Tensor networks, Weighted Automata & Spectral Learning

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

August 27, 2021
ICGI 2021

Aim of the talk

- Introduce tensor networks ($\neq$ tensor neural networks)
- What do tensor networks have to do with grammatical inference?
- Extend weighted finite automata (WFA) to continuous sequences (connections with RNN)
- How tensor networks can help learning continuous WFA and scale up spectral learning.

Aim of the talk

- Introduce tensor networks ($\neq$ tensor neural networks)
- What do tensor networks have to do with grammatical inference?
- Extend weighted finite automata (WFA) to continuous sequences (connections with RNN)
- How tensor networks can help learning continuous WFA and scale up spectral learning.

Connecting tensor networks and WFA for fun and (maybe) profit

Outline

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

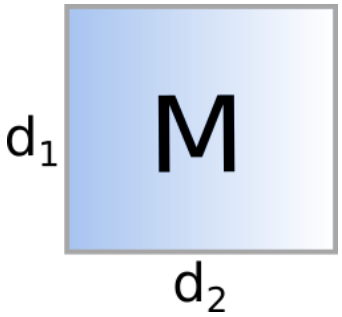
- Rabusseau, Guillaume, Tianyu Li, and Doina Precup. *"Connecting weighted automata and recurrent neural networks through spectral learning."* The 22nd International Conference on Artificial Intelligence and Statistics. PMLR, 2019.
- Li, Tianyu, Doina Precup, and Guillaume Rabusseau. *"Connecting Weighted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

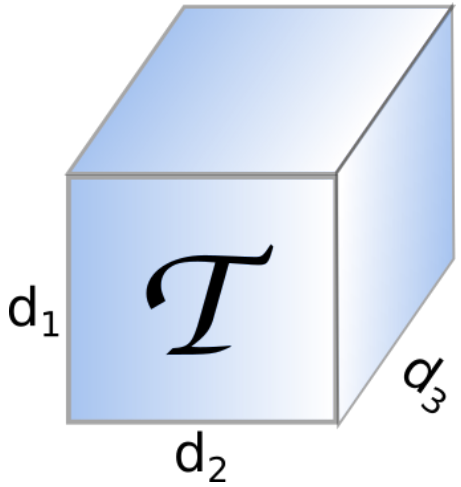
Tianyu Li:

Preliminaries

Tensor Networks

Tensors





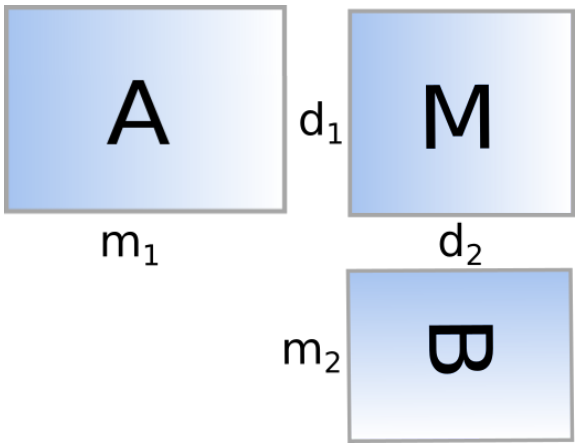
$\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$

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

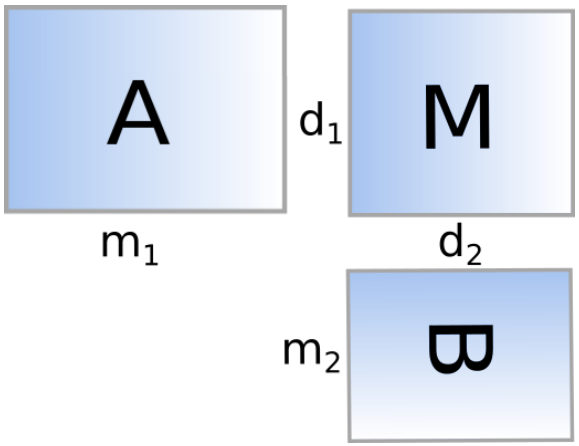
$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

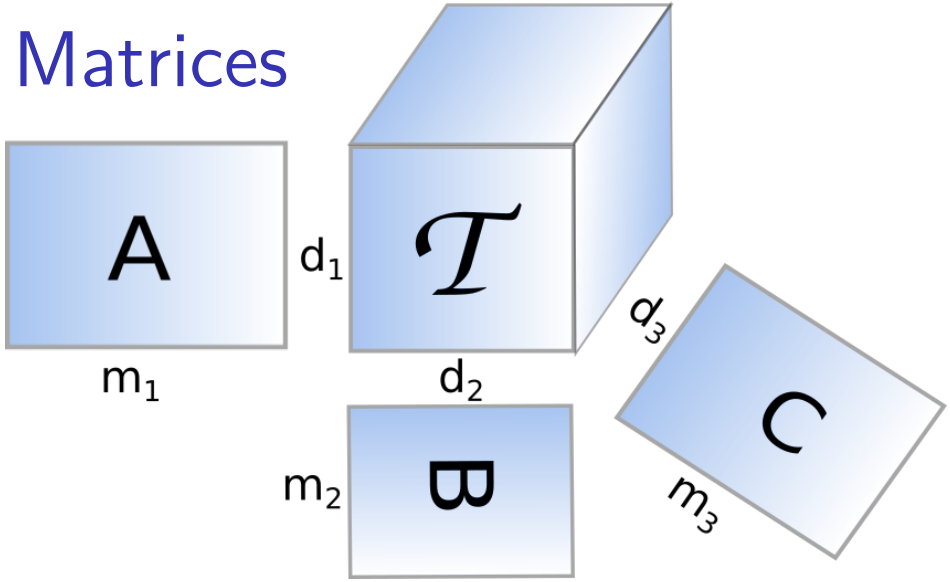
$(\mathcal{T}_{ijk}) \in \mathbb{R}$ for $i \in [d_1], j \in [d_2], k \in [d_3]$

