CSE 392: Matrix and Tensor Algorithms for Data Spring 2024

Lecture 23 — 2024.04.11

Instructor: Shashanka Ubaru *Scribe: Xuxi Chen*

1 Revisiting Randomized t-SVD

1.1 t-SVD

Theorem: For any $\mathcal{A} \in \mathbb{R}^{m \times l \times n}$, there exists a full tensor-SVD such that

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

with an $m \times m \times n$ orthogonal tensor $\mathcal{U}$, an $l \times l \times n$ orthogonal tensor $\mathcal{V}$, and an $m \times l \times n$ f-diagonal tensor $\mathcal{S}$ ordered such that the singular tubes $s_i = S_{i,i,:}$ having $\|s_1\|_F^2 \geq \|s_2\|_F^2 \geq \cdots$. The **t-rank** is the number of non-zero tube-fibers in $\mathcal{S}$.

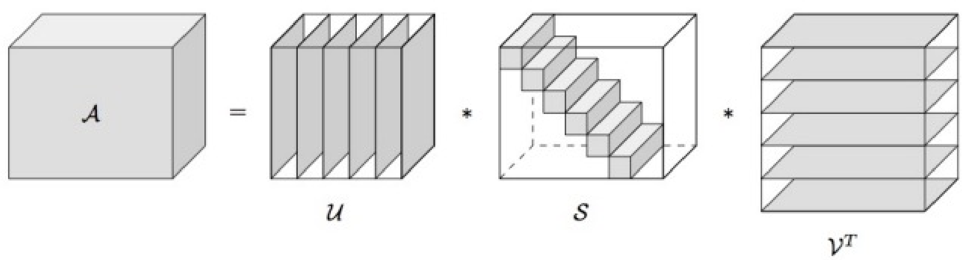

Figure 1: Demonstration of tensor-SVD.

1.2 t-SVD computation

The t-SVD can be computed efficiently in parallel by moving to the Fourier domain in the following steps:

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

1.3 Tensor-tensor SVDs

Theorem (Kilmer, Horesh, Avron, Newman): Let $\mathcal{A}$ be an $m \times p \times n$ tensor and $\mathcal{M}$ a non-zero multiple of a unitary/orthogonal matrix. The (full) $\star_M$ tensor SVD (t-SVDM) is

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

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

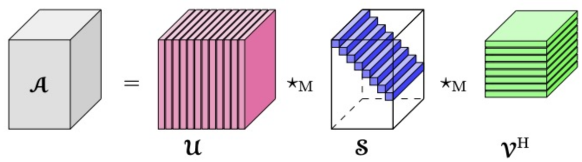

Figure 2: Demonstration of tensor-tensor SVDs

1.4 Practical Algorithm for t-SVDM

- $\hat{\mathcal{A}} \leftarrow \mathcal{A} \times_M \mathbf{M}$
- $\left[\hat{\mathcal{U}}_{:,:,i,} \hat{\mathcal{S}}_{:,:,i,} \hat{\mathcal{V}}_{:,:,i,}\right] = \text{SVD}(\hat{\mathcal{A}}_{:,:,i,})$, for $i = 1, \dots, n$
- $\mathcal{U} = \hat{\mathcal{U}} \times_3 \mathbf{M}^{-1}, \mathcal{S} = \hat{\mathcal{S}} \times_3 \mathbf{M}^{-1}, \mathcal{V} = \hat{\mathcal{V}} \times_3 \mathbf{M}^{-1}$

1.5 Randomized Variants

Randomized t-SVD with Subspace-type Iteration The randomized t-SVD algorithm is described as follows:

Input: $\mathcal{A} \in \mathbb{R}^{m \times l \times n}$, 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 $\mathcal{V}_k \in \mathbb{R}^{l \times k \times n}$.

Steps:

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

In practice, this algorithm can be implemented in the transformed domain with parallel matrix computations.

