Tensor Networks, RNNs and Weighted Automata

Guillaume Rabusseau
Assistant Professor at DIRO, UdeM
CIFAR Canada Chair in AI at Mila

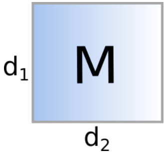
April 14, 2021
ANR TAUDoS - Kickoff

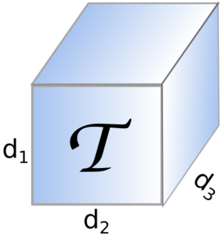
Outline

1. An Introduction to Tensor Networks
2. Weighted Automata Vs. RNNs
3. Tensor Network Models for Sequences
4. A Tensor Network View of the Spectral Learning Algorithm

An Introduction to Tensor Networks

Tensors





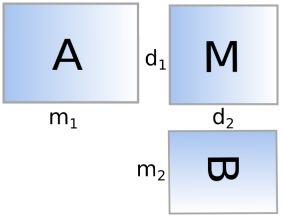
$\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$

$\mathbf{M}_{ij} \in \mathbb{R}$ for $i \in [d_1], j \in [d_2]$

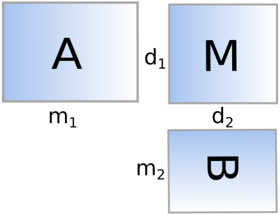
$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

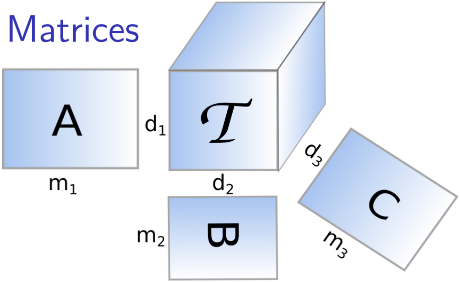
$(\mathcal{T}_{ijk}) \in \mathbb{R}$ for $i \in [d_1], j \in [d_2], k \in [d_3]$

Tensors: Multiplication with Matrices



Tensors: Multiplication with Matrices

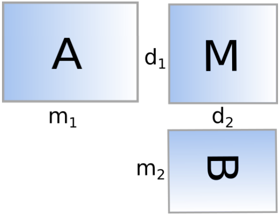


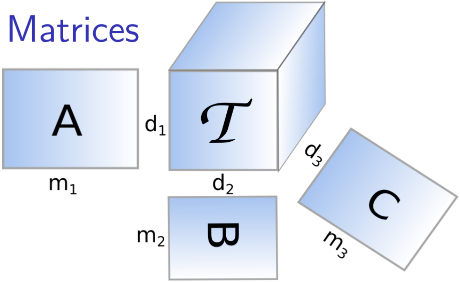


$\mathbf{AMB}^{\top} \in \mathbb{R}^{m_1 \times m_2}$

$\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$

Tensors: Multiplication with Matrices



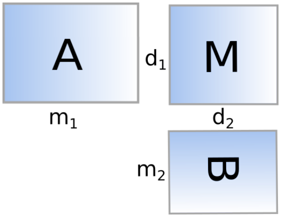


$\mathbf{AMB}^\top \in \mathbb{R}^{m_1 \times m_2}$

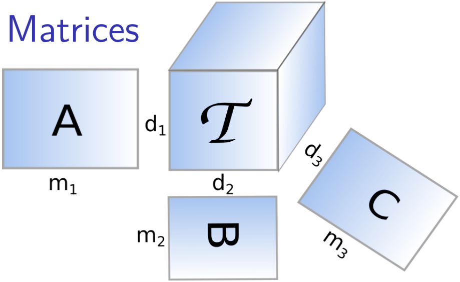
$\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$

ex: If $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and $\mathbf{A} \in \mathbb{R}^{m_1 \times d_1}, \mathbf{B} \in \mathbb{R}^{m_2 \times d_2}, \mathbf{C} \in \mathbb{R}^{m_3 \times d_3}$, then $\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ is defined by

Tensors: Multiplication with Matrices



$\mathbf{AMB}^\top \in \mathbb{R}^{m_1 \times m_2}$



$\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$

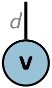
ex: If $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and $\mathbf{A} \in \mathbb{R}^{m_1 \times d_1}$, $\mathbf{B} \in \mathbb{R}^{m_2 \times d_2}$, $\mathbf{C} \in \mathbb{R}^{m_3 \times d_3}$, then $\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ is defined by

$$(\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C})_{i_1,i_2,i_3} = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \sum_{k_3=1}^{n_3} \mathcal{T}_{k_1 k_2 k_3} \mathbf{A}_{i_1 k_1} \mathbf{B}_{i_2 k_2} \mathbf{C}_{i_3 k_3}$$

for all $i_1 \in [d_1], i_2 \in [m_2], i_3 \in [d_3]$.

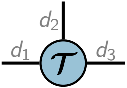
Tensor Networks

Degree of a node $\equiv$ order of tensor



$\mathbf{v} \in \mathbb{R}^d$

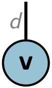
$\mathbf{M} \in \mathbb{R}^{m \times n}$



$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

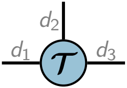
Tensor Networks

Degree of a node $\equiv$ order of tensor



$\mathbf{v} \in \mathbb{R}^{d}$

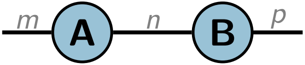
$\mathbf{M} \in \mathbb{R}^{m \times n}$



$\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

Edge $\equiv$ contraction

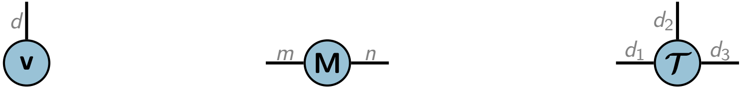
Matrix product:



$(\mathbf{AB})_{i_1,i_2} = \sum_{k=1}^{n} \mathbf{A}_{i_1 k} \mathbf{B}_{k i_2}$

Tensor Networks

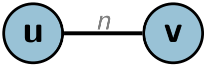
Degree of a node $\equiv$ order of tensor



$\mathbf{v} \in \mathbb{R}^d$ $\qquad$ $\mathbf{M} \in \mathbb{R}^{m \times n}$ $\qquad$ $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

Edge $\equiv$ contraction

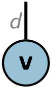
Inner product:



$\mathbf{u}^\top \mathbf{v} = \sum_{k=1}^{n} \mathbf{u}_k \mathbf{v}_k$

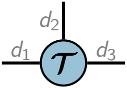
Tensor Networks

Degree of a node $\equiv$ order of tensor



$\mathbf{v} \in \mathbb{R}^d$

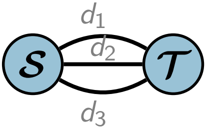
$\mathbf{M} \in \mathbb{R}^{m \times n}$



$\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

Edge $\equiv$ contraction

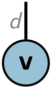
Inner product between tensors:



$$\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}} \rangle = \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} \sum_{i_3=1}^{d_3} \boldsymbol{\mathcal{S}}_{i_1 i_2 i_3} \boldsymbol{\mathcal{T}}_{i_1 i_2 i_3}$$