Tensors: Multiplication with Matrices



Tensors: Multiplication with Matrices

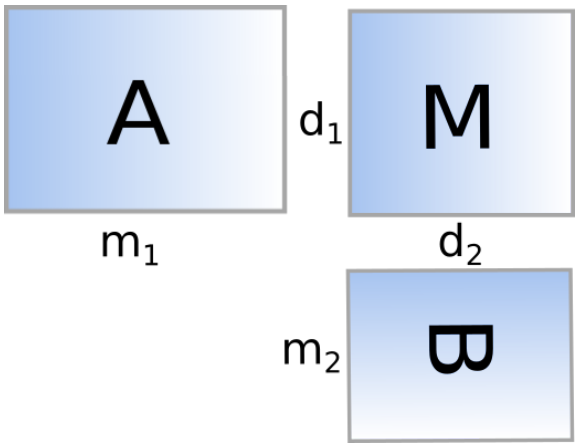


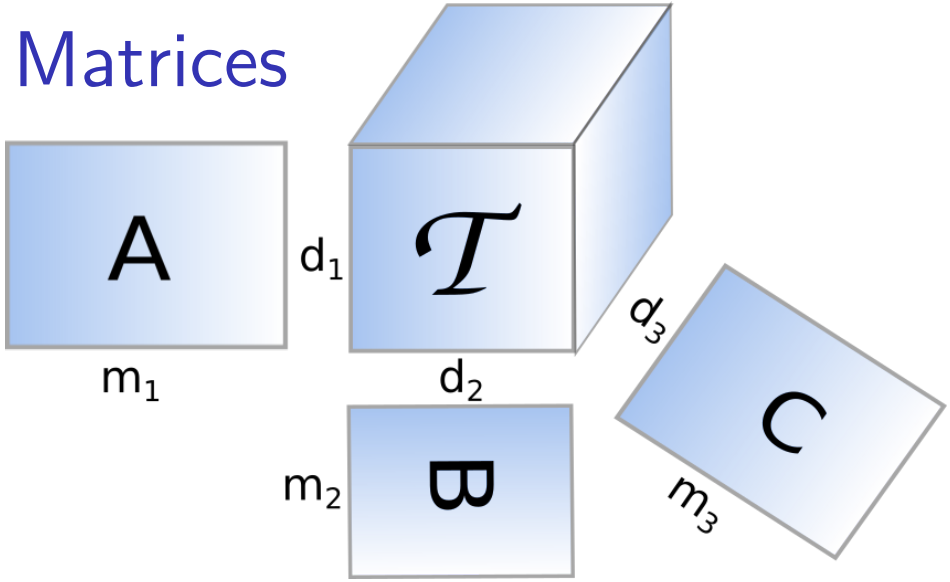


$\mathbf{AMB}^\top \in \mathbb{R}^{m_1 \times m_2}$

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

Tensors: Multiplication with Matrices



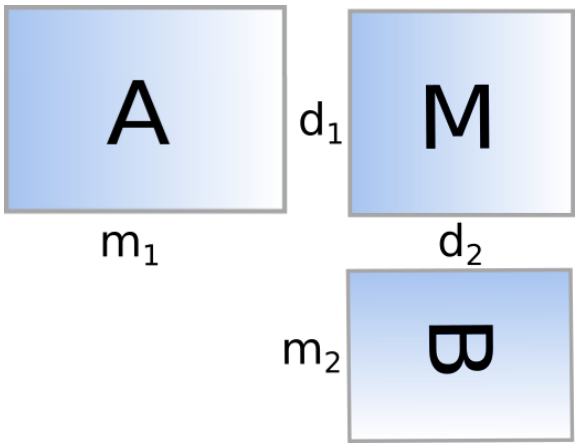


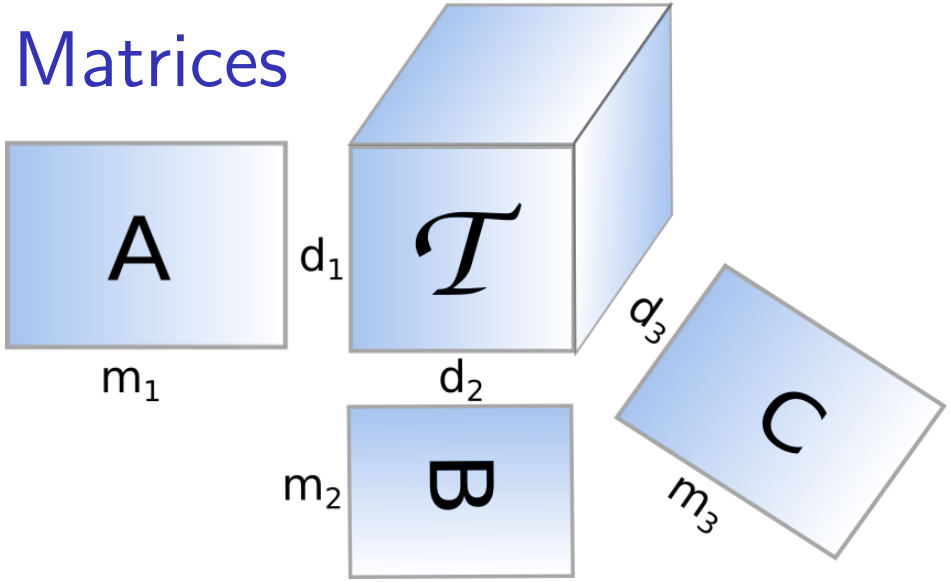
$\mathbf{AMB}^\top \in \mathbb{R}^{m_1 \times m_2}$

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

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





$\mathbf{A}\mathbf{M}\mathbf{B}^\top \in \mathbb{R}^{m_1 \times m_2}$

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

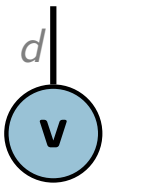
ex: If $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and $\mathbf{A} \in \mathbb{R}^{m_1 \times d_1}, \mathbf{B} \in \mathbb{R}^{m_2 \times d_2}, \mathbf{C} \in \mathbb{R}^{m_3 \times d_3}$, then $\mathcal{T} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ 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 $i_1 \in [d_1], i_2 \in [m_2], i_3 \in [d_3]$.

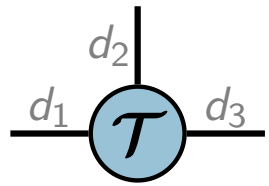
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

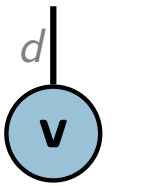
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

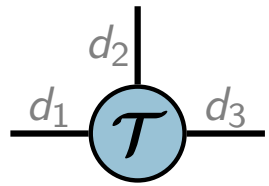
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

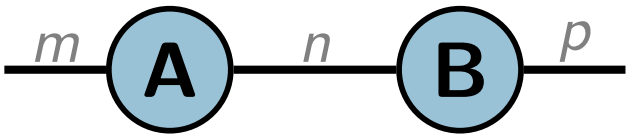
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

Edge $\equiv$ contraction

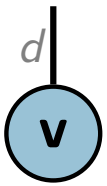
Matrix product:



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

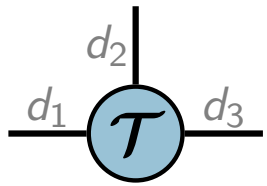
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