1.6 Analysis: Expectation of Error

Theorem. The output satisfies

$$\mathbb{E}\|\mathcal{A} - \mathcal{Q} * \mathcal{Q}^T * \mathcal{A}\| \leq \mathbb{E}\|\mathcal{A} - \mathcal{Q} * \mathcal{B}_k\|^2 \leq \frac{1}{n}(\sum_{i=1}^{n}(1 + \frac{k(\tau_k^{(i)})^{4q_i}}{p-1})(\sum_{j>k}(\hat{\sigma}_j^{(i)})^2)),$$

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

1.7 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

Figure 3: Recognition Rate

2 t-product applications

2.1 Application: Facial Recognition

A typical application is to conduct facial recognition. The algorithm is listed below:

- $\vec{\mathcal{X}}_j$, $j = 1, 2, \ldots, m$ are the training images;
- $\vec{\mathcal{Y}}$ is the mean image;
- $\vec{\mathcal{A}}_j = \vec{\mathcal{X}}_j - \vec{\mathcal{Y}}$ are 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 matrix $\mathcal{U}$ contains the principal components, so

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

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

Matching Coefficients. We keep the basis $\mathcal{U}_{:,1:k,:}$ and the tensor coefficients $\mathcal{U}^\top_{:,1:k,:} * \vec{\mathcal{A}}_j$. When a new mean subtracted image, oriented as a tensor $\mathcal{B}$ comes in, we compute its tensor coefficients $\mathcal{U}^T_{:,1:k,:} * \vec{\mathcal{B}}$. 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".

Figure 4

Facial Recognition Task Take 256 image subset (4 people, 64 different lighting conditions) and randomly removed 1 image per person. The Extended Yale Face Database B can be access at `http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html`. The image $\mathcal{A}$ is $192 \times 252 \times 128$. We truncate the images in eigenspaces to $k = 15$. The error is $\frac{\mathcal{A}-\hat{\mathcal{A}}}{\mathcal{A}} = .115$.

This means that

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

so the j-th lateral slice, *i.e.* a mean-subtracted image, is $\mathcal{A}_{:,j,:} = \sum_{i=1}^{k} \mathcal{U}_{:,i,:} * c_{i,j}$.

Facial Recognition Task (when M is a DFT matrix)

- 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

Figure 5: Examples of Facial Recognition Datasets

t-SVDII versus PCA Figure 6 demonstrates the performance comparison between t-SVDII and PCA.

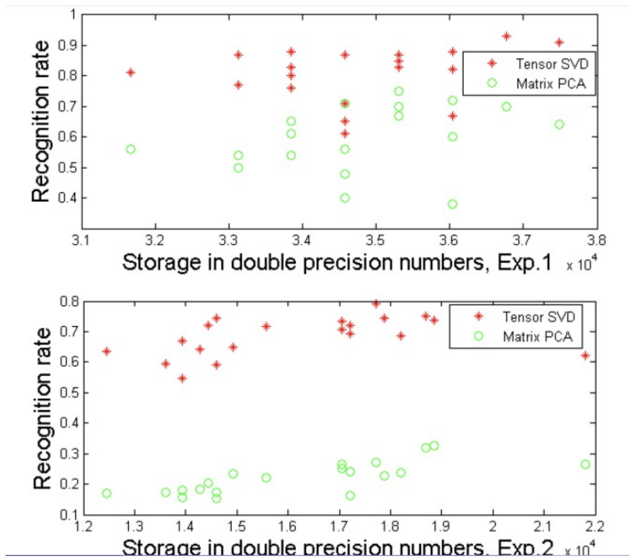

Figure 6: Performance comparison between t-SVDII versus PCA.

Performance on the Yale Faces Dataset Figure 7 demonstrates the results on the Yale Faces Dataset.

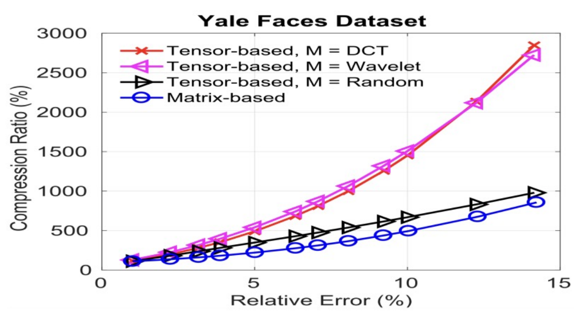

Figure 7: Performance on the Yale Faces Dataset

Hyperspectral Results Best performance are points lying closest to the upper left, *i.e.*, the most compression for the smallest relative error (shown in Figure 8).

