CSE 392: Matrix and Tensor Algorithms for Data

Instructor: Shashanka Ubaru

University of Texas, Austin
Spring 2024

Lecture 23: Randomized t-SVD, t-product applications

Outline

1. Randomized t-SVD

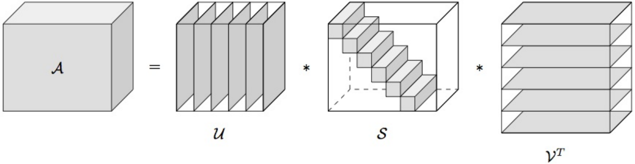
2. t-product applications
    - Face Recognition
    - Tensor Neural Network
    - Tensor Graph Neural Networks

t-SVD

Theorem: For $\mathcal{A} \in \mathbb{R}^{m\times\ell\times n}$ there exists a full tensor-SVD

$$\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^\top,$$

with $m \times m \times n$ orthogonal tensor $\mathcal{U}$, $\ell \times \ell \times n$ orthogonal tensor $\mathcal{V}$, and $m \times \ell \times n$ f-diagonal tensor $\mathcal{S}$ ordered such that the singular tubes $\mathbf{s}_i = \mathcal{S}_{i,i,:}$ have $\|\mathbf{s}_1\|_F^2 \geq \|\mathbf{s}_2\|_F^2 \geq \cdots$.



The **t-rank** is the number of non-zero tube-fibers in $\mathcal{S}$.

t-SVD Computation

The t-SVD can be computed efficiently (in parallel) by moving to the Fourier domain.

- Compute $\widehat{\mathcal{A}}$
- For $i = 1, \ldots, n$, find matrix SVD of each frontal slice: $\widehat{\mathcal{U}}_{:,:,i}\widehat{\mathcal{S}}_{:,:,i}\widehat{\mathcal{V}}^H_{:,:,i} = \widehat{\mathcal{A}}_{:,:,i}$
- To get $\mathcal{U}, \mathcal{S}, \mathcal{V}$, inverse FFT along tube fibers of $\widehat{\mathcal{U}}, \widehat{\mathcal{S}}, \widehat{\mathcal{V}}$.

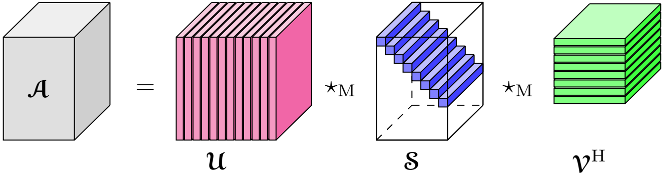
Tensor-tensor SVDs

Theorem (Kilmer, Horesh, Avron, Newman)

Let $\mathcal{A}$ *be a* $m \times p \times n$ *tensor and* $\boldsymbol{M}$ *a non-zero multiple of a unitary/orthogonal matrix. The (full)* $\star_M$ *tensor SVD (t-SVDM) is*

$$\mathcal{A} = \mathcal{U} \star_M \mathcal{S} \star_M \mathcal{V}^{\mathrm{H}} = \sum_{i=1}^{\min(m,p)} \mathcal{U}_{:,i,:} \star_M \mathcal{S}_{i,i,:} \star_M \mathcal{V}^{\mathrm{H}}_{:,i,:}$$

with $\mathcal{U}$*,* $\mathcal{V}$ $\star_M$*-unitary, &* $\|\mathcal{S}_{1,1,:}\|_F^2 \geq \|\mathcal{S}_{2,2,:}\|_F^2 \geq \dots$



Algorithm

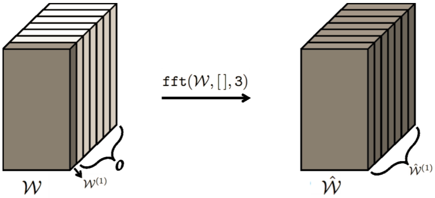
$\widehat{\mathcal{A}} \leftarrow \mathcal{A} \times_3 \boldsymbol{M}$
$i = 1, \ldots, n$
$[\widehat{\mathcal{U}}_{:,:,i}, \widehat{\mathcal{S}}_{:,:,i}, \widehat{\mathcal{V}}_{:,:,i}] = \texttt{svd}(\widehat{\mathcal{A}}_{:,:,i})$
$\mathcal{U} = \widehat{\mathcal{U}} \times_3 \boldsymbol{M}^{-1},\ \mathcal{S} = \widehat{\mathcal{S}} \times_3 \boldsymbol{M}^{-1},\ \mathcal{V} = \widehat{\mathcal{V}} \times_3 \boldsymbol{M}^{-1}.$

Perfectly (i.e. embarrassingly) parallelizable!

For face i, exist singular values $\hat{\sigma}_i^{(j)}$, $j = 1, .., \rho_i$

Randomized Variants

Need definition of a Gaussian Random Tensor, $\mathcal{W}$, then consider $\mathcal{A} * \mathcal{W}$:



Exercise: Verify that each frontal slice of $\widehat{\mathcal{W}}$ is the same.

Zhang, Saibaba, Kilmer, Aeron, NLAA, 2018

Randomized t-SVD with Subspace-type Iteration

Input $\mathcal{A} \in \mathbb{R}^{m \times \ell \times n}$, target truncation term k, oversampling parameter p, the number of iterations q

Output $\mathcal{U}_k \in \mathbb{R}^{m \times k \times n}$, $\mathcal{S}_k \in \mathbb{R}^{k \times k \times n}$, and $\mathcal{V}_k \in \mathbb{R}^{\ell \times k \times n}$

- Generate a Gaussian random tensor $\mathcal{W} \in \mathbb{R}^{\ell \times (k+p) \times n}$
- Form $\mathcal{Y} = (\mathcal{A} * \mathcal{A}^\top)^q * \mathcal{A} * \mathcal{W}$;
- Form tensor QR factorization $\mathcal{Y} = \mathcal{Q} * \mathcal{R}$;
- Form a tensor $\mathcal{B} = \mathcal{Q}^\top * \mathcal{A}$, the size of $\mathcal{B}$ is $(k+p) \times \ell \times n$;
- Compute t-SVD of $\mathcal{B}$, truncate it, and obtain $\mathcal{B}_k = \mathcal{U}_k * \mathcal{S}_k * \mathcal{V}_k^\top$;
- Form the rt-SVD of $\mathcal{A}$, $\mathcal{A} \approx (\mathcal{Q} * \mathcal{B}_k) = (\mathcal{Q} * \mathcal{U}_k) * \mathcal{S}_k * \mathcal{V}_k^\top$.