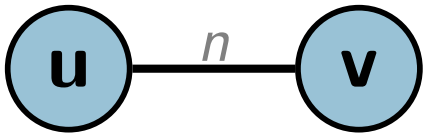
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

Edge $\equiv$ contraction

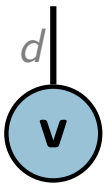
Inner product:



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

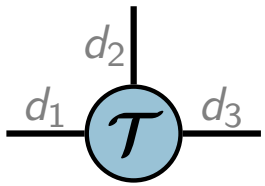
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

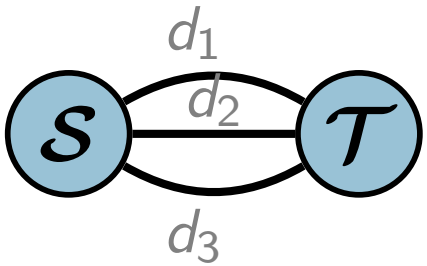
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

Edge $\equiv$ contraction

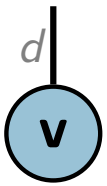
Inner product between tensors:



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

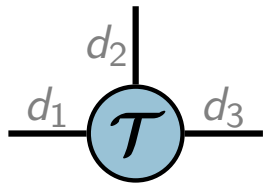
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

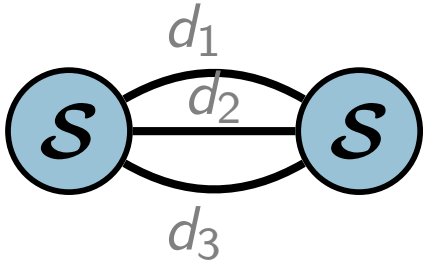
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

Edge $\equiv$ contraction

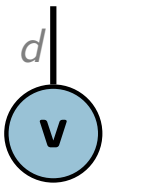
Frobenius norm of a tensor:



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

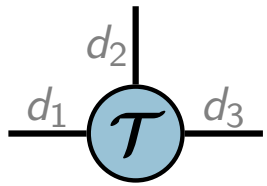
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

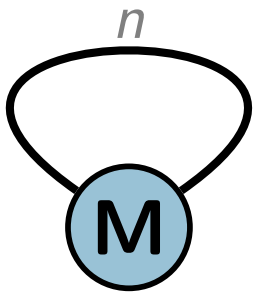
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

Edge $\equiv$ contraction

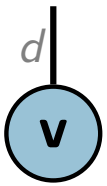
Trace of an $n \times n$ matrix:



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

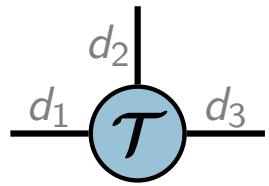
Tensor Networks

Degree of a node $\equiv$ order of tensor



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

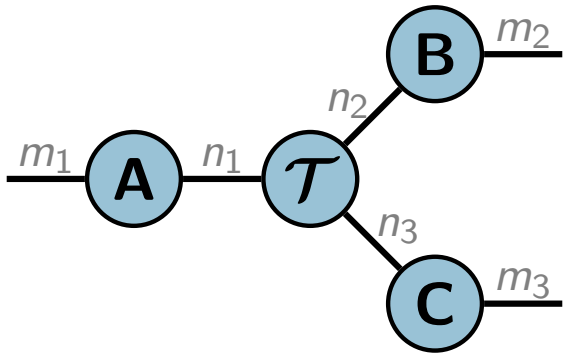
$\mathbf{M} \in \mathbb{R}^{m \times n}$



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

Edge $\equiv$ contraction

Tensor times matrices:



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

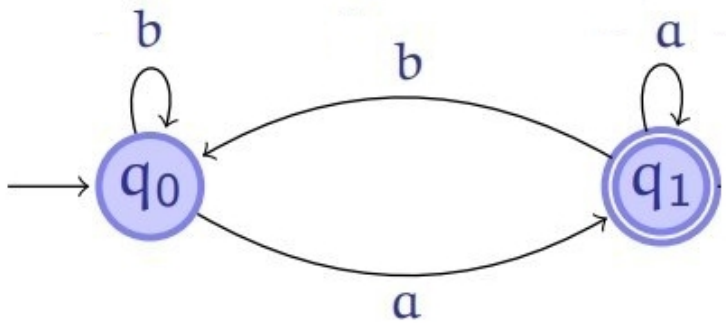
Weighted Automata

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)

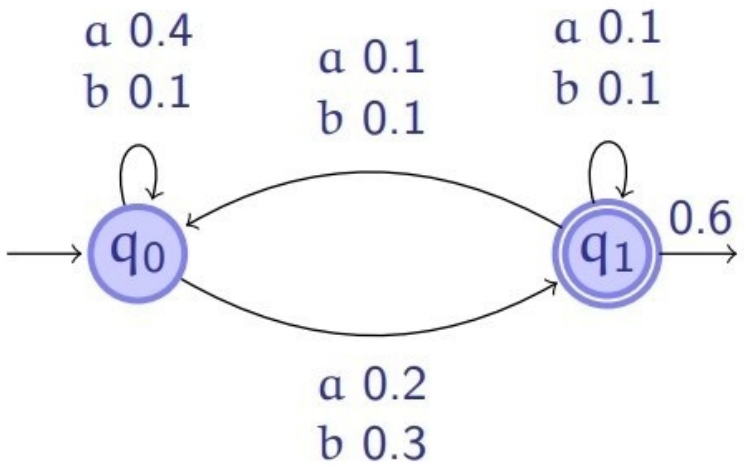
- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. $abba$), λ the empty string.
- Recall: a Deterministic Finite Automaton (DFA) recognizes a *language* (subset of Σ^*).