Numerical Results Figure 9 demonstrates the performance of hyperspectral compression.

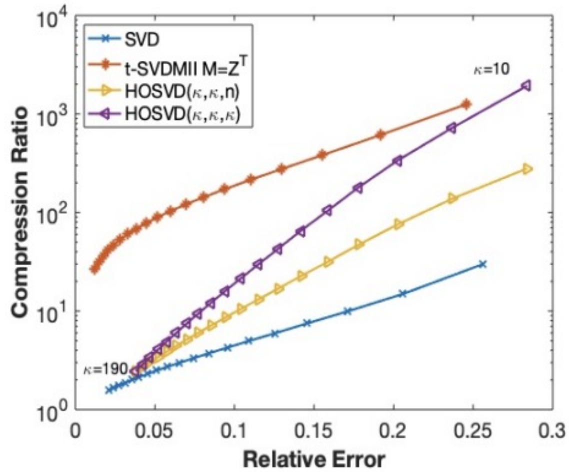

Figure 8: Hyperspectral compression ratio versus relative error.

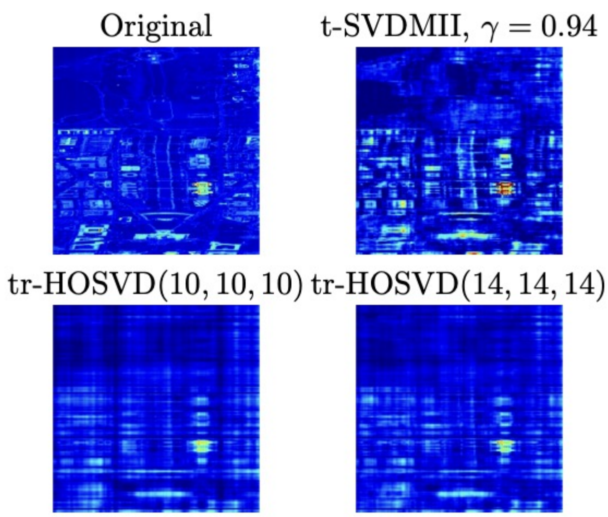

Figure 9: Approximation of hyperspectral wavelength 10 corresponding to upper right of graph.

2.2 Neural Networks, Hypothetically

Let a_0 be a feature vector with an associated target vector c. Let f be a function which propagates a_0 though connected layers:

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

where σ is a nonlinear, monotonic activation function.

Goal: Learn the function f which optimizes:

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

where $\mathcal{H}$ is a hypothesis space of functions.

Less is More: Reduced Parameterization In Figure 10, we can see why tensors can help reduce parameterization.

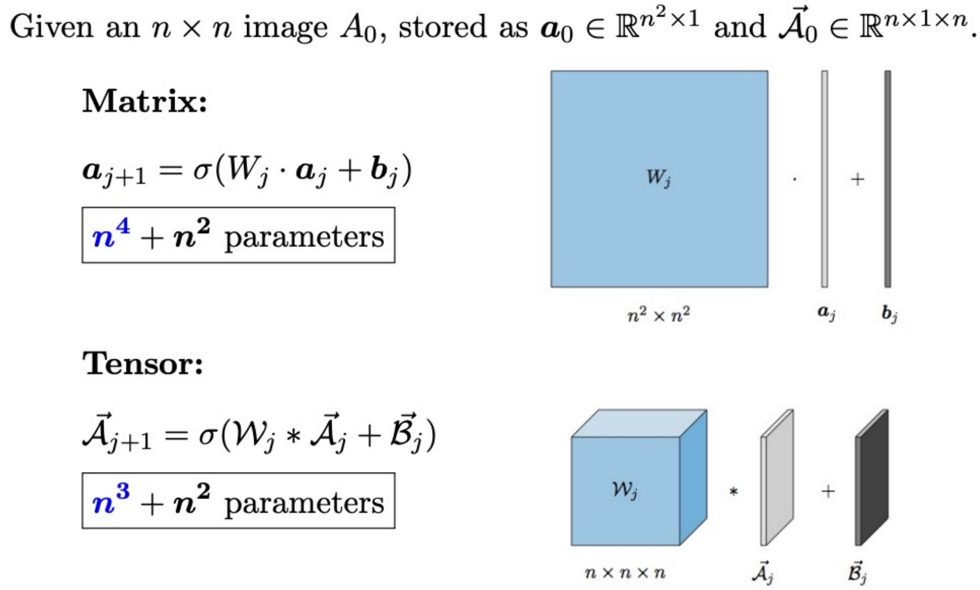

Figure 10: The mechanism of using tensors in neural networks.