In practice, implemented in transform domain, with parallel matrix computations.

Analysis: Expectation of Error

Implemented in transform domain, different iter count q_i per face.

Theorem

The output satisfies

$$\begin{aligned}
\mathbb{E}\|\mathcal{A}-\mathcal{Q}*\mathcal{Q}^\top*\mathcal{A}\|^2 &\leq \mathbb{E}\|\mathcal{A}-\mathcal{Q}*\mathcal{B}_k\|^2 \\
&\leq \frac{1}{n}\left(\sum_{i=1}^{n}\left(1+\frac{k(\tau_k^{(i)})^{4q_i}}{p-1}\right)\left(\sum_{j>k}(\widehat{\sigma}_j^{(i)})^2\right)\right),
\end{aligned}$$

where k is a target truncation term, $p \geq 2$ is the oversampling parameter, $\mathbf{q}$ is the iterations count vector, and the singular value gap $\tau_k^{(i)} = \frac{\widehat{\sigma}_{k+1}^{(i)}}{\widehat{\sigma}_k^{(i)}} \ll 1$.

If the term in blue were 1, then optimal.

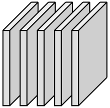
Zhang, Saibaba, Kilmer, Aeron, NLAA, 2018

Impact on Recognition Rate: Cropped Yale B, $k = 25$

	fold 1	fold 9	fold 10
t-SVD			
	0.9912	0.7368	0.9825
rt-SVD			
min	0.9912	0.7368	0.9737
mean	0.9912	0.7368	0.9772
max	0.9912	0.7368	0.9912
rt-SVD $q = 1$			
min	0.9912	0.7368	0.9737
mean	0.9912	0.7368	0.9833
max	0.9912	0.7368	0.9912
rt-SVD $q = 2$			
min	0.9912	0.7368	0.9825
mean	0.9912	0.7368	0.9882
max	0.9912	0.7368	0.9912

t-product applications

Application: Facial Recognition



$\vec{\mathcal{A}}_j$ is mean subtracted image

- $\vec{\mathcal{X}}_j$, $j = 1, 2, \ldots, m$ are the training images
- $\vec{\mathcal{Y}}$ is the mean image
- $\vec{\mathcal{A}}_j = \vec{\mathcal{X}}_j - \vec{\mathcal{Y}}$ has the mean-subtracted images
- $\mathcal{K} = \mathcal{A} * \mathcal{A}^\top = \mathcal{U} * \mathcal{S} * \mathcal{S}^\top * \mathcal{U}^\top$ is the covariance tensor
- Left orthogonal $\mathcal{U}$ contains the principal components, so

$$\vec{\mathcal{A}}_j \approx \mathcal{U}_{:,1:k,:} * \underbrace{(\mathcal{U}^\top_{:,1:k,:} * \vec{\mathcal{A}}_j)}_{\text{tensor coefs}}$$

- Note $\mathcal{U}_{:,1:k,:} * \mathcal{U}^\top_{:,1:k,:}$ is orthogonal projection tensor.

Matching Coefficients

We keep the basis $\mathcal{U}_{:,1:k,:}$ and the tensor coefficients $\mathcal{U}_{:,1:k,:}^{\top} * \vec{\mathcal{A}}_j$.

When a new (mean subtracted) image, oriented as a tensor, $\vec{\mathcal{B}}$, comes in, we compute its tensor coefficients $\mathcal{U}_{:,1:k,:}^{\top} * \vec{\mathcal{B}}$

Then we look for the image with the smallest Frobenius norm difference with the tensor coefficients in the database.

This is fundamentally different treatment than “eigenfaces.”

Facial Recognition Task

Take 256 image subset (4 people, 64 different lighting conditions).

Randomly removed 1 image per person.

The Extended Yale Face Database B, `http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html`

Facial Recognition

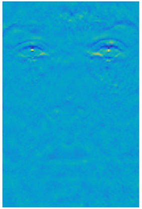
$\mathcal{A}$ is 192 × 252 × 128. Truncated to $k = 15$. $\frac{\|\mathcal{A}-\widehat{\mathcal{A}}\|}{\|\mathcal{A}\|} = .115$

Recall, this means

$$\mathcal{A} \approx \mathcal{U}_{:,1:k,:} * (\mathcal{S}_{1:k,1:k,:} * \mathcal{V}^{\top}_{:,1:k,:}) = \mathcal{U}_{:,1:k,:} * \underbrace{(\mathcal{U}^{\top}_{:,1:k,:} * \mathcal{A})}_{\mathcal{C}},$$

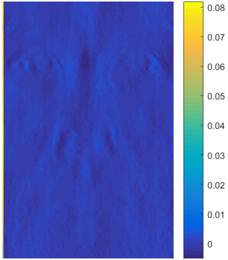
so the jth lateral slice, a (mean subtracted) image, is $\mathcal{A}_{:,j,:} = \sum_{i=1}^{k} \mathcal{U}_{:,i,:} * \mathbf{c}_{i,j}$.

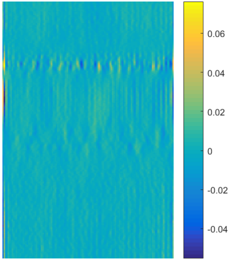
Difference image of first slice:



Facial Recognition

Interpretability: The $\mathcal{U}_{:,i,:}$ are the basis elements, do we expect they look like ghost images as in eigenfaces?

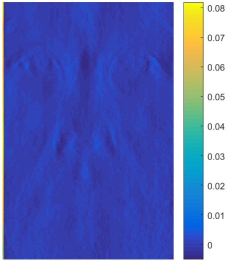


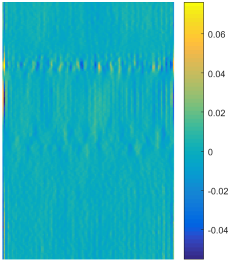


Exercise: How much (implicit) storage is required for the training data, and what is the ratio of this to the storage for $\mathcal{A}$?