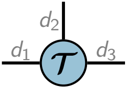
Tensor Networks

Degree of a node ≡ order of tensor



$\mathbf{v} \in \mathbb{R}^d$

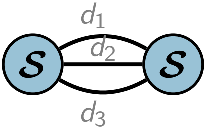
$\mathbf{M} \in \mathbb{R}^{m \times n}$



$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

Edge ≡ contraction

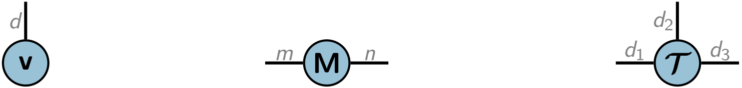
Frobenius norm of a tensor:



$$\|\mathcal{S}\|_F^2 = \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} \sum_{i_3=1}^{d_3} (\mathcal{S}_{i_1 i_2 i_3})^2$$

Tensor Networks

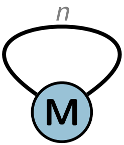
Degree of a node $\equiv$ order of tensor



$\mathbf{v} \in \mathbb{R}^{d}$ $\quad$ $\mathbf{M} \in \mathbb{R}^{m \times n}$ $\quad$ $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

Edge $\equiv$ contraction

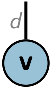
Trace of an $n \times n$ matrix:



$\mathrm{Tr}(\mathbf{M}) = \sum_{i=1}^{n} \mathbf{M}_{ii}$

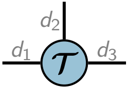
Tensor Networks

Degree of a node $\equiv$ order of tensor



$\mathbf{v} \in \mathbb{R}^d$

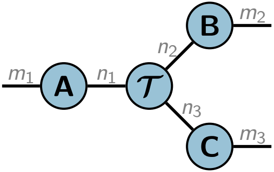
$\mathbf{M} \in \mathbb{R}^{m \times n}$



$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

Edge $\equiv$ contraction

Tensor times matrices:



$$(\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C})_{i_1,i_2,i_3} = \sum_{k_1=1}^{n_1} \sum_{k_2=1}^{n_2} \sum_{k_3=1}^{n_3} \mathcal{T}_{k_1 k_2 k_3} \mathbf{A}_{i_1 k_1} \mathbf{B}_{i_2 k_2} \mathbf{C}_{i_3 k_3}$$

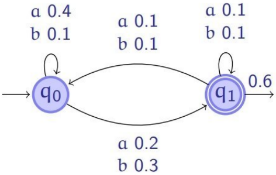
Weighted Automata Vs. RNNs

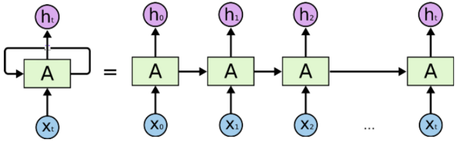
Most of what I will talk about today is based on joint work with Tianyu Li (PhD student) and Doina Precup:

- Rabusseau, Guillaume, Tianyu Li, and Doina Precup. *"Connecting weighted automata and recurrent neural networks through spectral learning."* The 22nd International Conference on Artificial Intelligence and Statistics. PMLR, 2019.
- Li, Tianyu, Doina Precup, and Guillaume Rabusseau. *"Connecting Weighted Automata, Tensor Networks and Recurrent Neural Networks through Spectral Learning."* arXiv preprint arXiv:2010.10029 (2020).

Tianyu Li:

Weighted Automata Vs. Recurrent Neural Networks





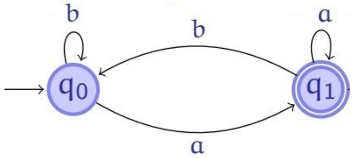
- Weighted automata are "robust" models for sequence data
- Recurrent neural networks can also deal with sequence data
  - ⊕ Remarkably expressive models, impressive results in speech and audio recognition
  - ⊖ Less tractable than WA, limited understanding of their inner working
- Connections between WA and RNN:
  - Can RNN learn regular languages? [Giles et al, 1992], [Avcu et al., 2018]
  - Can we extract finite state machines from RNNs? [Giles et al, 1992], [Weiss et al., 2018], [Ayache et al., 2018]
  - Can we combine FSMs with WA? [Rastogi et al., 2016], [Dyer et al., 2016]
  - To which extent Weighted Automata are linear RNNs?

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. *abba*), λ the empty string.

String Weighted Automata (WA)

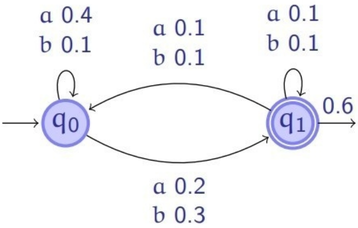
- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. *abba*), λ the empty string.
- Recall: a Deterministic Finite Automaton (DFA) recognizes a *language* (subset of Σ^*).



$\hookrightarrow$ a DFA computes a function $f : \Sigma^* \to \{\top, \bot\}$.

Weighted Automata: States and Weighted Transitions

Example with 2 states and alphabet $\Sigma = \{a, b\}$

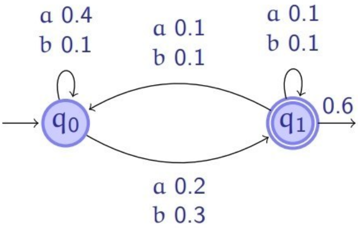


$$f(ab) = 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084$$

slide credits: B. Balle, X. Carreras, A. Quattoni - ENMLP'14 tutorial

Weighted Automata: States and Weighted Transitions

Example with 2 states and alphabet $\Sigma = \{a, b\}$



Operator Representation

$$\boldsymbol{\alpha} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} \quad \mathbf{A}^{a} = \begin{bmatrix} 0.4 & 0.2 \\ 0.1 & 0.1 \end{bmatrix}$$

$$\boldsymbol{\omega} = \begin{bmatrix} 0.0 \\ 0.6 \end{bmatrix} \quad \mathbf{A}^{b} = \begin{bmatrix} 0.1 & 0.3 \\ 0.1 & 0.1 \end{bmatrix}$$

$$f(ab) = 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084$$
$$= \boldsymbol{\alpha}^{\top} \mathbf{A}^{a} \mathbf{A}^{b} \boldsymbol{\omega}$$

slide credits: B. Balle, X. Carreras, A. Quattoni - ENMLP'14 tutorial

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. *abba*)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. *abba*)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$
- Weighted Automaton: $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\omega})$ where

$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector
$\boldsymbol{\omega} \in \mathbb{R}^n$ final weights vector
$\mathbf{A}^\sigma \in \mathbb{R}^{n\times n}$ transition weights matrix for each $\sigma \in \Sigma$

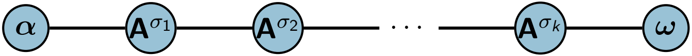
String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. *abba*)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$
- Weighted Automaton: $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\omega})$ where

$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector
$\boldsymbol{\omega} \in \mathbb{R}^n$ final weights vector
$\mathbf{A}^\sigma \in \mathbb{R}^{n\times n}$ transition weights matrix for each $\sigma \in \Sigma$

- A computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by

$$f_A(\sigma_1\sigma_2\cdots\sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1}\mathbf{A}^{\sigma_2}\cdots\mathbf{A}^{\sigma_k}\boldsymbol{\omega}$$



2nd order RNNs

- Recurrent Neural Network (RNN):

$$(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots) \mapsto (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \cdots)$$

- Vanilla RNN:

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1}), \qquad \mathbf{y}_t = g(\mathbf{M}\mathbf{h}_t)$$

2nd order RNNs

- Recurrent Neural Network (RNN):

$$(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots) \mapsto (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \cdots)$$

- Vanilla RNN:

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1}), \qquad \mathbf{y}_t = g(\mathbf{M}\mathbf{h}_t)$$

- Second-order RNN [Giles et al., NIPS'90]:

$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$

$\rightarrow$ order 2 multiplicative interactions: $[\mathbf{h}_t]_i = g\left(\sum_{j,k} \mathcal{W}_{ijk}[\mathbf{x}_t]_j[\mathbf{h}_{t-1}]_k\right)$.