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

Weighted Automata: States and Weighted Transitions

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

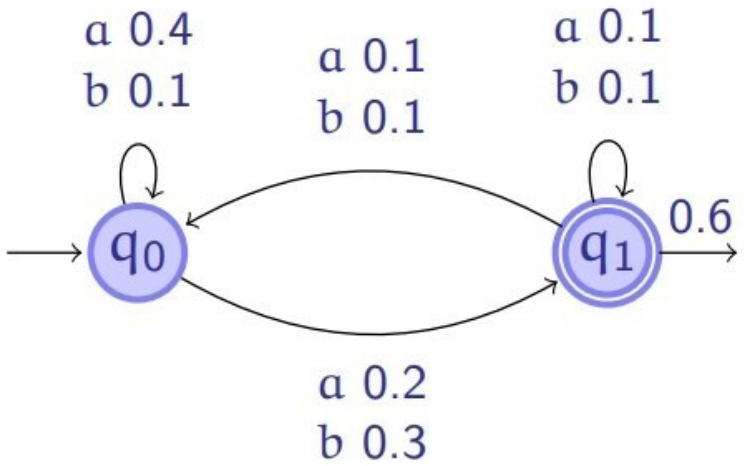


$$f(ab) = 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084$$

image 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 $\sigma \in \Sigma$

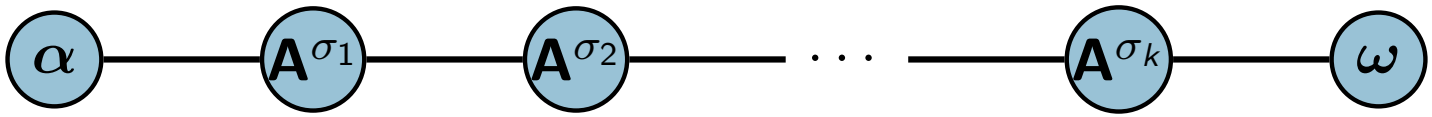
String Weighted Automata (WA)

- 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

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



Spectral Learning

Hankel matrix

- Weighted Finite Automata:
  - Σ a finite alphabet of size d (e.g. $\{a, b\}$)
  - Σ^* strings on Σ (e.g. $abba$)
  - A WFA computes a function $f : \Sigma^* \to \mathbb{R}$:

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

Hankel matrix

- Weighted Finite Automata:
  - Σ a finite alphabet of size d (e.g. $\{a, b\}$)
  - Σ^* strings on Σ (e.g. $abba$)
  - A WFA computes a function $f : \Sigma^* \to \mathbb{R}$:

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

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

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

Spectral Learning of WFAs

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: 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]:

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

Spectral Learning of WFAs

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: 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]:

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

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

Spectral Learning of WFA

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

Spectral Learning of WFA

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

Spectral Learning of WFA

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

$(\mathbf{h}_{\mathcal{P}})_u = f(u)$,

Spectral Learning of WFA

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

$(\mathbf{h}_\mathcal{P})_u = f(u)$, $(\mathbf{h}_\mathcal{S})_v = f(v)$,

Spectral Learning of WFA

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

$(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$

Spectral Learning of WFA

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

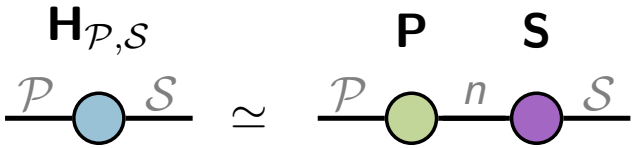
$(\mathbf{h}_{\mathcal{P}})_u = f(u)$, $(\mathbf{h}_{\mathcal{S}})_v = f(v)$, $(\mathbf{H}_{\mathcal{P},\mathcal{S}})_{u,v} = f(uv)$ and $(\boldsymbol{\mathcal{H}}_{\mathcal{P},\Sigma,\mathcal{S}})_{u,\sigma,v} = f(u\sigma v)$

Spectral Learning of WFA

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 WFA parameters $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$:



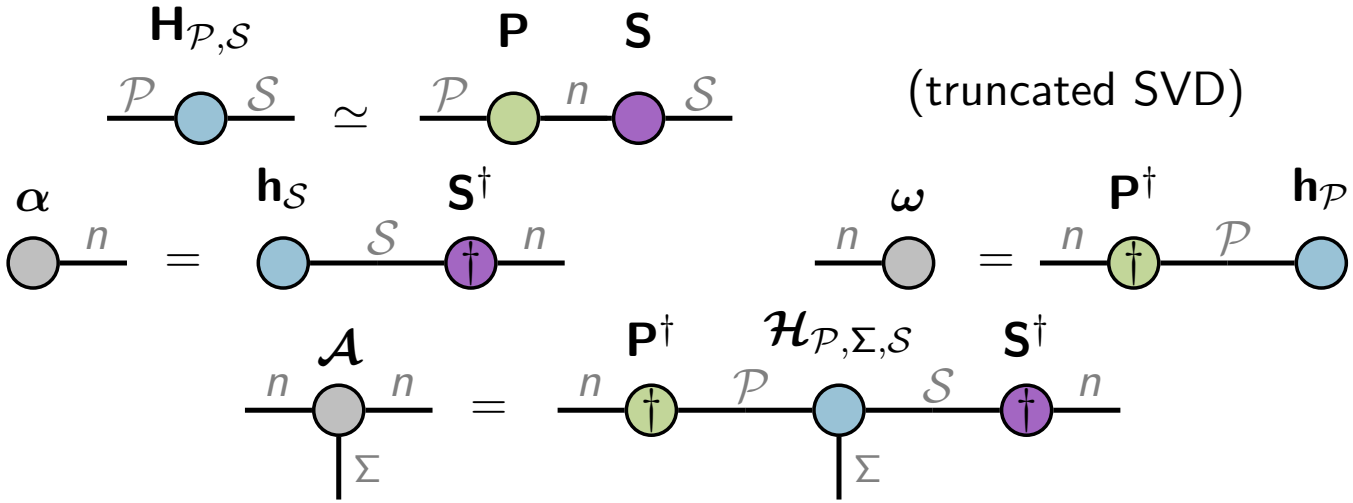
(truncated SVD)

Spectral Learning of WFA

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 WFA parameters $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$:

