	64,958	100	31 days	3	80	10	10



Partitioning of $\mathcal{A}$ into training, validation and testing data.

TensorGCN - Edge classification results

Table: Results for edge classification. Performance measures is F1 score.

	Dataset			
Method	Bitcoin OTC	Bitcoin Alpha	Reddit	Chess
WD-GCN	0.2062	0.1920	**0.2337**	0.4311
EvolveGCN	0.3284	0.1609	0.2012	0.4351
GCN	0.3317	0.2100	0.1805	0.4342
TensorGCN (Proposal)	**0.3529**	**0.2331**	0.2028	**0.4708**

$$\text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

TensorGCN - Link Prediction results

Table: Results for link prediction. Performance measure is Mean Average Precision (MAP).

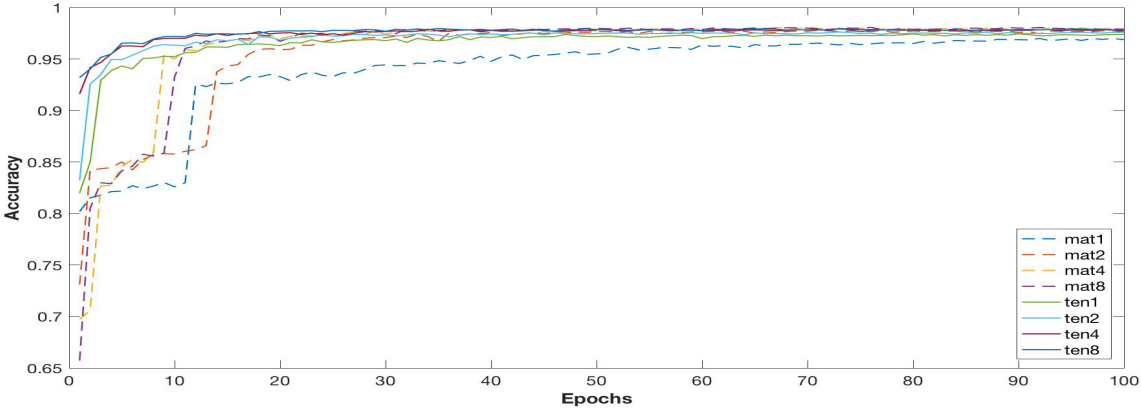
	Dataset			
Method	Bitcoin OTC	Bitcoin Alpha	Reddit	Chess
WD-GCN	0.6979	0.8067	**0.1818**	0.1077
EvolveGCN	0.6019	0.3474	0.1730	0.0655
GCN	0.6872	0.7392	0.1788	0.0852
TensorGCN (Proposal)	**0.7817**	**0.8094**	0.1601	**0.1736**

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

Tensor vs. Matrix Learning: MNIST Database Results

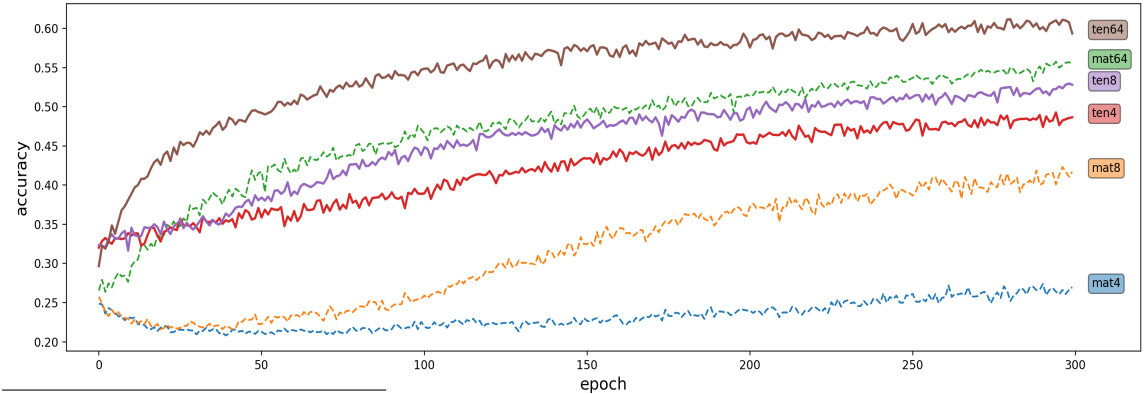
Data: 28×28 grayscale images of handwritten digits, 60000 train, 10000 test
Fixed parameters: $h = 0.1$, $\alpha = 0.1$, $\sigma = tanh$, batch size $= 20$, 100 epochs
Learnable parameters: matrix - $28^4 N + 28^2 N$, tensor - $28^3 N + 28^2 N$