2nd order RNNs

- Recurrent Neural Network (RNN):

$$(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots) \mapsto (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \cdots)$$

- Vanilla RNN:

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1}), \qquad \mathbf{y}_t = g(\mathbf{M}\mathbf{h}_t)$$

- Second-order RNN [Giles et al., NIPS'90]:

$$\mathbf{h}_t = g(\boldsymbol{\mathcal{W}} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$

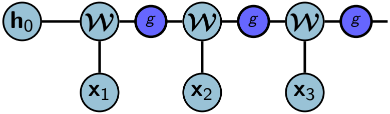
$\rightarrow$ order 2 multiplicative interactions: $[\mathbf{h}_t]_i = g\left(\sum_{j,k} \boldsymbol{\mathcal{W}}_{ijk}[\mathbf{x}_t]_j[\mathbf{h}_{t-1}]_k\right)$.

$\hookrightarrow$ (side note) 2nd order RNN subsume vanilla RNN

Weighted Automata and Recurrent Neural Networks

- The hidden state of a second-order RNN is computed by

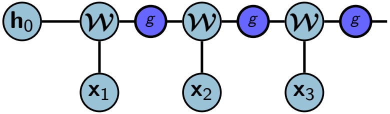
$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$



Weighted Automata and Recurrent Neural Networks

- The hidden state of a second-order RNN is computed by

$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$

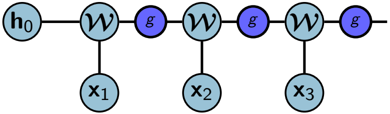


- The computation of a weighted automaton is very similar!

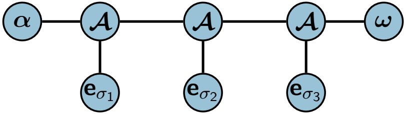
Weighted Automata and Recurrent Neural Networks

- The hidden state of a second-order RNN is computed by

$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$



- The computation of a weighted automaton is very similar!



(where $\mathcal{A} \in \mathbb{R}^{n \times \Sigma \times n}$ defined by $\mathcal{A}_{:,\sigma,:} = \mathbf{A}^\sigma$)

WAs $\equiv$ linear 2-RNNs

Theorem

WAs are expressively equivalent to second-order linear RNNs for computing functions over ***sequences of discrete symbols****.*

WAs $\equiv$ linear 2-RNNs

Theorem

WAs are expressively equivalent to second-order linear RNNs for computing functions over ***sequences of discrete symbols****.*

- But 2-RNNs can compute functions over sequences of continuous vectors (e.g., word embeddings), what about WAs?

$\hookrightarrow$ We can extend the definitions of WAs to continuous vectors!

Continuous WA / linear 2-RNN

Definition

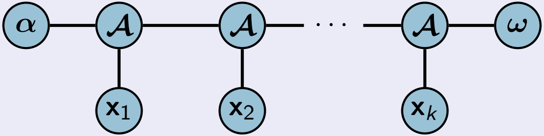
A **continuous WA** is a tuple $A = (\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ where

$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector

$\boldsymbol{\omega} \in \mathbb{R}^n$ final weights vector

$\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n \times d \times n}$ is the transition tensor.

A computes a function $f_A : (\mathbb{R}^d)^* \to \mathbb{R}$ defined by

$f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) =$ 

WAs $\equiv$ linear 2-RNNs

Theorem

WAs are expressively equivalent to second-order linear RNNs (linear 2-RNNs) for computing functions over ***sequences of discrete symbols****.*

- But 2-RNNs can compute functions over sequences of continuous vectors (e.g., word embeddings), what about WAs?

$\hookrightarrow$ We can extend the definitions of WAs to continuous vectors!

WAs $\equiv$ linear 2-RNNs

Theorem

WAs are expressively equivalent to second-order linear RNNs (linear 2-RNNs) for computing functions over ***sequences of discrete symbols****.*

- But 2-RNNs can compute functions over sequences of continuous vectors (e.g., word embeddings), what about WAs?

$\hookrightarrow$ We can extend the definitions of WAs to continuous vectors!

- Can we learn linear 2-RNNs from data?
  - ⋆ Over sequences of discrete symbols?

  $\hookrightarrow$ Yes: spectral learning of WA
  - ⋆ Over sequences of continuous vectors?

  $\hookrightarrow$ Yes: technical contribution of [AISTATS'19]

Future directions

- Extension to tree models:
  - Linear Recursive Tensor Neural Networks (Socher et al., 2013) are Weighted Tree Automata!
  - Continuous extension of WTA and spectral learning algorithm.

Future directions

- Extension to tree models:
  - Linear Recursive Tensor Neural Networks (Socher et al., 2013) are Weighted Tree Automata!
  - Continuous extension of WTA and spectral learning algorithm.
- What do linear counterparts of neural sequential models (LSTMs, bi-directionnal RNNs, etc.) correspond to?

Future directions

- Extension to tree models:
  - Linear Recursive Tensor Neural Networks (Socher et al., 2013) are Weighted Tree Automata!
  - Continuous extension of WTA and spectral learning algorithm.
- What do linear counterparts of neural sequential models (LSTMs, bi-directionnal RNNs, etc.) correspond to?
- Spectral initialization of RNNs (*ongoing work of Maude Lizaire*).

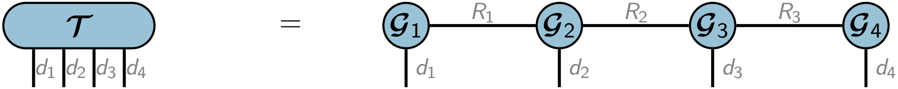
Future directions

- Extension to tree models:
  - Linear Recursive Tensor Neural Networks (Socher et al., 2013) are Weighted Tree Automata!
  - Continuous extension of WTA and spectral learning algorithm.
- What do linear counterparts of neural sequential models (LSTMs, bi-directionnal RNNs, etc.) correspond to?
- Spectral initialization of RNNs (*ongoing work of Maude Lizaire*).
- More accurate map of equivalences between WA and RNNs (e.g. Multiplicative interaction RNNs are special case of 2nd order RNNs)...