Tensor Neural Networks (tNNs) To update parameters, we can use gradient descent methods, as demonstrated in Figure 11

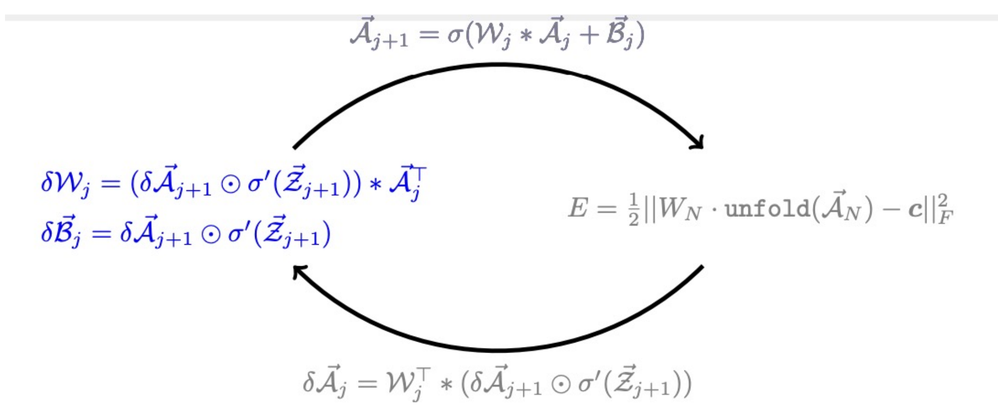

Figure 11: Gradient descent methods for tensor neural networks.

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:

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

This is a forward Euler discretization of the continuous system:

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

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

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

This is a well-posed learning problem:

- $\max_i Re(\lambda_i(W(t))) \leq 0 \Longrightarrow$ Stable forward propagation
- $\max_i Re(\lambda_i(W(t))) \approx 0 \Longrightarrow$ Distinctions remain distinct

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

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

This is again a well-posed learning problem:

- $\max_i Re(\lambda_i(\text{bcirc}(\mathcal{W}(t)))) \leq 0 \Longrightarrow$ Stable forward propagation
- $\max_i Re(\lambda_i(\text{bcirc}(\mathcal{W}(t)))) \approx 0 \Longrightarrow$ Distinctions remain distinct

Implement stable forward propagation scheme which ensures well-posedness!

A Hamiltonian-Inspired Framework **Definition of Hamiltonian:** A system $H(a(t), z(t))$ which satisfies $\dot{a}(t) = \nabla_z H$ and $\dot{z}(t) = -\nabla_z H$.

Physical Intuition: a = position, z = velocity/momentum

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

Properties:

- Time reversibility $\to$ Backward propagation
- Energy conservation $\to$ Stable forward propagation
- Volume preservation $\to$ Distinctions remain distinct

Seamless Matrix to Tensor Reformulation of Complex Architectures Consider the symmetrized, Hamiltonian-inspired system:

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

The system is antisymmetric and hence inherently stable. We discretize with leapfrog integration which is stable for purely imaginary eigenvalues:

$$z_{j+\frac{1}{2}} = z_{j-\frac{1}{2}} - h\sigma(W_j^T \cdot a_j + b_j),$$
$$a_{j+1} = a_j + h\sigma(W_j^T \cdot z_{j+\frac{1}{2}} + b_j).$$

Consider the symmetrized, Hamiltonian-inspired system:

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

The system is antisymmetric and hence inherently stable. We discretize with leapfrog integration which is stable for purely imaginary eigenvalues:

$$\vec{\mathcal{Z}}_{j+\frac{1}{2}} = \vec{\mathcal{Z}}_{j-\frac{1}{2}} - h\sigma(\mathcal{W}_j^T \cdot \vec{\mathcal{A}}_j + \vec{\mathcal{B}}_j),$$
$$\vec{\mathcal{A}}_{j+1} = \vec{\mathcal{A}}_j + h\sigma(W_j^T \cdot \vec{\mathcal{Z}}_{j+\frac{1}{2}} + \vec{\mathcal{B}}_j).$$

Tensor versus Matrix Learning: MNIST Database Results. Details:

Data: 28×28 grayscale images of handwritten digits having 60000 train images and 10000 test images.