Facial Recognition

Not necessarily - remember, these are NOT linear combinations anymore.





Exercise: How much (implicit) storage is required for the training data, and what is the ratio of this to the storage for $\mathcal{A}$?

Facial Recognition

How well does the matrix PCA approximation to $k = 15$ terms compare? The relative error is about $2\times$ as large!

All 4 test cases were correctly identified by the tensor-based PCA approach. Only 3 of the 4 were correctly identified by the matrix-based PCA approach.

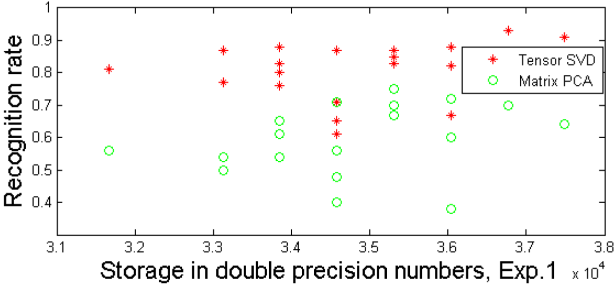
Same data, treated differently!

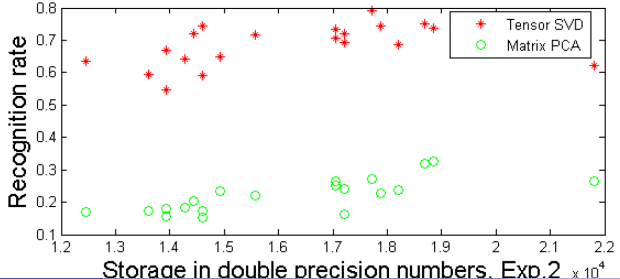
Facial Recognition Task, Revisited $\boldsymbol{M}$ is DFT

- Experiment 1: randomly select 15 images of each person as training, test all remaining images
- Experiment 2: randomly selected 5 images of each person as training, test all remaining images
- 20 trials for each experiment

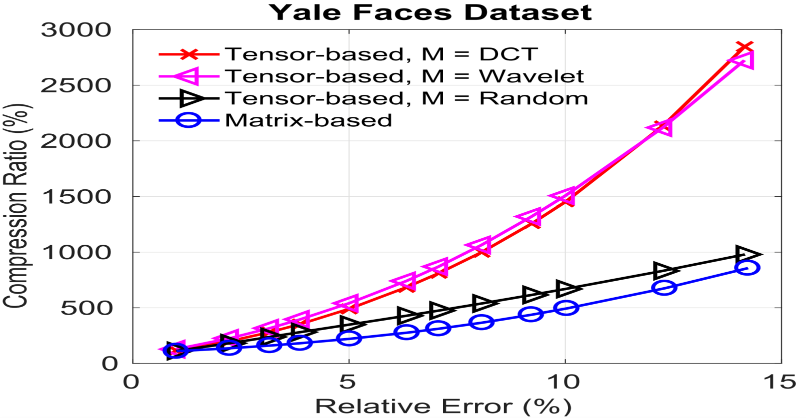
Results from Hao, et al, SIIMS, 2013

t-SVDII vs. PCA





Yale Example



Kilmer, Horesh, Avron, Newman, **Tensor-tensor products for optimal representation and compression**, (2021)

Truncated-HOSVD in the $\star_M$ Framework

Define $\boldsymbol{M} = (\mathbf{U}^{(3)})^\top$ from the HOSVD

Then we can express the HOSVD in the $\star_M$ tensor framework!

We can show that the t-SVDM, t-SVDMII are superior to tr-HOSVD for appropriate truncation levels, as well.

Hyperspectral Results

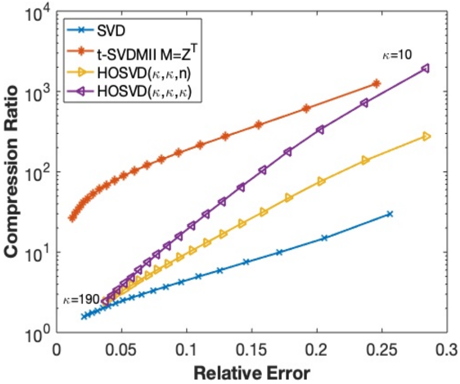
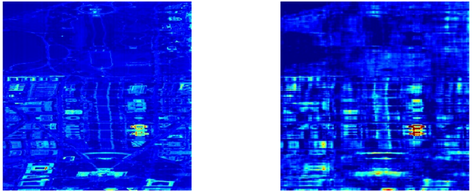


Figure: Hyperspectral compression vs. relative error. Best performance are points lying closest to the upper left; i.e., the most compression for the smallest relative error.

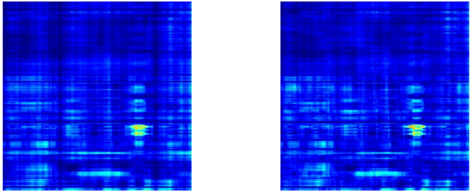
Numerical Results

Approximation of hyperspectral wavelength 10, corresponds to upper right of graph.



Original

t-SVDMII, $\gamma = 0.94$



tr-HOSVD$(10, 10, 10)$

tr-HOSVD$(14, 14, 14)$

Neural Networks, Hypothetically

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, Hypothetically

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

Neural Networks, Hypothetically

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

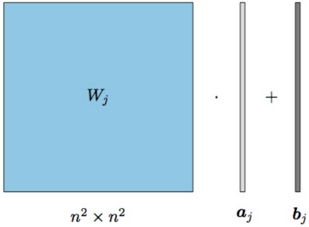
Less is More: Reduced Parameterization

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:

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

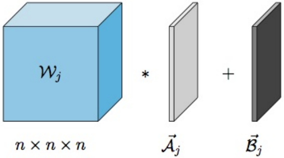
$$\boxed{\boldsymbol{n^4} + \boldsymbol{n^2} \text{ parameters}}$$