Tensor Network Models for Sequences

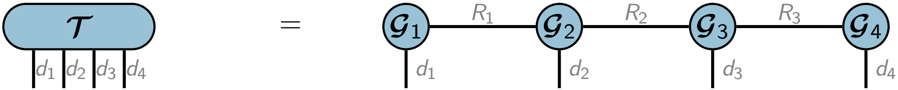
Tensor Train / Matrix Product State (MPS) decomposition

- Tensor Train decomposition [Oseledets, 2011]:



Tensor Train / Matrix Product State (MPS) decomposition

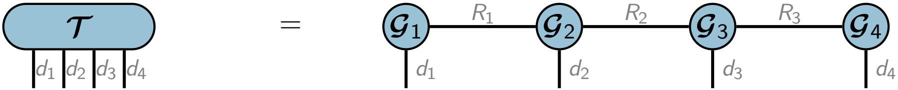
- Tensor Train decomposition [Oseledets, 2011]:



$\Rightarrow$ $d_1R_1 + R_1d_2R_2 + R_2d_2R_3 + R_3d_4$ parameters instead of $d_1d_2d_3d_4$.

Tensor Train / Matrix Product State (MPS) decomposition

- Tensor Train decomposition [Oseledets, 2011]:

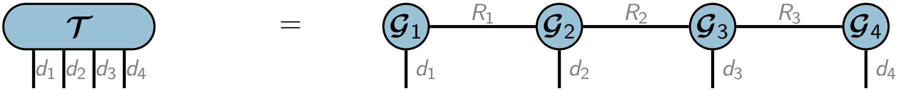


$\Rightarrow$ $d_1 R_1 + R_1 d_2 R_2 + R_2 d_2 R_3 + R_3 d_4$ parameters instead of $d_1 d_2 d_3 d_4$.

- If the ranks are all the same $(R_1 = R_2 = \cdots = R)$, can represent a vector of size 2^n with $\mathcal{O}\left(nR^2\right)$ parameters!

Tensor Train / Matrix Product State (MPS) decomposition

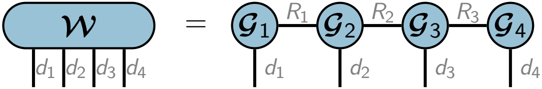
- Tensor Train decomposition [Oseledets, 2011]:



$\Rightarrow$ $d_1R_1 + R_1d_2R_2 + R_2d_2R_3 + R_3d_4$ parameters instead of $d_1d_2d_3d_4$.

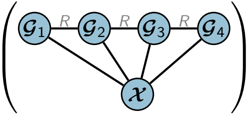
- If the ranks are all the same $(R_1 = R_2 = \cdots = R)$, can represent a vector of size 2^n with $\mathcal{O}(nR^2)$ parameters!
- We can also efficiently perform operations on TT tensors:
  - Inner product, sum, component-wise product, ... all in time linear in n for vectors of size d^n.

Tensor Train / Matrix Product States

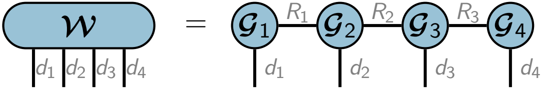


- We can parameterize linear classification models with MPS [Stoudenmire & Schwab, 2016]:

$$f(\mathcal{X}) = \mathrm{sign}(\langle \mathcal{W}, \mathcal{X} \rangle) = \mathrm{sign}$$

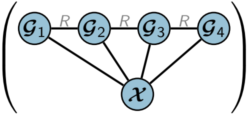


Tensor Train / Matrix Product States



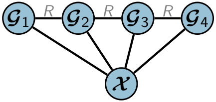
- We can parameterize linear classification models with MPS [Stoudenmire & Schwab, 2016]:

$$f(\boldsymbol{\mathcal{X}}) = \mathrm{sign}(\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}} \rangle) = \mathrm{sign} \left(\quad \right)$$

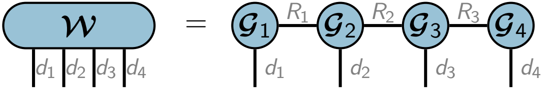


- We can also model probability distributions with MPS [Han et al., 2018]:

$$\mathbb{P}(\boldsymbol{\mathcal{X}}) =$$

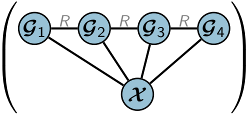


Tensor Train / Matrix Product States



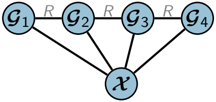
- We can parameterize linear classification models with MPS [Stoudenmire & Schwab, 2016]:

$$f(\mathcal{X}) = \text{sign}(\langle \mathcal{W}, \mathcal{X} \rangle) = \text{sign}$$



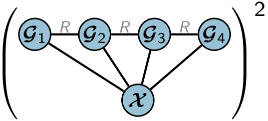
- We can also model probability distributions with MPS [Han et al., 2018]:

$$\mathbb{P}(\mathcal{X}) =$$



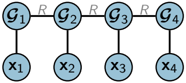
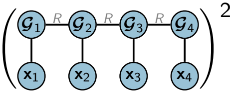
or

$$\mathbb{P}(\mathcal{X}) =$$



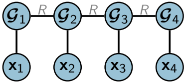
MPS for sequence modeling

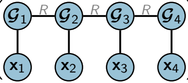
- We can also use MPS to model functions and distributions over fixed length sequences:

$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$  or $\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$ 

MPS for sequence modeling

- We can also use MPS to model functions and distributions over fixed length sequences:

$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$ 

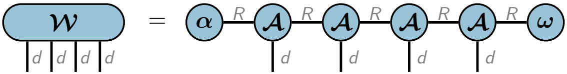
or $\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \Big($ 

$\Big)^2$

$\hookrightarrow$ How to model distributions over variable length sequences?

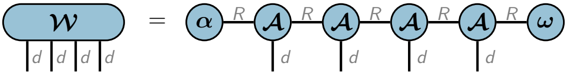
Uniform MPS

- **uniform MPS** (uMPS) decomposition $\equiv$ MPS with same core at each site:

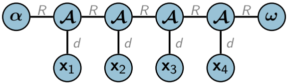
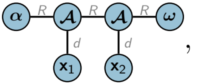


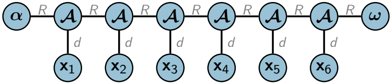
Uniform MPS

- **uniform MPS** (uMPS) decomposition $\equiv$ MPS with same core at each site:



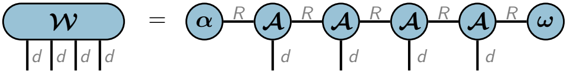
- With uMPS, we can model functions and distributions over variable length sequences:

$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$  $, \mathbb{P}(\mathbf{x}_1, \mathbf{x}_2) =$  ,

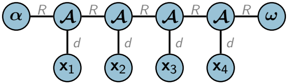
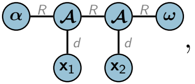
$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6) =$  $, \cdots$

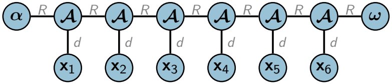
Uniform MPS

- **uniform MPS** (uMPS) decomposition $\equiv$ MPS with same core at each site:



- With uMPS, we can model functions and distributions over variable length sequences:

$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$  , $\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2) =$  ,

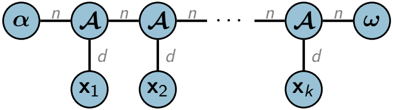
$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6) =$  $, \cdots$

$\hookrightarrow$ Nothing else than the continuous WA we defined previously!