L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, (2019)

Tensor vs. Matrix Learning: CIFAR-10 Database Results

Data: $32 \times 32 \times 3$ RGB images from 10 classes, 50000 train, 10000 test
Fixed parameters: $h = 0.1$, $\alpha = 0.01$, $\sigma = tanh$, batch $= 100$, 300 epochs, $M =$ DCT matrix.
Learnable params: mat-$(3^2 \cdot 32^4)N + 3 \cdot 32^2 N$, ten-$(3^2 \cdot 32^3)N + 3 \cdot 32^2 N$



A. Krizhevsky, **Learning multiple layers of features from tiny images**, 2009

L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, (2019)

Model reduction for NN?

Recall - Proper Orthogonal Decomposition

- Dynamical system (scalar nonlinear PDE):

$$\frac{\partial \mathbf{y}(t)}{\partial t} = \mathbf{A}\mathbf{y}(t) + \mathbf{F}(\mathbf{y}(t)),$$

  $t \in [0, T]$ denotes time, $\mathbf{y}(t) = [\mathbf{y}_1(t), \ldots, \mathbf{y}_n(t)]^T \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n \times n}$ a constant matrix, and $\mathbf{F}$ a nonlinear function $\mathbf{F} = [F(\mathbf{y}_1(t)), \ldots, F(\mathbf{y}_n(t))]^T$.
- The discretized system:

$$\mathbf{A}\mathbf{y}(\mu) + \mathbf{F}(\mathbf{y}(\mu)) = 0,$$

- Corresponding Jacobian,

$$\mathbf{J}(\mathbf{y}(\mu)) := \mathbf{A} + \mathbf{J}_{\mathbf{F}}(\mathbf{y}(\mu)),$$

  with

$$\mathbf{J}_{\mathbf{F}}(\mathbf{y}(\mu)) = diag\{F'(\mathbf{y}_1(\mu)), \ldots, F'(\mathbf{y}_n(\mu))\} \in \mathbb{R}^{n \times n},$$

  F' denotes the first derivative of F.

Proper Orthogonal Decomposition

- POD uses first k left singular vectors of the snapshot matrix defined as: $\mathbf{Y} = \{\mathbf{y}^1, \ldots, \mathbf{y}^{n_s}\}$. Given the SVD of $\mathbf{Y}$,

$$\mathbf{Y} = \mathbf{V}\Sigma\mathbf{W}^T$$

- Projecting the system as:

$$\frac{\partial \tilde{\mathbf{y}}(t)}{\partial t} = \mathbf{V}_k^T \mathbf{A} \mathbf{V}_k \tilde{\mathbf{y}}(t) + \mathbf{V}_k^T \mathbf{F}(\mathbf{V}_k \tilde{\mathbf{y}}(t)).$$

The reduced order system becomes:

$$\tilde{\mathbf{A}}\tilde{\mathbf{y}}(\mu) + \mathbf{V}_k^T \mathbf{F}(\mathbf{V}_k \tilde{\mathbf{y}}(\mu)) = 0,$$

and the corresponding Jacobian,

$$\tilde{\mathbf{J}}(\tilde{\mathbf{y}}(\mu)) := \tilde{\mathbf{A}} + \mathbf{V}_k^T \mathbf{J}_{\mathbf{F}}(\mathbf{V}_k \tilde{\mathbf{y}}(\mu)) \mathbf{V}_k,$$

where $\tilde{\mathbf{A}} = \mathbf{V}_k^T \mathbf{A} \mathbf{V}_k$.