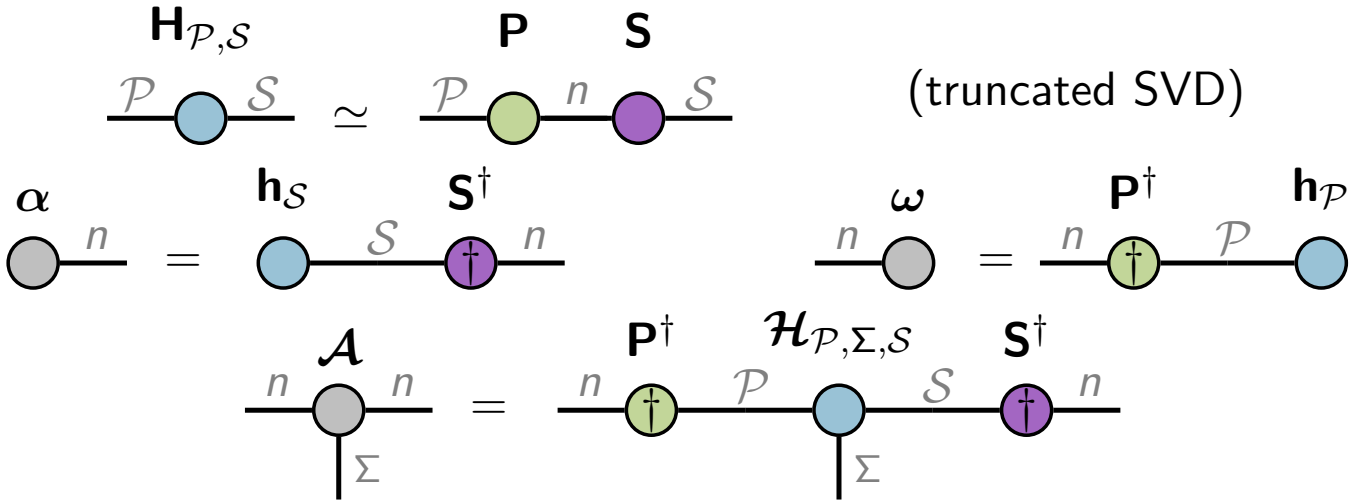


Spectral Learning of WFA

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 WFA parameters $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$:



→ Efficient and consistent learning algorithms for 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 WFA 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 WFA computing f.

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

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

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

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

then the spectral learning algorithm returns a WFA computing f.

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

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

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., e.g., PhD thesis of B. Balle]

Estimating Hankel matrices

How to estimate the Hankel matrices from data?

- Language modeling. If f is a distribution over Σ^*: empirical frequencies.

$$S = \left\{ \begin{matrix} aa, ab, aa, b, \\ b, aba, a, bb \end{matrix} \right\} \quad \rightarrow \quad \hat{\mathbf{H}} = \begin{matrix} \\ \lambda \\ a \\ b \end{matrix} \begin{matrix} \begin{matrix} a & b & aa & ba \end{matrix} \\ \begin{bmatrix} 1/8 & 2/8 & 2/8 & 0 \\ 2/8 & 1/8 & 0 & 1/8 \\ 0 & 1/8 & 0 & 0 \end{bmatrix} \end{matrix}$$

Estimating Hankel matrices

How to estimate the Hankel matrices from data?

- Language modeling. If f is a distribution over Σ^*: empirical frequencies.

$$S = \left\{ \begin{matrix} aa, ab, aa, b, \\ b, aba, a, bb \end{matrix} \right\} \quad \rightarrow \quad \hat{\mathbf{H}} = \begin{matrix} \\ \lambda \\ a \\ b \end{matrix} \begin{matrix} \begin{matrix} a & b & aa & ba \end{matrix} \\ \begin{bmatrix} 1/8 & 2/8 & 2/8 & 0 \\ 2/8 & 1/8 & 0 & 1/8 \\ 0 & 1/8 & 0 & 0 \end{bmatrix} \end{matrix}$$

- Regression. What if f is an arbitrary function?

$$S = \left\{ \begin{matrix} (aa,\ 0.2),\ (ab,\ -0.5),\ (b,\ 1.2), \\ (aba,\ 1.1),\ (a,\ -2),\ (bb,\ 0.4) \end{matrix} \right\} \quad \rightarrow \quad \hat{\mathbf{H}} = \begin{matrix} \\ \lambda \\ a \\ b \end{matrix} \begin{matrix} \begin{matrix} a & b & aa & ba \end{matrix} \\ \begin{bmatrix} -2 & 1.2 & 0.2 & ? \\ 0.2 & -0.5 & ? & 1.1 \\ ? & 0.4 & ? & ? \end{bmatrix} \end{matrix}$$

Estimating Hankel matrices

How to estimate the Hankel matrices from data?

- Language modeling. If f is a distribution over Σ^*: empirical frequencies.

$$S = \left\{ \begin{matrix} aa, ab, aa, b, \\ b, aba, a, bb \end{matrix} \right\} \quad \rightarrow \quad \hat{\mathbf{H}} = \begin{matrix} \\ \lambda \\ a \\ b \end{matrix} \begin{matrix} \begin{matrix} a & b & aa & ba \end{matrix} \\ \begin{bmatrix} 1/8 & 2/8 & 2/8 & 0 \\ 2/8 & 1/8 & 0 & 1/8 \\ 0 & 1/8 & 0 & 0 \end{bmatrix} \end{matrix}$$

- Regression. What if f is an arbitrary function?

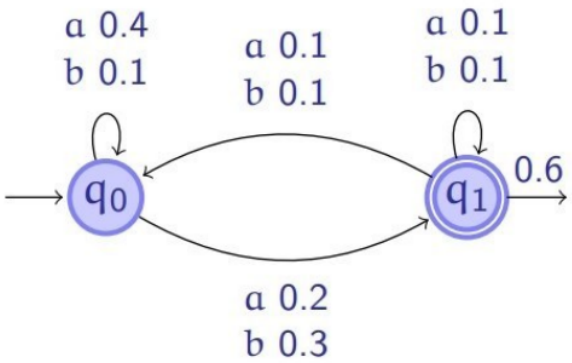
$$S = \left\{ \begin{matrix} (aa,\ 0.2),\ (ab,\ -0.5),\ (b,\ 1.2), \\ (aba,\ 1.1),\ (a,\ -2),\ (bb,\ 0.4) \end{matrix} \right\} \quad \rightarrow \quad \hat{\mathbf{H}} = \begin{matrix} \\ \lambda \\ a \\ b \end{matrix} \begin{matrix} \begin{matrix} a & b & aa & ba \end{matrix} \\ \begin{bmatrix} -2 & 1.2 & 0.2 & ? \\ 0.2 & -0.5 & ? & 1.1 \\ ? & 0.4 & ? & ? \end{bmatrix} \end{matrix}$$

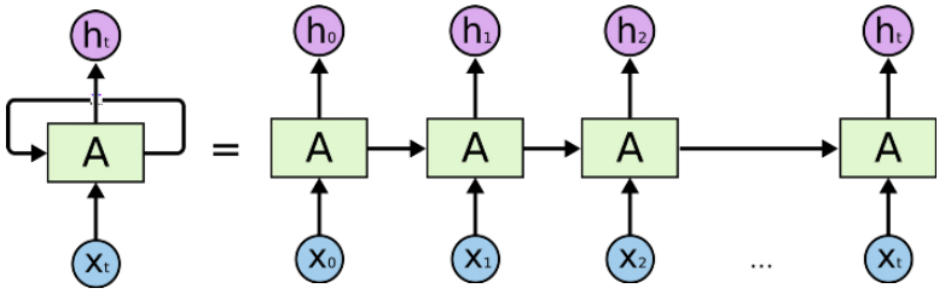
$\hookrightarrow$ Two steps [Balle & Mohri, NeurIPS 2012]:

1. Structured matrix completion to infer missing entries
2. Spectral learning algorithm

Weighted Automata Vs. RNNs

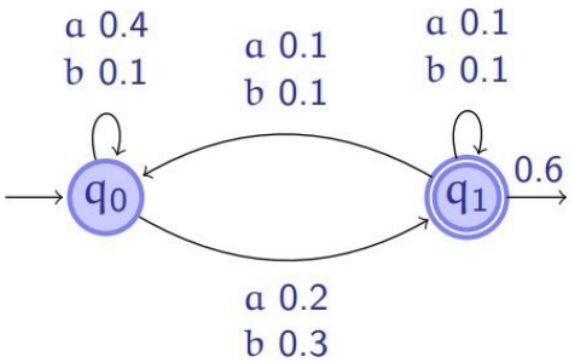
Weighted Automata Vs. Recurrent Neural Networks

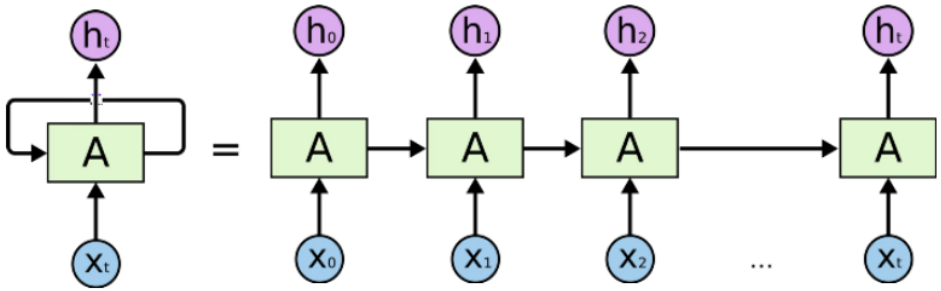




- Weighted automata are "robust" models for sequence data
- Recurrent neural networks can also deal with sequence data
  - ⊕ Remarkably expressive models, impressive results in speech and audio recognition
  - ⊖ Less tractable than WA, limited understanding of their inner working
- Connections between WFA 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., 2018,2019], [Ayache et al., 2018]
  - Can we combine FSMs with WFA? [Rastogi et al., 2016], [Dyer et al., 2016]
  - To which extent Weighted Automata are linear RNNs?

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 WFA 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., 2018,2019], [Ayache et al., 2018]
  - Can we combine FSMs with WFA? [Rastogi et al., 2016], [Dyer et al., 2016]
  - To which extent Weighted Automata are linear RNNs?
  - Can we extend WFAs to input sequences of continuous vectors?

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]:

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

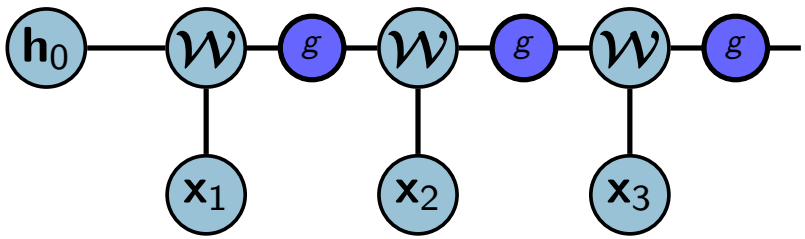
$\rightarrow$ order 2 multiplicative interactions: $[\mathbf{h}_t]_i = g\left(\sum_{j,k} \mathcal{W}_{ijk}[\mathbf{x}_t]_j[\mathbf{h}_{t-1}]_k\right)$.

$\hookrightarrow$ (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

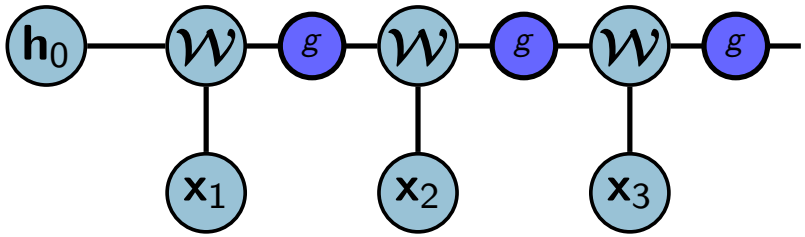
$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$



Weighted Automata and Recurrent Neural Networks

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

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

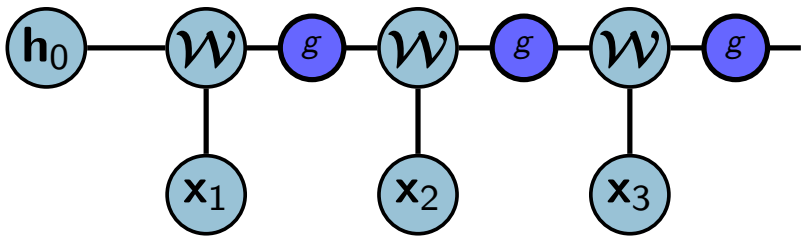


- The computation of a weighted automaton is very similar!

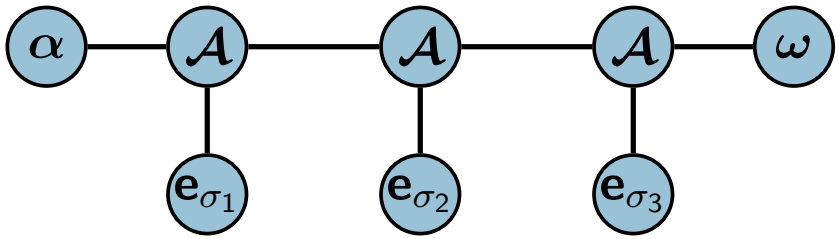
Weighted Automata and Recurrent Neural Networks

- 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 $\mathcal{A}_{:,\sigma,:} = \mathbf{A}^\sigma$)

WFAs $\equiv$ linear 2-RNNs

Theorem

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

WFAs $\equiv$ linear 2-RNNs

Theorem

*WFAs 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 WFAs?