Connections between uMPS and other models

- A uMPS is given by a tuple $(\boldsymbol{\alpha} \in \mathbb{R}^n, \boldsymbol{\mathcal{A}} \in \mathbb{R}^{n\times d\times n}, \boldsymbol{\omega} \in \mathbb{R}^n)$ and maps any sequence of vectors $\mathbf{x}_1, \cdots, \mathbf{x}_k \in \mathbb{R}^d$ to a scalar:

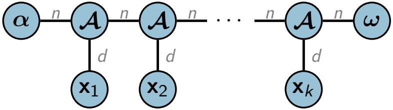
$$f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) =$$



Connections between uMPS and other models

- A uMPS is given by a tuple $(\boldsymbol{\alpha} \in \mathbb{R}^n, \boldsymbol{\mathcal{A}} \in \mathbb{R}^{n \times d \times n}, \boldsymbol{\omega} \in \mathbb{R}^n)$ and maps any sequence of vectors $\mathbf{x}_1, \cdots, \mathbf{x}_k \in \mathbb{R}^d$ to a scalar:

$$f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) =$$

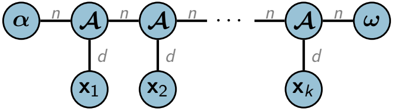


- If the inputs are one-hot encoding, uMPS $\equiv$ Weighted Automata (generalization of HMMs)
  - $\hookrightarrow$ If the probability of a sequence is $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)^2 \equiv$ Quadratic weighted automata (Bailly, 2011) / MPS from quantum physics

Connections between uMPS and other models

- A uMPS is given by a tuple $(\boldsymbol{\alpha} \in \mathbb{R}^n, \boldsymbol{\mathcal{A}} \in \mathbb{R}^{n\times d\times n}, \boldsymbol{\omega} \in \mathbb{R}^n)$ and maps any sequence of vectors $\mathbf{x}_1, \cdots, \mathbf{x}_k \in \mathbb{R}^d$ to a scalar:

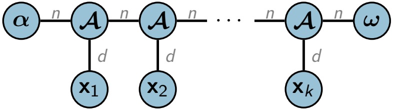
$f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) =$



- If the inputs are one-hot encoding, uMPS $\equiv$ Weighted Automata (generalization of HMMs)
  - $\hookrightarrow$ If the probability of a sequence is $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)^2 \equiv$ Quadratic weighted automata (Bailly, 2011) / MPS from quantum physics
- Linear second order RNNs $\equiv$ uMPS

Connections between uMPS and other models

- A uMPS is given by a tuple $(\boldsymbol{\alpha} \in \mathbb{R}^n, \boldsymbol{\mathcal{A}} \in \mathbb{R}^{n \times d \times n}, \boldsymbol{\omega} \in \mathbb{R}^n)$ and maps any sequence of vectors $\mathbf{x}_1, \cdots, \mathbf{x}_k \in \mathbb{R}^d$ to a scalar:

$f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) =$ 

- If the inputs are one-hot encoding, uMPS $\equiv$ Weighted Automata (generalization of HMMs)
  - $\hookrightarrow$ If the probability of a sequence is $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)^2 \equiv$ Quadratic weighted automata (Bailly, 2011) / MPS from quantum physics
- Linear second order RNNs $\equiv$ uMPS
- For a thorough discussion of connections between uMPS, stochastic processes and automata, see

Srinivasan, S., Adhikary, S., Miller, J., Rabusseau, G. and Boots, B.

Quantum Tensor Networks, Stochastic Processes, and Weighted Automata (AISTATS 2021).

Future Directions

- Versatile sampling algorithm:
  - We can exactly sample from a uMPS/WFA distribution projected onto the support of a regular language / context free grammar.

Jacob Miller, Guillaume Rabusseau, and John Terilla. *Tensor Networks for Language Modeling.* arXiv preprint arXiv:2003.01039 (AISTATS 2021).

Future Directions

- Versatile sampling algorithm:
  - We can exactly sample from a uMPS/WFA distribution projected onto the support of a regular language / context free grammar.

  Jacob Miller, Guillaume Rabusseau, and John Terilla. *Tensor Networks for Language Modeling.* arXiv preprint arXiv:2003.01039 (AISTATS 2021).

- Scale up learning to very large state spaces (work of Jacob Miller).

Future Directions

- Versatile sampling algorithm:
  - We can exactly sample from a uMPS/WFA distribution projected onto the support of a regular language / context free grammar.

  Jacob Miller, Guillaume Rabusseau, and John Terilla. *Tensor Networks for Language Modeling.* arXiv preprint arXiv:2003.01039 (AISTATS 2021).

- Scale up learning to very large state spaces (work of Jacob Miller).
- Training uMPS/WFA with word embeddings for language modeling (work of Jacob Miller).

Future Directions

- Versatile sampling algorithm:
  - We can exactly sample from a uMPS/WFA distribution projected onto the support of a regular language / context free grammar.

  Jacob Miller, Guillaume Rabusseau, and John Terilla. *Tensor Networks for Language Modeling.* arXiv preprint arXiv:2003.01039 (AISTATS 2021).

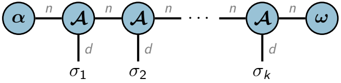
- Scale up learning to very large state spaces (work of Jacob Miller).
- Training uMPS/WFA with word embeddings for language modeling (work of Jacob Miller).
- Learning dynamics in uMPS/WFA trained by gradient based method could provide theoretical insights on training RNN.

A Tensor Network View of the Spectral Learning Algorithm

Hankel matrix

- We consider the case where inputs are sequences of discrete symbols:
  - Σ a finite alphabet of size d (e.g. $\{a, b\}$)
  - Σ^* strings on Σ (e.g. *abba*)
  - A uMPS computes a function $f : \Sigma^* \to \mathbb{R}$:

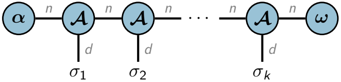
$$f(\sigma_1 \cdots \sigma_k) =$$



Hankel matrix

- We consider the case where inputs are sequences of discrete symbols:
  - Σ a finite alphabet of size d (e.g. $\{a, b\}$)
  - Σ^* strings on Σ (e.g. $abba$)
  - A uMPS computes a function $f : \Sigma^* \to \mathbb{R}$:

$f(\sigma_1 \cdots \sigma_k) =$



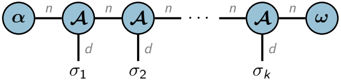
- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$
  - *Definition*: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

$$
\begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & aa & ab & \dots \end{array} \\
\begin{bmatrix}
f(aa) & f(ab) & \dots & \dots & \dots \\
f(ba) & f(bb) & \dots & \dots & \dots \\
f(aaa) & f(aab) & \dots & \dots & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\end{array}
$$

Hankel matrix

- We consider the case where inputs are sequences of discrete symbols:
  - Σ a finite alphabet of size d (e.g. $\{a, b\}$)
  - Σ^* strings on Σ (e.g. *abba*)
  - A uMPS computes a function $f : \Sigma^* \to \mathbb{R}$:

$$f(\sigma_1 \cdots \sigma_k) =$$



- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$
  - *Definition*: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

	a	b	aa	ab	$\dots$
a	$f(aa)$	$f(ab)$	$\dots$	$\dots$	$\dots$
b	$f(ba)$	$f(bb)$	$\dots$	$\dots$	$\dots$
aa	$f(aaa)$	$f(aab)$	$\dots$	$\dots$	$\dots$
ab	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Spectral Learning of uMPS

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$

  Definition: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

- Fundamental theorem [Carlyle and Paz, 1971; Fliess 1974]:

$$\text{rank}(\mathbf{H}_f) < \infty \iff f \text{ can be computed by a uMPS}$$

Spectral Learning of uMPS

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$

  Definition: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

- Fundamental theorem [Carlyle and Paz, 1971; Fliess 1974]:

  $$\text{rank}(\mathbf{H}_f) < \infty \iff f \text{ can be computed by a uMPS}$$

$\hookrightarrow$ Proof is constructive! From a low rank factorization of $\mathbf{H}_f$ we can recover a uMPS computing f...