Fixed parameters: $h = 0.1$, $\alpha = 0.1$, $\sigma = \tanh$, batch size = 20, training for 100 epochs.

Learnable parameters: matrix - $28^4 N + 28^2 N$, tensor - $28^3 N + 28^2 N$

Tensor vs. Matrix Learning: CIFAR-10 Database Results Details:

Data: $32 \times 32 \times 3$ RGB images from 10 classes, 50000 training images, 10000 test images.

Fixed parameters: $h = 0.1$, $\alpha = 0.01$, $\sigma = \tanh$, batch size = 100, 300 epochs, M = DCT matrix.

Learnable parameters: matrix - $(3^2 \cdot 32^4)N + 3 \cdot 32^2 N$, tensor - $(3^2 \cdot 32^4)N + 3 \cdot 32^2 N$

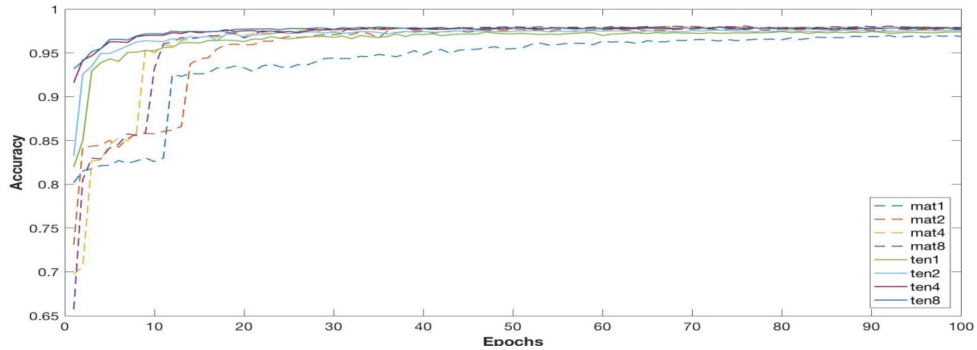

Figure 12: MNIST results.

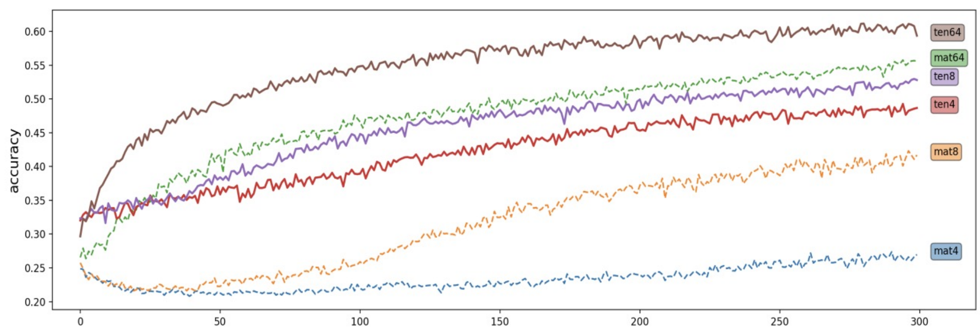

Figure 13: CIFAR-10 results.

2.3 Dynamic Graphs

The character of 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

Figure 14 demonstrates some examples of dynamic graphs.

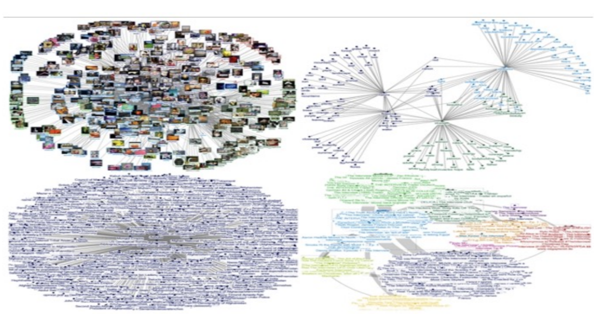

Figure 14: Dynamic graph examples.