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

Continuous WFA / linear 2-RNN

Definition

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

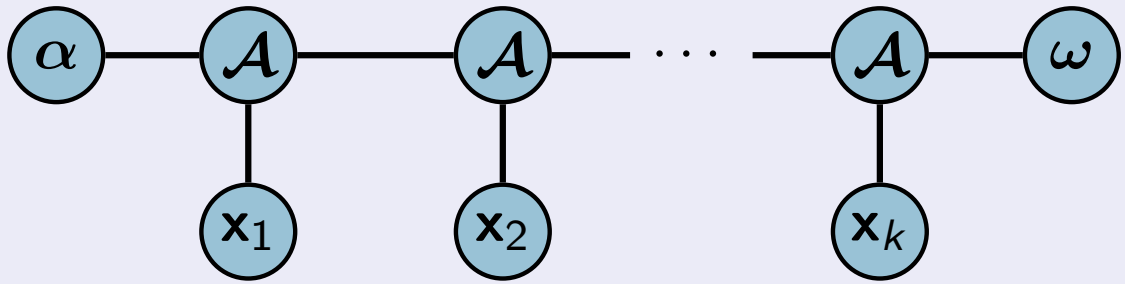
$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector

$\boldsymbol{\omega} \in \mathbb{R}^n$ 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

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



WFAs $\equiv$ linear 2-RNNs

Theorem

WFAs 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 WFAs?

$\hookrightarrow$ We can extend the definition of WFAs to continuous vectors!

WFAs $\equiv$ linear 2-RNNs

Theorem

WFAs 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 WFAs?

$\hookrightarrow$ We can extend the definition of WFAs to continuous vectors!

- Can we learn linear 2-RNNs from data?
  - $\star$ Over sequences of discrete symbols?

  $\hookrightarrow$ Yes: spectral learning of WFA
  - $\star$ Over sequences of continuous vectors?

  $\hookrightarrow$ Yes: technical contribution of [GR, T. Li, D. Precup, AISTATS'19]

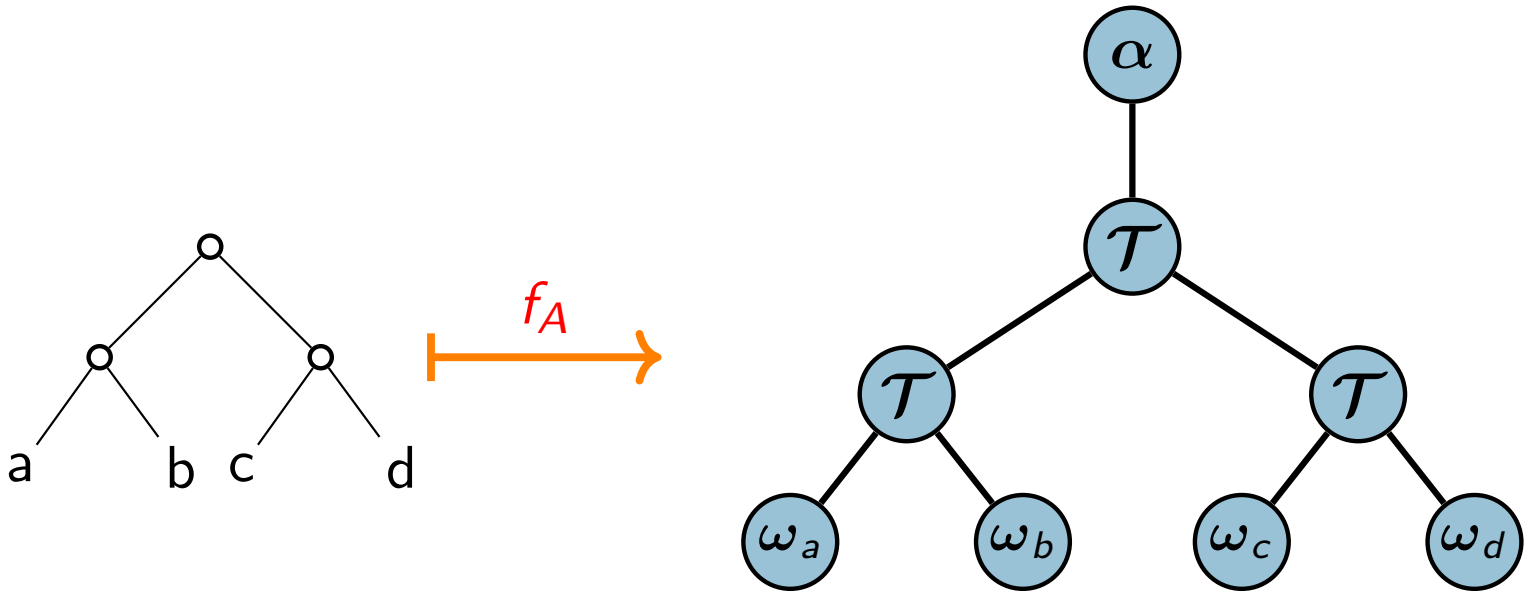
Future directions

- 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.

Weighted Tree Automata

- A weighted tree automaton (WTA) is a tuple $A = \langle \boldsymbol{\alpha}, \boldsymbol{\mathcal{T}}, \{\boldsymbol{\omega}_\sigma\}_{\sigma \in \Sigma} \rangle$

$$\boldsymbol{\alpha} \in \mathbb{R}^n : \text{ vector of initial weights}$$
$$\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n \times n \times n} : \text{ tensor of transition weights}$$
$$\boldsymbol{\omega}_\sigma \in \mathbb{R}^n : \text{ vector of final weights associated with } \sigma \in \Sigma$$



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.
  - (Using the spectral learning for WTA to extract PCFG from RNN (i.e., extending [Barbot et al., ICGI 2021] to stochastic setting or using RNNs with ordered neurons [Shen et al., ICLR 2018]))

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.
  - (Using the spectral learning for WTA to extract PCFG from RNN (i.e., extending [Barbot et al., ICGI 2021] to stochastic setting or using RNNs with ordered neurons [Shen et al., ICLR 2018]))
- 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.
  - (Using the spectral learning for WTA to extract PCFG from RNN (i.e., extending [Barbot et al., ICGI 2021] to stochastic setting or using RNNs with ordered neurons [Shen et al., ICLR 2018]))
- What do linear counterparts of neural sequential models (LSTMs, bi-directionnal RNNs, etc.) correspond to?
- Spectral initialization of RNNs (*ongoing work of Maude Lizaire*).

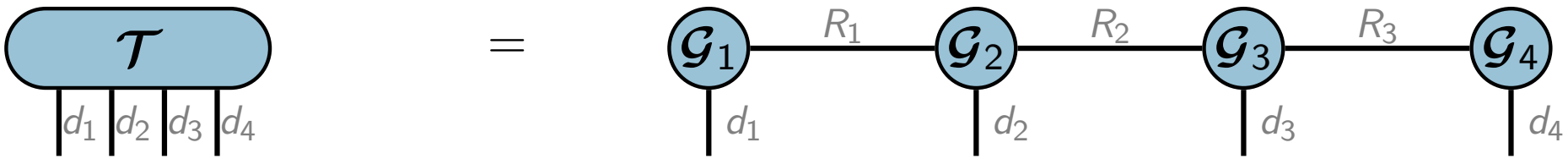
Future directions

- 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.
  - (Using the spectral learning for WTA to extract PCFG from RNN (i.e., extending [Barbot et al., ICGI 2021] to stochastic setting or using RNNs with ordered neurons [Shen et al., ICLR 2018]))
- 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 WFA and RNNs (e.g. Multiplicative interaction RNNs are special case of 2nd order RNNs, formal hierarchy of higher-order RNNs)...