Spectral Learning of uMPS (in a nutshell)

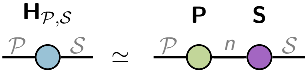
1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.

Spectral Learning of uMPS (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{h}_{\mathcal{P}} \in \mathbb{R}^{\mathcal{P}}$, $\mathbf{h}_{\mathcal{S}} \in \mathbb{R}^{\mathcal{S}}$, $\mathbf{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$, $\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\Sigma\times\mathcal{S}}$ defined by
   $(\mathbf{h}_{\mathcal{P}})_u = f(u),\ (\mathbf{h}_{\mathcal{S}})_v = f(v),\ (\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$

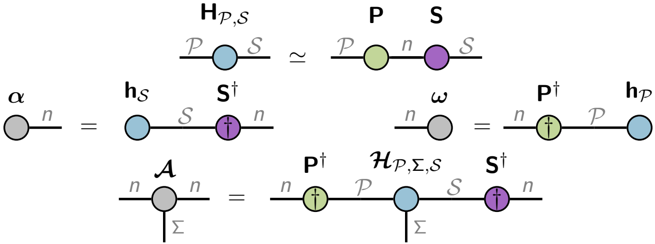
Spectral Learning of uMPS (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{h}_{\mathcal{P}} \in \mathbb{R}^{\mathcal{P}}$, $\mathbf{h}_{\mathcal{S}} \in \mathbb{R}^{\mathcal{S}}$, $\mathbf{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$, $\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\Sigma\times\mathcal{S}}$ defined by
   $(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$
3. Recover uMPS parameters $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$:



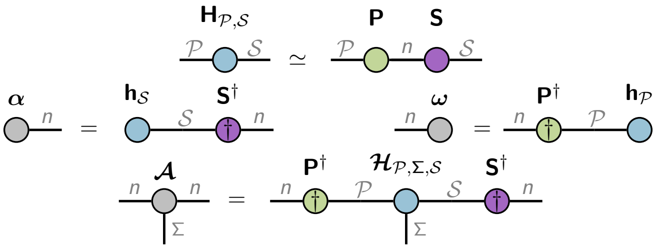
Spectral Learning of uMPS (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{h}_\mathcal{P} \in \mathbb{R}^\mathcal{P}$, $\mathbf{h}_\mathcal{S} \in \mathbb{R}^\mathcal{S}$, $\mathbf{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$, $\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\Sigma\times\mathcal{S}}$ defined by
   $(\mathbf{h}_\mathcal{P})_u = f(u),\ (\mathbf{h}_\mathcal{S})_v = f(v),\ (\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$
3. Recover uMPS parameters $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$:



Spectral Learning of uMPS (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{h}_{\mathcal{P}} \in \mathbb{R}^{\mathcal{P}}$, $\mathbf{h}_{\mathcal{S}} \in \mathbb{R}^{\mathcal{S}}$, $\mathbf{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$, $\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\Sigma\times\mathcal{S}}$ defined by
   $(\mathbf{h}_{\mathcal{P}})_u = f(u),\ (\mathbf{h}_{\mathcal{S}})_v = f(v),\ (\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$
3. Recover uMPS parameters $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$:



$\rightarrow$ Efficient and consistent learning algorithms for uMPS/weighted automata [Hsu et al., 2009; Bailly et al. 2009; Balle et al., 2014, ...].

Spectral Learning: when does it work?

Theorem (Exact case)

If the set of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ *are such that*

$$\text{rank}(\mathbf{H}_{\mathcal{P},\mathcal{S}}) = \text{rank}(\mathbf{H}_f) < \infty$$

then the spectral learning algorithm returns a uMPS computing f.

Spectral Learning: when does it work?

Theorem (Exact case)

If the set of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ are such that

$$\text{rank}(\mathbf{H}_{\mathcal{P},\mathcal{S}}) = \text{rank}(\mathbf{H}_f) < \infty$$

then the spectral learning algorithm returns a uMPS computing f.

Suppose f is computed by a uMPS. By a continuity argument, if we are given noisy estimates
$\hat{\mathbf{H}}_{\mathcal{P},\mathcal{S}} = \mathbf{H}_{\mathcal{P},\mathcal{S}} + \boldsymbol{\xi}_{\mathcal{P},\mathcal{S}},\ \hat{\boldsymbol{\mathcal{H}}}_{\mathcal{P},\Sigma,\mathcal{S}} = \hat{\boldsymbol{\mathcal{H}}}_{\mathcal{P},\Sigma,\mathcal{S}} + \boldsymbol{\xi}_{\mathcal{P},\Sigma,\mathcal{S}}, \ldots$ we have

$$\lim_{\|\boldsymbol{\xi}_{\mathcal{P},\mathcal{S}}\|\to 0,\quad \|\boldsymbol{\xi}_{\mathcal{P},\Sigma,\mathcal{S}}\|\to 0} \hat{f} = f$$

where $\hat{f}$ is the estimator returned by the spectral method.

Spectral Learning: when does it work?

Theorem (Exact case)

If the set of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ are such that

$$\mathrm{rank}(\mathbf{H}_{\mathcal{P},\mathcal{S}}) = \mathrm{rank}(\mathbf{H}_f) < \infty$$

then the spectral learning algorithm returns a uMPS computing f.

Suppose f is computed by a uMPS. By a continuity argument, if we are given noisy estimates
$\hat{\mathbf{H}}_{\mathcal{P},\mathcal{S}} = \mathbf{H}_{\mathcal{P},\mathcal{S}} + \boldsymbol{\xi}_{\mathcal{P},\mathcal{S}}, \ \hat{\boldsymbol{\mathcal{H}}}_{\mathcal{P},\Sigma,\mathcal{S}} = \hat{\boldsymbol{\mathcal{H}}}_{\mathcal{P},\Sigma,\mathcal{S}} + \boldsymbol{\xi}_{\mathcal{P},\Sigma,\mathcal{S}}, \dots$ we have

$$\lim_{\|\boldsymbol{\xi}_{\mathcal{P},\mathcal{S}}\| \to 0, \ \ \|\boldsymbol{\xi}_{\mathcal{P},\Sigma,\mathcal{S}}\| \to 0} \hat{f} = f$$

where $\hat{f}$ is the estimator returned by the spectral method.