Dynamic Graphs - Applications Figure 15 shows some examples of the applications of dynamic graphs.

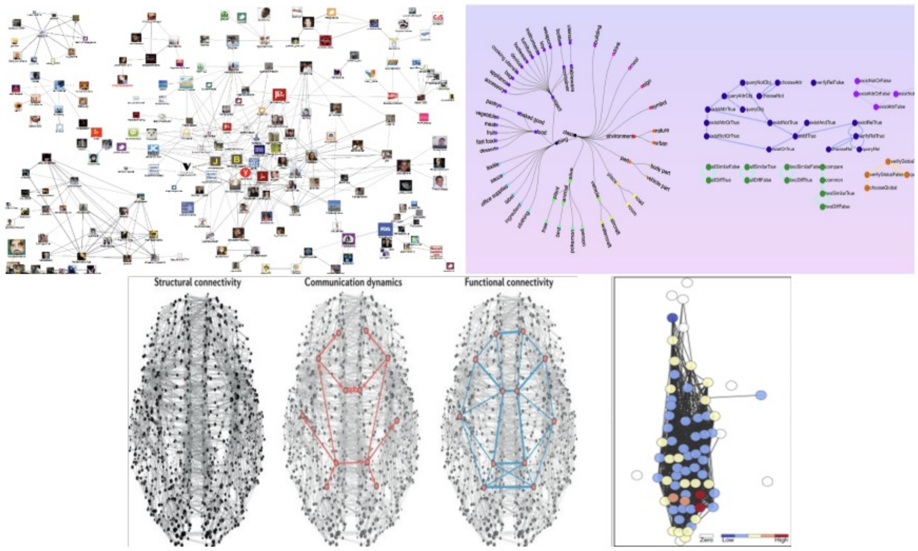

Figure 15: Corporate/financial networks, Natural Language Understanding (NLU), Social networks, Neural activity networks, and Traffic predictions.

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

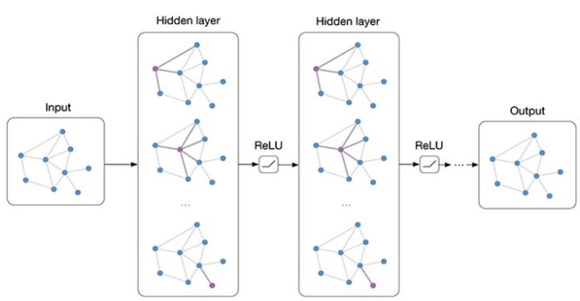

Figure 16: Image by (Kipf & Welling, 2016).

Motivation:

- Convolution of two signals x and y:

$$x \otimes y = F^{-1}(Fx \otimes Fy),$$

where F is a Fourier transform (DFT matrix)

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

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

- Filtered convolution:

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

with a matrix filter function $h(L) = Uh(\Lambda)U^T$.

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

$$H_{i+1} = \sigma(h_\theta(L)H_i W^{(i)},$$

where h_θ is a filter function parametrized by θ and σ, a nonlinear function (e.g., ReLU), and $W_{(i)}$ a weight matrix with $H_0 = X$.

- Defferrard et al. (2016) used Chebyshev approximation $T_{m+1}(L) = 2LT_m(L) - T_{m-1}(L)$:

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

- GCN (Kipf & Welling, 2016): Each layer takes form:

$$\sigma(LXW).$$

A two-layer example:

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

- We use the $\star_M$-Product to extend the standard GCN to dynamic graphs, and can propose a tensor GCN model

$$\sigma(\mathcal{A} \star_M \mathcal{X} \star_M \mathcal{W})$$

- A two-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 M to be lower triangular and banded (causal):

$$M_{tk} = \begin{cases} \frac{1}{\min(b,t)} \text{ or } \frac{1}{k} \text{ if } \max(1, t-b+1) \leq k \leq t, \\ 0 \qquad\qquad \text{otherwise} \end{cases}.$$

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

2.4 Theoretical Motivation

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

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

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

$$\|g(\mathcal{D}) - \sum_{k=0}^{K} \mathcal{D}^{\star k} \star \theta^{(k)}\| \leq \epsilon$$

  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 $\mathcal{D}^{\star 0} = \mathcal{J}$.