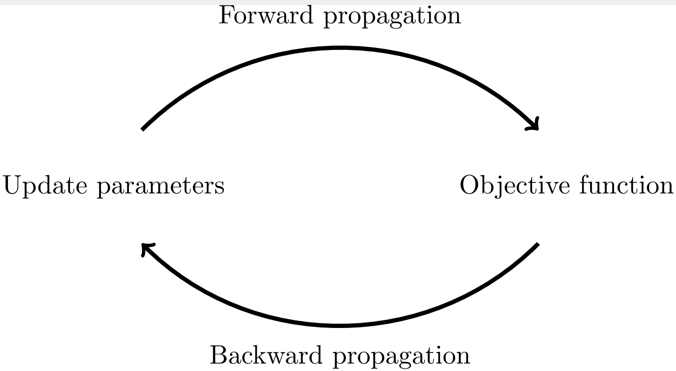
Tensor:

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

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



Tensor Neural Networks (tNNs)



Forward propagation

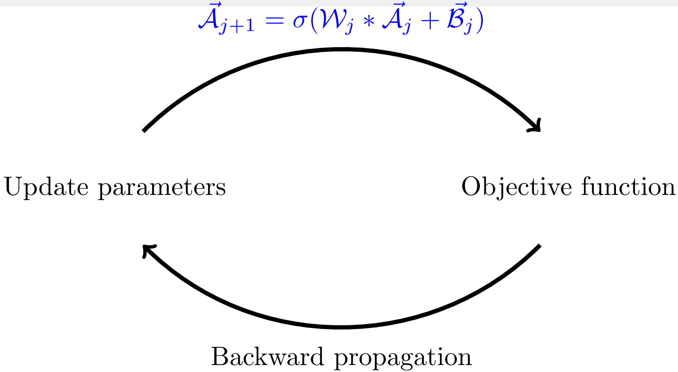
Update parameters

Objective function

Backward propagation

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

Tensor Neural Networks (tNNs)



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

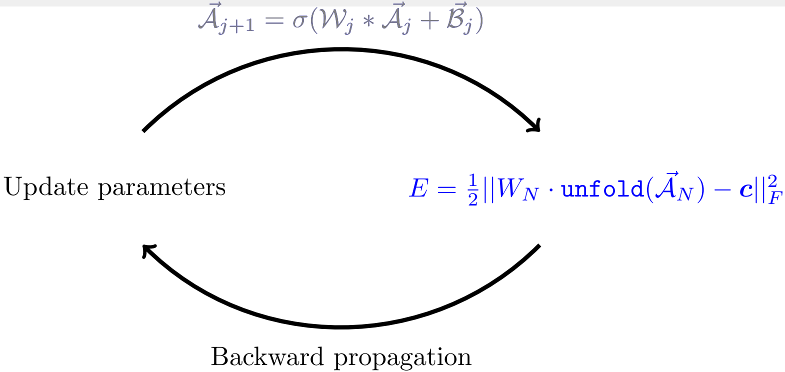
Update parameters

Objective function

Backward propagation

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

Tensor Neural Networks (tNNs)



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

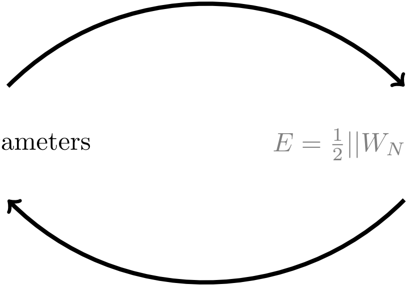
Update parameters

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

Backward propagation

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

Tensor Neural Networks (tNNs)



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

Update parameters

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

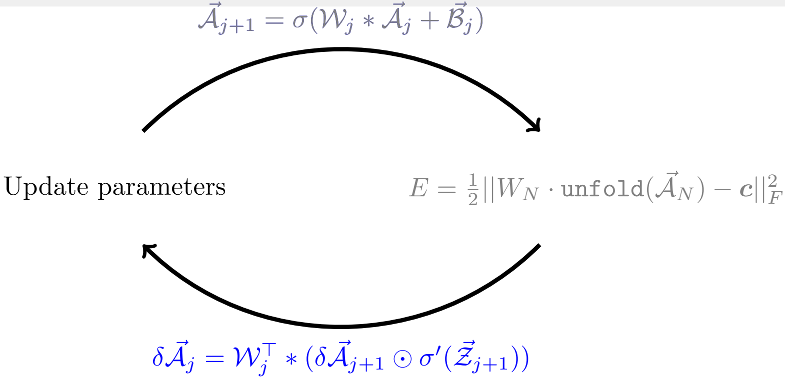
$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top * (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

where $\vec{\mathcal{Z}}_{j+1} = \mathcal{W}_j * \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j$ 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)



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

Update parameters

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

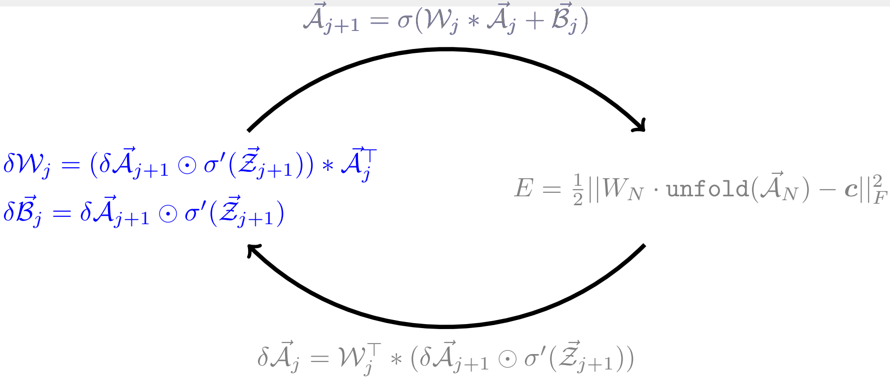
$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top * (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

where $\vec{\mathcal{Z}}_{j+1} = \mathcal{W}_j * \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j$ 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)



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

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

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

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

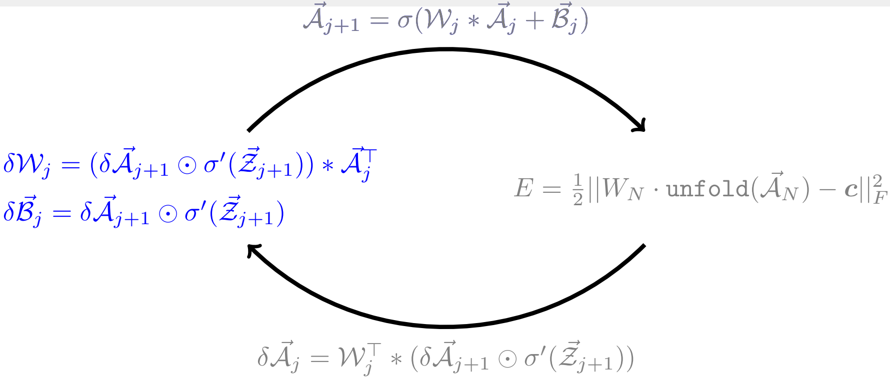
$$\delta\vec{\mathcal{A}}_j = \mathcal{W}_j^\top * (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}))$$