$\hookrightarrow$ When f is a probability distribution, we get an unbiased and consistent estimator! [c.f. work of B. Balle]

A closer look at the Hankel matrix of a uMPS

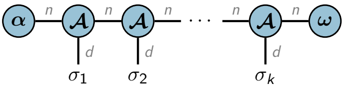
- Let $f : \Sigma^* \to \mathbb{R}$ be the function computed by a uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$.
- Define the ℓth order Hankel tensor $\boldsymbol{\mathcal{H}}^{(\ell)} \in \mathbb{R}^{\Sigma\times\Sigma\times\cdots\times\Sigma}$ by

$$\boldsymbol{\mathcal{H}}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell)$$

A closer look at the Hankel matrix of a uMPS

- Let $f : \Sigma^* \to \mathbb{R}$ be the function computed by a uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$.
- Define the ℓth order Hankel tensor $\boldsymbol{\mathcal{H}}^{(\ell)} \in \mathbb{R}^{\Sigma \times \Sigma \times \cdots \times \Sigma}$ by

$$\boldsymbol{\mathcal{H}}^{(\ell)}_{\sigma_1, \sigma_2, \cdots, \sigma_\ell} = f(\sigma_1 \sigma_2 \cdots \sigma_\ell) = \qquad (1)$$

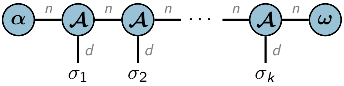


for all $\sigma_1, \cdots \sigma_\ell \in \Sigma$

A closer look at the Hankel matrix of a uMPS

- Let $f : \Sigma^* \to \mathbb{R}$ be the function computed by a uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$.
- Define the ℓth order Hankel tensor $\boldsymbol{\mathcal{H}}^{(\ell)} \in \mathbb{R}^{\Sigma \times \Sigma \times \cdots \times \Sigma}$ by

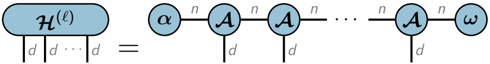
$$\boldsymbol{\mathcal{H}}^{(\ell)}_{\sigma_1, \sigma_2, \cdots, \sigma_\ell} = f(\sigma_1 \sigma_2 \cdots \sigma_\ell) =$$



(1)

for all $\sigma_1, \cdots \sigma_\ell \in \Sigma$

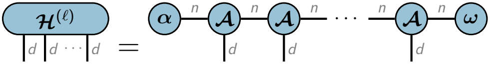
- For each ℓ, the tensor $\boldsymbol{\mathcal{H}}^{(\ell)}$ has low uniform MPS rank:



(2)

A closer look at the Hankel matrix of a uMPS

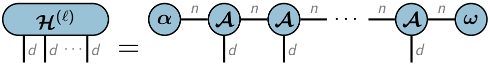
- For each ℓ, the tensor $\boldsymbol{\mathcal{H}}^{(\ell)}$ (defined by $\boldsymbol{\mathcal{H}}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell)$) has low uniform MPS rank:



(3)

A closer look at the Hankel matrix of a uMPS

- For each ℓ, the tensor $\boldsymbol{\mathcal{H}}^{(\ell)}$ (defined by $\boldsymbol{\mathcal{H}}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell)$) has low uniform MPS rank:



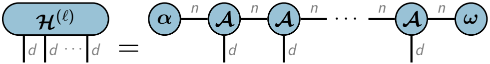
(3)

- It follows that the Hankel matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ can be decomposed in sub-blocks of low uMPS rank:

$$\mathbf{H}_f = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & aa & ab & \dots \end{array} \\ \begin{bmatrix} f(aa) & f(ab) & \dots & \dots & \dots \\ f(ba) & f(bb) & \dots & \dots & \dots \\ f(aaa) & f(aab) & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

A closer look at the Hankel matrix of a uMPS

- For each ℓ, the tensor $\boldsymbol{\mathcal{H}}^{(\ell)}$ (defined by $\boldsymbol{\mathcal{H}}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell)$) has low uniform MPS rank:



(3)

- It follows that the Hankel matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^*\times\Sigma^*}$ can be decomposed in sub-blocks of low uMPS rank:

$$\mathbf{H}_f = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & aa & ab & \dots \end{array} \\ \begin{bmatrix} f(aa) & f(ab) & \dots & \dots & \dots \\ f(ba) & f(bb) & \dots & \dots & \dots \\ f(aaa) & f(aab) & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array} = \begin{array}{c|c|c|c} & a\ \ b & aa\ \ ab\ \ ba\ \ bb & aaa\ \ aab\ \ \dots \\ \hline \begin{array}{c} a \\ b \end{array} & \mathcal{H}^{(2)}_{\Sigma\times\Sigma} & \mathcal{H}^{(3)}_{\Sigma\times\Sigma^2} & \mathcal{H}^{(4)}_{\Sigma\times\Sigma^3} \quad \dots \\ \hline \begin{array}{c} aa \\ ab \\ ba \\ bb \end{array} & \mathcal{H}^{(3)}_{\Sigma^2\times\Sigma} & \mathcal{H}^{(4)}_{\Sigma^2\times\Sigma^2} & \mathcal{H}^{(5)}_{\Sigma^2\times\Sigma^3} \quad \dots \\ \hline \begin{array}{c} aaa \\ aab \\ \vdots \end{array} & \mathcal{H}^{(4)}_{\Sigma^3\times\Sigma} \ \vdots & \mathcal{H}^{(5)}_{\Sigma^3\times\Sigma^2} \ \vdots & \mathcal{H}^{(6)}_{\Sigma^3\times\Sigma^3} \ \vdots \quad \dots \end{array}$$

Back to the spectral learning algorithm

- In the spectral algorithm, we need to estimate $(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$ for some sets of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.

Back to the spectral learning algorithm

- In the spectral algorithm, we need to estimate $(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$ for some sets of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
- If we choose $\mathcal{P} = \mathcal{S} = \Sigma^\ell$ we have

$$\mathbf{h}_{\mathcal{P}} = \mathbf{h}_{\mathcal{S}} = \boldsymbol{\mathcal{H}}^{(\ell)}_{\Sigma^\ell}, \quad \mathbf{H}_{\mathcal{P},\mathcal{S}} = \boldsymbol{\mathcal{H}}^{(2\ell)}_{\Sigma^\ell \times \Sigma^\ell} \text{ and } (\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}}) = \boldsymbol{\mathcal{H}}^{(2\ell+1)}_{\Sigma^\ell \times \Sigma \times \Sigma^\ell}$$

Back to the spectral learning algorithm

- In the spectral algorithm, we need to estimate $(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$ for some sets of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
- If we choose $\mathcal{P} = \mathcal{S} = \Sigma^\ell$ we have

$$\mathbf{h}_{\mathcal{P}} = \mathbf{h}_{\mathcal{S}} = \boldsymbol{\mathcal{H}}^{(\ell)}_{\Sigma^\ell}, \quad \mathbf{H}_{\mathcal{P},\mathcal{S}} = \boldsymbol{\mathcal{H}}^{(2\ell)}_{\Sigma^\ell \times \Sigma^\ell} \text{ and } (\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}}) = \boldsymbol{\mathcal{H}}^{(2\ell+1)}_{\Sigma^\ell \times \Sigma \times \Sigma^\ell}$$

$\hookrightarrow$ All the quantities we need to estimate are matricization of low uMPS rank tensors!