Discrete empirical interpolation method

- DEIM approximates the nonlinear function by projecting onto a space generated by function with basis of dimension $m \ll n$.
- Considering a subspace $\mathbf{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$, we approximate $\mathbf{F}(\tau) \approx \mathbf{U}\mathbf{c}(\tau)$, where $\mathbf{c}(\tau)$ is corresponding coefficient vector.
- An interpolation matrix:
$$\mathbf{P} = [\mathbf{e}_{\phi_1}, \ldots, \mathbf{e}_{\phi_m}] \in \mathbb{R}^{n \times m}$$
where $\mathbf{e}_{\phi_i}$ is a basis vector, i.e., ϕ_ith column of identity matrix.
- The nonlinear function in the PDE and the Jacobian approximated as:
$$\mathbf{F}(\mathbf{V}_k \tilde{\mathbf{y}}(\mu)) \approx \mathbf{U}(\mathbf{P}^T\mathbf{U})^{-1}\mathbf{F}(\mathbf{P}^T\mathbf{V}_k\tilde{\mathbf{y}}(\mu)),$$
$$\tilde{\mathbf{J}}_{\mathbf{F}}(\tilde{\mathbf{y}}(\mu)) \approx \mathbf{V}_k^T\mathbf{U}(\mathbf{P}^T\mathbf{U})^{-1}\mathbf{J}_{\mathbf{F}}(\mathbf{P}^T\mathbf{V}_k\tilde{\mathbf{y}}(\mu))\mathbf{P}^T\mathbf{V}_k.$$

Chaturantabut and Sorensen , **Discrete Empirical Interpolation for nonlinear model reduction**, 2009

Saibaba, **Randomized Discrete Empirical Interpolation Method for Nonlinear Model Reduction**, (2019)

PDE based NNs

- ***Residual networks (ResNets)***: Given training data by $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_s] \in \mathbb{R}^{n \times s}$ and target $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_s] \in \mathbb{R}^{d \times s}$, N layers ResNet is given by:

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \sigma(\mathbf{A}_j \mathbf{Y}_j + b_j) \qquad for \qquad j = 0, \ldots, N-1.$$

- General formulation:

$$\mathbf{F}(\theta, \mathbf{Y}) = \mathbf{A}_2(\theta^{(3)})\sigma(\mathcal{N}(\mathbf{A}_1(\theta^{(1)})\mathbf{Y}, \theta^{(2)})).$$

- with forward propagation as

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathbf{F}(\theta^{(j)}, \mathbf{Y}_j) \qquad for \qquad j = 0, \ldots, N-1.$$

- Forward Euler discretization of the initial value problem

$$\begin{aligned} \partial_t \mathbf{Y}(\theta, t) &= \mathbf{F}(\theta(t), \mathbf{Y}(t)), \quad for \quad t \in (0, T] \\ \mathbf{Y}(\theta, 0) &= \mathbf{Y}_0. \end{aligned}$$

Ruthotto and Haber, **Deep Neural Networks Motivated by Partial Differential Equations**, 2019

Model reduction for PDE-NN?

- Consider stable variant of Convolutional ResNets with symmetric layer:

$$\mathbf{F}_{sym}(\theta, \mathbf{Y}) = -\mathbf{A}(\theta)^T \sigma(\mathcal{N}(\mathbf{A}(\theta)\mathbf{Y}, \theta)).$$

- The Jacobian of this function with respect to the features will be:

$$\mathbf{J}_{\mathbf{F}}(\mathbf{Y}) = \mathbf{A}(\theta)^T diag(\sigma'(\mathbf{A}(\theta)\mathbf{Y}))\mathbf{A}(\theta),$$

  σ' derivative of pointwise nonlinearity.

- **Reduced parameters via. DEIM:** Precompute the projection basis $\mathbf{U}$ and interpolation matrix $\mathbf{P}$,

$$\mathbf{F}_{sym}(\theta, \mathbf{Y}) \approx \mathbf{U}(\mathbf{P}^T\mathbf{U})^{-1}\tilde{\mathbf{F}}_{sym}(\theta, \mathbf{P}^T\mathbf{Y}),$$

  where $\tilde{\mathbf{F}}_{sym}(\theta, \mathbf{P}^T\mathbf{Y}) = -\tilde{\mathbf{A}}(\theta)^T \sigma(\mathcal{N}(\tilde{\mathbf{A}}(\theta)\mathbf{P}^T\mathbf{Y}, \theta))$ where $\tilde{\mathbf{A}} \in \mathbb{R}^{d\times m}$ is the reduced weight matrix to be learned.

- Effective approach to compute $\mathbf{U}$ and $\mathbf{P}$, as the function $\mathbf{F}$ depends on θ.

Thank you!