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

---

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

Tensor Neural Networks (tNNs)



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

$$\begin{aligned} \delta\mathcal{W}_j &= (\delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1})) * \vec{\mathcal{A}}_j^\top \\ \delta\vec{\mathcal{B}}_j &= \delta\vec{\mathcal{A}}_{j+1} \odot \sigma'(\vec{\mathcal{Z}}_{j+1}) \end{aligned}$$

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

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

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

Update parameters = Gradient descent!

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

Mimetic Structure

- The **update relations** are **analogous** to their matrix counterparts by **no coincidence**
- In the **M-product** framework, tensors are **M-linear** operators just as **matrices** are **linear** operators

A Dynamic Perspective on Neural Networks

Consider a **residual network** matrix **forward propagation** scheme:

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

This is a **forward Euler** discretization of the continuous system:

$$\dot{\boldsymbol{a}}(t) = \sigma(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t)) \text{ for } t \in [0, T]$$

Haber and Ruthotto, **Inverse Problems**, 2017

He, Zhang, and Ren, **CVPR**, 2015

A Dynamic Perspective on Neural Networks

Consider a **residual network** matrix **forward propagation** scheme:

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

This is a **forward Euler** discretization of the continuous system:

$$\dot{\boldsymbol{a}}(t) = \sigma(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t)) \text{ for } t \in [0, T]$$

Network layers are discrete steps in time!

Haber and Ruthotto, **Inverse Problems**, 2017

He, Zhang, and Ren, **CVPR**, 2015

A Dynamic Perspective on Neural Networks

Consider a **residual network** matrix **forward propagation** scheme:

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

This is a **forward Euler** discretization of the continuous system:

$$\dot{\boldsymbol{a}}(t) = \sigma(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t)) \text{ for } t \in [0, T]$$

Network layers are discrete steps in time!

Well-posed learning problem

- Forward propagation is **stable**. Converge to a solution
- **Classification** function **depends continuously** on **initialization** of parameters. Distinctions remain distinct

---

Haber and Ruthotto, **Inverse Problems**, 2017

He, Zhang, and Ren, **CVPR**, 2015

Trainable Networks - Tensor Formulation

In the continuous case, $\dot{\boldsymbol{a}}(t) = \sigma(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t))$, **stability** depends on the **eigenvalues of the Jacobian:**

$$J(t) = W(t)^\top \cdot \texttt{diag}(\sigma'(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t)))$$

Haber and Ruthotto, **Inverse Problems**, 2017

Trainable Networks - Tensor Formulation

In the continuous case, $\dot{\boldsymbol{a}}(t) = \sigma(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t))$, **stability** depends on the **eigenvalues of the Jacobian:**

$$J(t) = W(t)^{\top} \cdot \texttt{diag}(\sigma'(W(t) \cdot \boldsymbol{a}(t) + \boldsymbol{b}(t)))$$

Well-posed Learning Problem

$\max_i \mathrm{Re}(\lambda_i(W(t))) \leq 0 \implies$ stable forward propagation

$\max_i \mathrm{Re}(\lambda_i(W(t))) \approx 0 \implies$ distinctions remain distinct

---

Haber and Ruthotto, **Inverse Problems**, 2017

Trainable Networks - Tensor Formulation

In the continuous case, $\dot{\vec{\mathcal{A}}}(t) = \sigma(\mathcal{W}(t) * \vec{\mathcal{A}}(t) + \vec{\mathcal{B}}(t))$, stability depends on the **eigenvalues of the Jacobian:**

$$J(t) = \texttt{bcirc}(\mathcal{W}(t))^\top \cdot \texttt{diag}(\sigma'(\texttt{unfold}(\mathcal{W}(t) * \vec{\mathcal{A}}(t) + \vec{\mathcal{B}}(t))))$$

Well-posed Learning Problem

$\max_i \mathrm{Re}(\lambda_i(\texttt{bcirc}(\mathcal{W}(t)))) \leq 0 \implies$ **stable** forward propagation

$\max_i \mathrm{Re}(\lambda_i(\texttt{bcirc}(\mathcal{W}(t)))) \approx 0 \implies$ **distinctions** remain distinct

---

Haber and Ruthotto, **Inverse Problems**, 2017

Trainable Networks - Tensor Formulation

In the continuous case, $\dot{\vec{\mathcal{A}}}(t) = \sigma(\mathcal{W}(t) * \vec{\mathcal{A}}(t) + \vec{\mathcal{B}}(t))$, stability depends on the **eigenvalues of the Jacobian:**

$$J(t) = \texttt{bcirc}(\mathcal{W}(t))^\top \cdot \texttt{diag}(\sigma'(\texttt{unfold}(\mathcal{W}(t) * \vec{\mathcal{A}}(t) + \vec{\mathcal{B}}(t))))$$

Well-posed Learning Problem

$\max_i \mathrm{Re}(\lambda_i(\texttt{bcirc}(\mathcal{W}(t)))) \leq 0 \Longrightarrow$ **stable** forward propagation

$\max_i \mathrm{Re}(\lambda_i(\texttt{bcirc}(\mathcal{W}(t)))) \approx 0 \Longrightarrow$ **distinctions** remain distinct

Implement stable forward propagation scheme which ensures well-posedness!