Back to the spectral learning algorithm

- In the spectral algorithm, we need to estimate $(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$ for some sets of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
- If we choose $\mathcal{P} = \mathcal{S} = \Sigma^\ell$ we have

$$\mathbf{h}_{\mathcal{P}} = \mathbf{h}_{\mathcal{S}} = \boldsymbol{\mathcal{H}}^{(\ell)}_{\Sigma^\ell}, \quad \mathbf{H}_{\mathcal{P},\mathcal{S}} = \boldsymbol{\mathcal{H}}^{(2\ell)}_{\Sigma^\ell \times \Sigma^\ell} \text{ and } (\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}}) = \boldsymbol{\mathcal{H}}^{(2\ell+1)}_{\Sigma^\ell \times \Sigma \times \Sigma^\ell}$$

$\hookrightarrow$ All the quantities we need to estimate are matricization of low uMPS rank tensors!

- This leads to an efficient learning algorithm:
  - Estimate $\boldsymbol{\mathcal{H}}^{(\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell+1)}$ directly in the MPS/TT format
  - Use the spectral algorithm to convert the MPS decomposition into a uniform MPS model.

Spectral Learning $\equiv$ Conversion from MPS to uMPS

- Let $f : \Sigma^* \to \mathbb{R}$ be a function for which we have access to an MPS decomposition of the Hankel tensors $\boldsymbol{\mathcal{H}}^{(\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell+1)}$.

  $\rightarrow$ f can be a probability distribution, or the wave function of a quantum system.

- Spectral learning algorithm $\equiv$ **efficient** way to recover a uMPS computing f from the 3 Hankel tensors

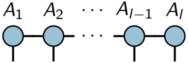
Spectral Learning $\equiv$ Conversion from MPS to uMPS

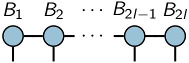
- Let $f : \Sigma^* \to \mathbb{R}$ be a function for which we have access to an MPS decomposition of the Hankel tensors $\boldsymbol{\mathcal{H}}^{(\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell+1)}$.

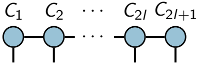
  $\to$ f can be a probability distribution, or the wave function of a quantum system.

- Spectral learning algorithm $\equiv$ **efficient** way to recover a uMPS computing f from the 3 Hankel tensors

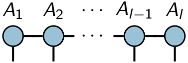
$\hookrightarrow$ From $\boldsymbol{\mathcal{H}}^{(\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell)}, \boldsymbol{\mathcal{H}}^{(2\ell+1)}$, we can compute the value of f on sequences of arbitrary length!

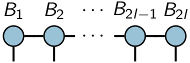
Input: $\mathcal{H}^{(\ell)} =$ 

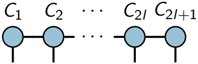
, $\mathcal{H}^{(2\ell)} =$ 

, $\mathcal{H}^{(2\ell+1)} =$ 

Output: uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ computing f

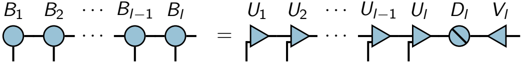
Input: $\mathcal{H}^{(\ell)} =$ 

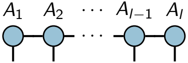
, $\mathcal{H}^{(2\ell)} =$ 

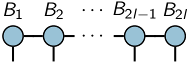
, $\mathcal{H}^{(2\ell+1)} =$ 

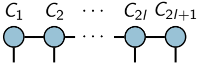
Output: uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ computing f

1. Left-orthonormalisation of $B_1, \cdots, B_\ell$ (first half of $\mathcal{H}^{(2\ell)}$)



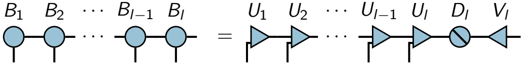
Input: $\mathcal{H}^{(\ell)} =$ 

, $\mathcal{H}^{(2\ell)} =$ 

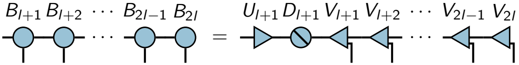
, $\mathcal{H}^{(2\ell+1)} =$ 

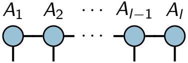
Output: uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ computing f

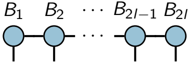
1. Left-orthonormalisation of $B_1, \cdots, B_\ell$ (first half of $\mathcal{H}^{(2\ell)}$)

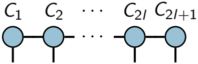


2. Right-orthonormalisation of $B_{\ell+1}, \cdots, B_{2\ell}$ (second half of $\mathcal{H}^{(2\ell)}$)



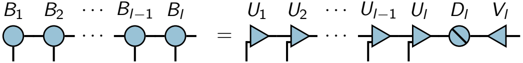
Input: $\mathcal{H}^{(\ell)} =$ 

, $\mathcal{H}^{(2\ell)} =$ 

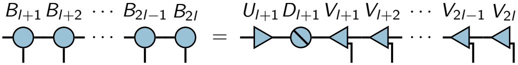
, $\mathcal{H}^{(2\ell+1)} =$ 

Output: uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ computing f

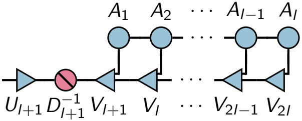
1. Left-orthonormalisation of $B_1, \cdots, B_\ell$ (first half of $\mathcal{H}^{(2\ell)}$)

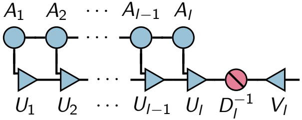


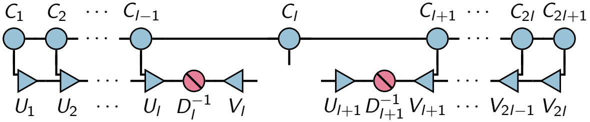
2. Right-orthonormalisation of $B_{\ell+1}, \cdots, B_{2\ell}$ (second half of $\mathcal{H}^{(2\ell)}$)



3. Computation of the uMPS parameters:

$\boldsymbol{\alpha} =$ 

$\boldsymbol{\omega} =$ 

$\boldsymbol{\mathcal{A}} =$ 

Spectral Learning with Tensor Networks

- Recap:
  - More structure than matrix rank in the Hankel matrix.
  - When $\mathcal{P} = \mathcal{S} = \Sigma^\ell$, the spectral learning algorithm can be performed efficiently in the MPS/TT format.

$\hookrightarrow$ Time complexity is reduced from $\mathcal{O}\left(n|\Sigma|^{2\ell} + n^2|\Sigma|^{\ell+1}\right)$ to $\mathcal{O}\left(n^3\ell|\Sigma|\right)$.

Spectral Learning with Tensor Networks

- Recap:
  - More structure than matrix rank in the Hankel matrix.
  - When $\mathcal{P} = \mathcal{S} = \Sigma^\ell$, the spectral learning algorithm can be performed efficiently in the MPS/TT format.

  $\hookrightarrow$ Time complexity is reduced from $\mathcal{O}\left(n|\Sigma|^{2\ell} + n^2|\Sigma|^{\ell+1}\right)$ to $\mathcal{O}\left(n^3\ell|\Sigma|\right)$.
- Future directions:
  - Spectral learning of continuous WA/uMPS for RL (work of Tianyu Li)
  - Similar connections and algorithms can be derived for models on trees
  - What about graphs? (e.g. potential connections between TN and GNN)
  - Lots of connections between quantum TN, probabilistic models, formal languages, machine learning, etc. to explore!
    (e.g., using density matrices to model languages (see work of Tai-Danae Bradley)

That's all, folks!

Thanks for listening!
Questions?