Haber and Ruthotto, **Inverse Problems**, 2017

A Hamiltonian-Inspired Framework

Definition (Hamiltonian)

A system $H(\boldsymbol{a}(t), \boldsymbol{z}(t))$ which satisfies $\dot{\boldsymbol{a}}(t) = \nabla_{\boldsymbol{z}} H$ and $\dot{\boldsymbol{z}}(t) = -\nabla_{\boldsymbol{a}} H$

Physical Intuition: $\boldsymbol{a}$ = position, $\boldsymbol{z}$ = velocity/momentum

$$H(\boldsymbol{a}(t), \boldsymbol{z}(t)) = \underbrace{\tfrac{1}{2}\boldsymbol{z}(t)^{\top} \cdot \boldsymbol{z}(t)}_{\text{kinetic}} + \underbrace{U(\boldsymbol{a}(t))}_{\text{potential}}$$

Properties:

Time reversibility → backward propagation

Energy conservation → stable forward propagation

Volume preservation → distinctions remain distinct

Seamless Matrix to Tensor Reformulation of Complex Architectures

Consider the symmetrized, Hamiltonian-inspired system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{a}(t) \\ \boldsymbol{z}(t) \end{bmatrix} = \sigma\left(\begin{bmatrix} 0 & W(t) \\ -W(t)^\top & 0 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{a}(t) \\ \boldsymbol{z}(t) \end{bmatrix} + \begin{bmatrix} -\boldsymbol{b}(t) \\ \boldsymbol{b}(t) \end{bmatrix}\right)$$

The system is antisymmetric and hence inherently stable

E. Haber, L. Ruthotto, **Stable architectures for deep neural networks**, Inverse Problems, 2017

L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, arxiv 1811.06569, 2018

Seamless Matrix to Tensor Reformulation of Complex Architectures

Consider the symmetrized, Hamiltonian-inspired system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{a}(t) \\ \boldsymbol{z}(t) \end{bmatrix} = \sigma\left(\begin{bmatrix} 0 & W(t) \\ -W(t)^\top & 0 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{a}(t) \\ \boldsymbol{z}(t) \end{bmatrix} + \begin{bmatrix} -\boldsymbol{b}(t) \\ \boldsymbol{b}(t) \end{bmatrix}\right)$$

The system is antisymmetric and hence inherently stable

We discretize with leapfrog integration which is stable for purely imaginary eigenvalues:

$$\begin{aligned} \boldsymbol{z}_{j+\frac{1}{2}} &= \boldsymbol{z}_{j-\frac{1}{2}} - h\,\sigma(W_j^\top \cdot \boldsymbol{a}_j + \boldsymbol{b}_j), \\ \boldsymbol{a}_{j+1} &= \boldsymbol{a}_j + h\,\sigma(W_j \cdot \boldsymbol{z}_{j+\frac{1}{2}} + \boldsymbol{b}_j) \end{aligned}$$

E. Haber, L. Ruthotto, **Stable architectures for deep neural networks**, Inverse Problems, 2017

L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, arxiv 1811.06569, 2018

Seamless Matrix to Tensor Reformulation of Complex Architectures

Consider the symmetrized, Hamiltonian-inspired system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \vec{\mathcal{A}}(t) \\ \vec{\mathcal{Z}}(t) \end{bmatrix} = \sigma\left(\begin{bmatrix} 0 & \mathcal{W}(t) \\ -\mathcal{W}(t)^\top & 0 \end{bmatrix} * \begin{bmatrix} \vec{\mathcal{A}}(t) \\ \vec{\mathcal{Z}}(t) \end{bmatrix} + \begin{bmatrix} -\vec{\mathcal{B}}(t) \\ \vec{\mathcal{B}}(t) \end{bmatrix}\right)$$

The system is antisymmetric and hence inherently stable

We discretize with leapfrog integration which is stable for purely imaginary eigenvalues:

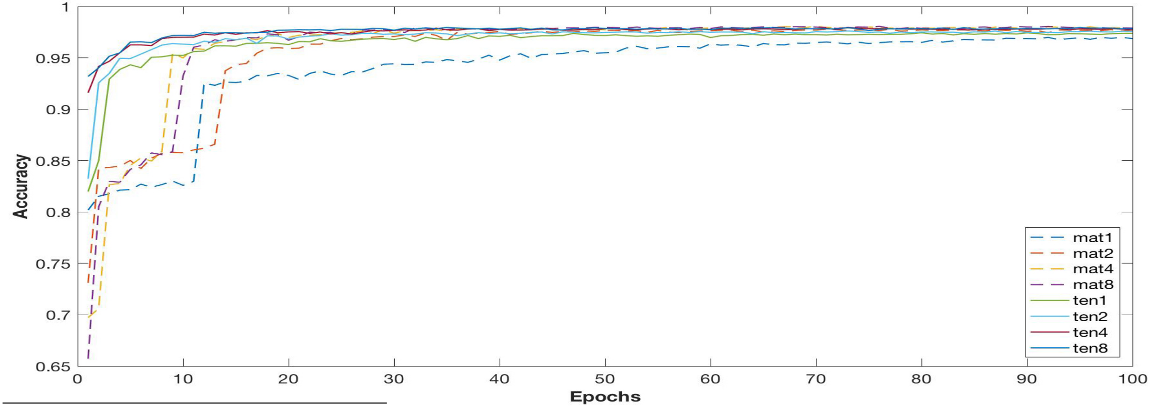
$$\begin{aligned} \vec{\mathcal{Z}}_{j+\frac{1}{2}} &= \vec{\mathcal{Z}}_{j-\frac{1}{2}} - h\,\sigma(\mathcal{W}_j^\top * \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j), \\ \vec{\mathcal{A}}_{j+1} &= \vec{\mathcal{A}}_j + h\,\sigma(\mathcal{W}_j * \vec{\mathcal{Z}}_{j+\frac{1}{2}} + \vec{\mathcal{B}}_j) \end{aligned}$$

E. Haber, L. Ruthotto, **Stable architectures for deep neural networks**, Inverse Problems, 2017

L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, arxiv 1811.06569, 2018

Tensor vs. Matrix Learning: MNIST Database Results

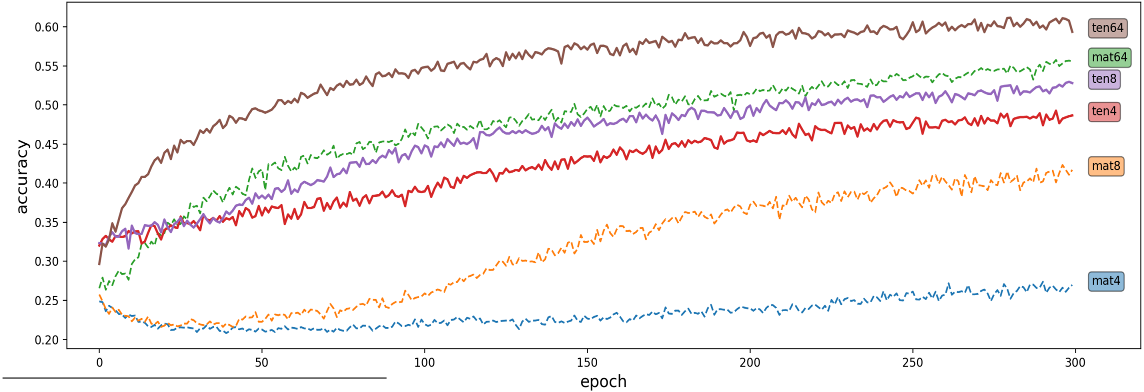
Data: 28×28 grayscale images of handwritten digits, 60000 train, 10000 test
Fixed parameters: $h = 0.1$, $\alpha = 0.1$, $\sigma = tanh$, batch size = 20, 100 epochs
Learnable parameters: matrix - $28^4 N + 28^2 N$, tensor - $28^3 N + 28^2 N$



L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, 2018, `https://arxiv.org/abs/1811.06569`

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-$(3^2 \cdot 32^3)N + 3 \cdot 32^2 N$

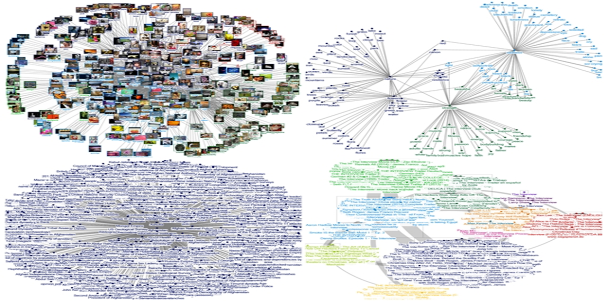


A. Krizhevsky, **Learning multiple layers of features from tiny images**, 2009

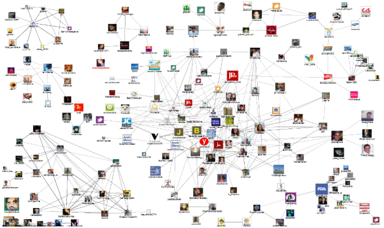
L. Newman, L. Horesh, H. Avron, M. Kilmer, **Stable tensor neural networks for rapid deep learning**, arxiv 1811.06569, 2018

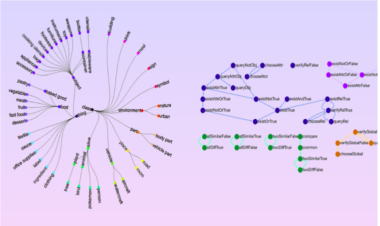
Dynamic Graphs

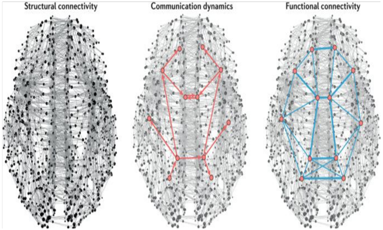
- Graphs are ubiquitous data structures - represent interactions and structural relationships
- In many real-world applications, underlying graph changes over time
- Learning representations of dynamic graphs is essential

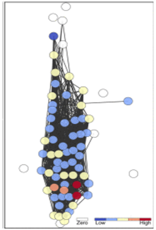


Dynamic Graphs - Applications





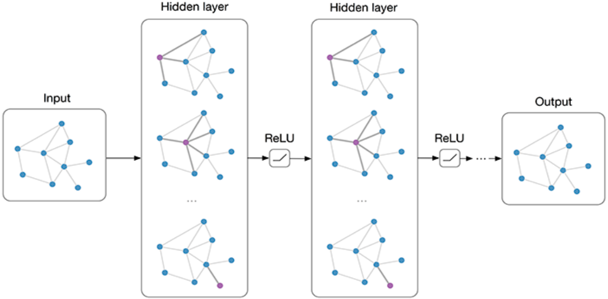




Corporate/financial networks, Natural Language Understanding (NLU), Social networks, Neural activity networks, Traffic predictions.

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 $\boldsymbol{x}$ and $\boldsymbol{y}$:

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

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

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

$$\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{U}(\boldsymbol{U}^\top\boldsymbol{x} \odot \boldsymbol{U}^\top\boldsymbol{y})$$

- Filtered convolution:

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

  with matrix filter function $h(\boldsymbol{L}) = \boldsymbol{U}h(\Lambda)\boldsymbol{U}^\top$

Graph Convolutional Neural Networks

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

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

  h_θ filter function parametrized by θ, σ a nonlinear function (e.g., RELU), and $\boldsymbol{W}^{(i)}$ a weight matrix with $\boldsymbol{H}_0 = \boldsymbol{X}$
- Defferrard et al., (2016) used Chebyshev approximation $T_{m+1}(\boldsymbol{L}) = 2\boldsymbol{L}T_m(\boldsymbol{L}) - T_{m-1}(\boldsymbol{L})$:

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

- *GCN* (Kipf & Welling, 2016): Each layer takes form: $\sigma(\boldsymbol{L}\boldsymbol{X}\boldsymbol{W})$
- 2-layer example:

$$\boldsymbol{Z} = \text{softmax}(\boldsymbol{L}\,\sigma(\boldsymbol{L}\boldsymbol{X}\boldsymbol{W}^{(0)})\,\boldsymbol{W}^{(1)})$$

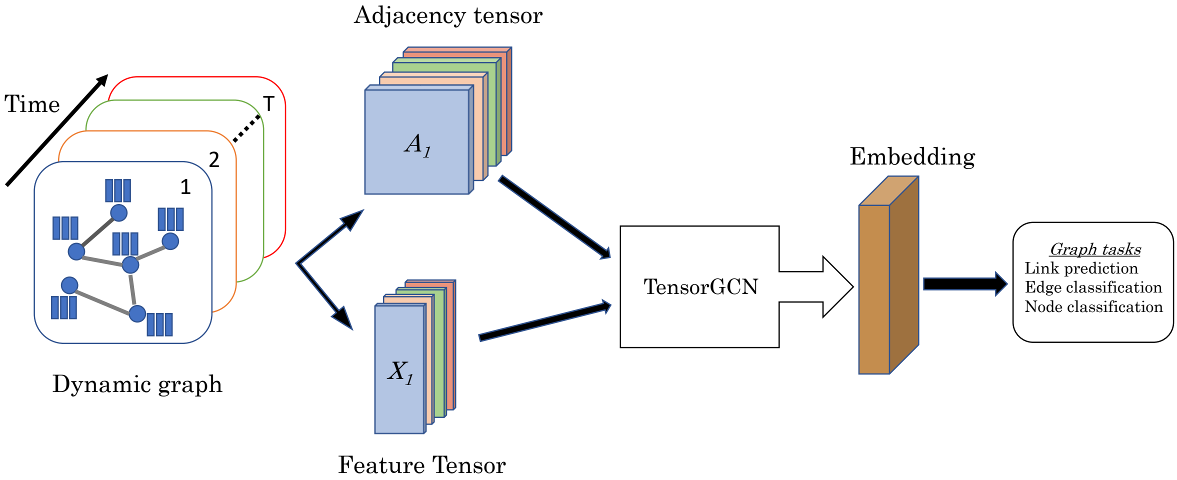
GCN for Dynamic Graphs

- 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
- **How ?**

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
- **How ?** Use a tensor-tensor framework!
- T adjacency matrices $\boldsymbol{A}_{::t} \in \mathbb{R}^{N \times N}$ stacked into tensor $\mathcal{A} \in \mathbb{R}^{N \times N \times T}$
- T node feature matrices $\boldsymbol{X}_{::t} \in \mathbb{R}^{N \times F}$ stacked into tensor $\mathcal{X} \in \mathbb{R}^{N \times F \times T}$

TM-GCN


Adjacency tensor
Time
T
2
1
A_1
Embedding
TensorGCN
Graph tasks
Link prediction
Edge classification
Node classification
X_1
Dynamic graph
Feature Tensor

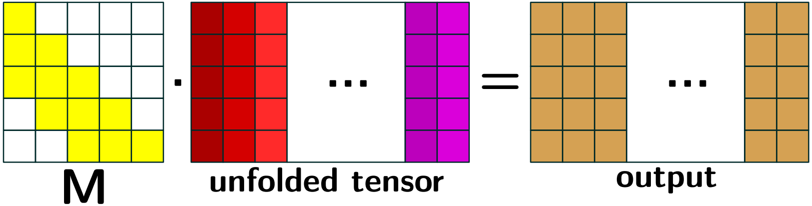
TM-GCN

- 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:

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

- We choose $\boldsymbol{M}$ to be lower triangular and banded (causal):

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



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

Theoretical Motivation

- The tensor $\mathcal{A}$ has an eigendecomposition $\mathcal{A} = \mathcal{Q} \star \mathcal{D} \star \mathcal{Q}^\top$.
- *Filtering:* Given a signal $\mathcal{X} \in \mathbb{R}^{N \times 1 \times T}$ and a function $g : \mathbb{R}^{1 \times 1 \times T} \to \mathbb{R}^{1 \times 1 \times T}$, we define the *tensor spectral graph filtering* of $\mathcal{X}$ with respect to g as

  $$\mathcal{X}_{\text{filt}} = \mathcal{Q} \star g(\mathcal{D}) \star \mathcal{Q}^\top \star \mathcal{X},$$

  where

  $$g(\mathcal{D})_{mn:} = \begin{cases} g(\mathcal{D}_{mn:}) & \text{if } m = n, \\ 0 & \text{if } m \neq n. \end{cases}$$

- Suppose g satisfies above. For any $\varepsilon > 0$, there exists an integer K and a set $\{\theta^{(k)}\}_{k=1}^{K} \subset \mathbb{R}^{1 \times 1 \times T}$ such that

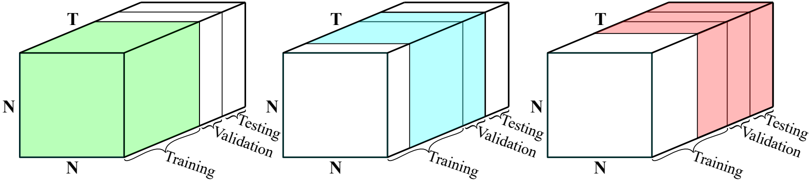
  $$\left\| g(\mathcal{D}) - \sum_{k=0}^{K} \mathcal{D}^{\star k} \star \theta^{(k)} \right\| < \varepsilon, \tag{1}$$

  where $\| \cdot \|$ is the tensor Frobenius norm, and where $\mathcal{D}^{\star k} = \mathcal{D} \star \cdots \star \mathcal{D}$ is the M-product of k instances of $\mathcal{D}$, with the convention that $\mathcal{D}^{\star 0} = \mathcal{J}$

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}
SBM	1,000	1,601,999	50	–	–	35	5	10
BitcoinOTC	6,005	35,569	135	14	2	95	20	20
BitcoinAlpha	7,604	24,173	135	14	2	95	20	20
Reddit	3,818	163,008	86	14	2	66	10	10
Chess	7,301	64,958	100	31	3	80	10	10



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

TM-GCN - Edge Classification Results

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

	Dataset			
Method	Bitcoin OTC	Bitcoin Alpha	Reddit	Chess
WD-GCN	0.3562	0.2533	**0.2337**	0.4311
EvolveGCN	0.3483	0.2273	0.2012	0.4351
GCN	0.3402	0.2381	0.1968	0.4342
TM-GCN - M1	0.3660	**0.3243**	0.2057	**0.4708**
TM-GCN - M2	**0.4361**	0.2466	0.1833	0.4513

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

TM-GCN - Link Prediction Results

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

	Dataset				
Method	SBM	Bitcoin OTC	Bitcoin Alpha	Reddit	Chess
WD-GCN	0.9436	0.8071	0.8795	**0.3896**	0.1279
EvolveGCN	0.7620	0.6985	0.7722	0.2866	0.0915
GCN	0.9201	0.6847	0.7655	0.3099	0.0899
TM-GCN - M1	0.9684	0.8026	0.9318	0.2270	**0.1882**
TM-GCN - M2	**0.9799**	**0.8458**	**0.9631**	0.1405	0.1514

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

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

Questions?