Tensor Networks and Weighted Automata

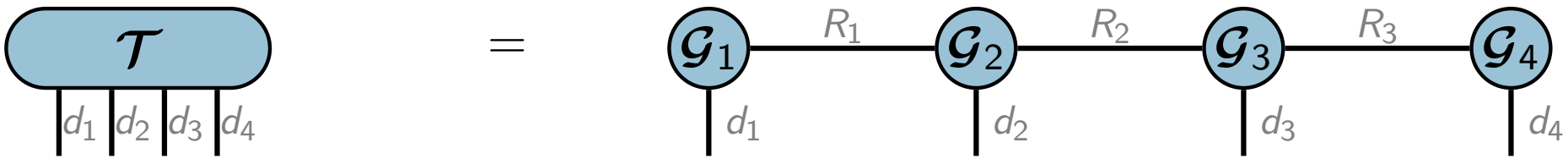
Tensor Train / Matrix Product State (MPS) decomposition

- TT/MPS decomposition [Oseledets (2011), Fannes et al. (1992)]:



Tensor Train / Matrix Product State (MPS) decomposition

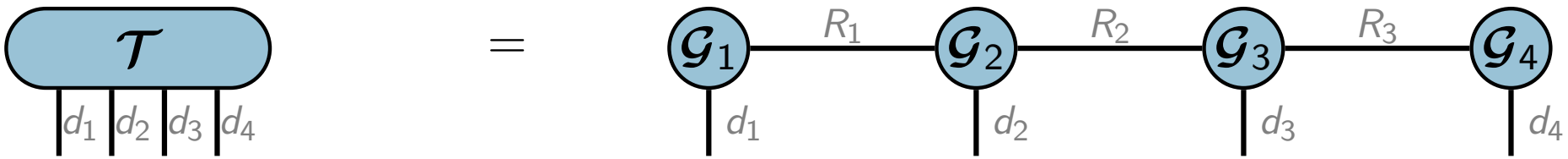
- TT/MPS decomposition [Oseledets (2011), Fannes et al. (1992)]:



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

Tensor Train / Matrix Product State (MPS) decomposition

- TT/MPS decomposition [Oseledets (2011), Fannes et al. (1992)]:

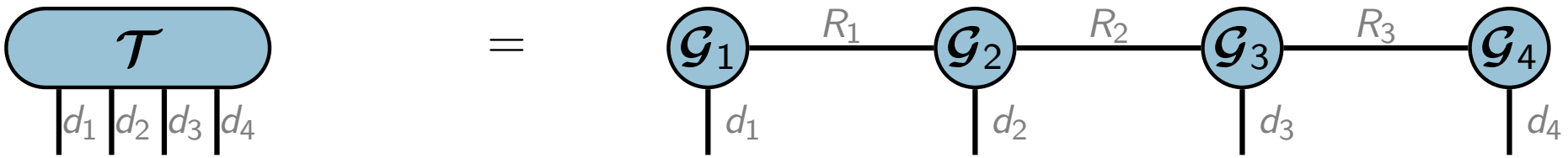


$\Rightarrow$ $d_1 R_1 + R_1 d_2 R_2 + R_2 d_2 R_3 + R_3 d_4$ parameters instead of $d_1 d_2 d_3 d_4$.

- 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

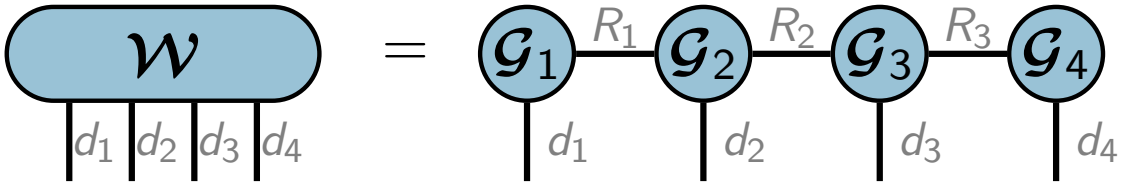
- TT/MPS decomposition [Oseledets (2011), Fannes et al. (1992)]:



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

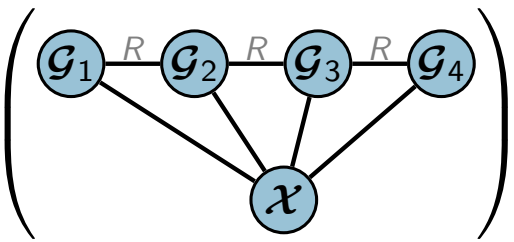
- If the ranks are all the same ($R_1 = R_2 = \cdots = R$), can represent a vector of size 2^n with $\mathcal{O}\left(nR^2\right)$ parameters!
- We can also efficiently perform operations on MPS tensors:
  - Inner product, sum, component-wise product, ... all in time linear in n for vectors of size d^n.

Tensor Train / Matrix Product States

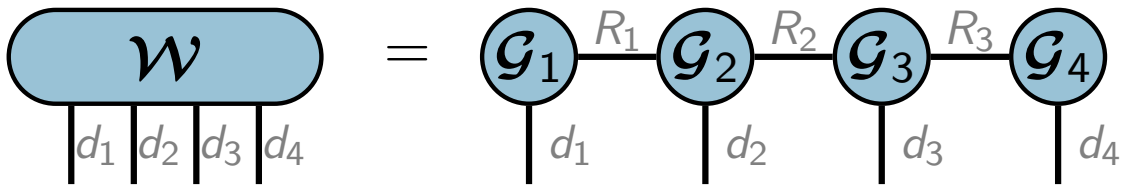


- We can parameterize linear classification models with MPS [Stoudenmire & Schwab, 2016]:

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

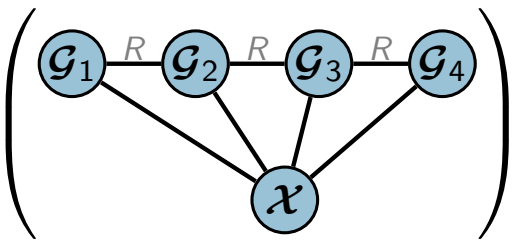


Tensor Train / Matrix Product States



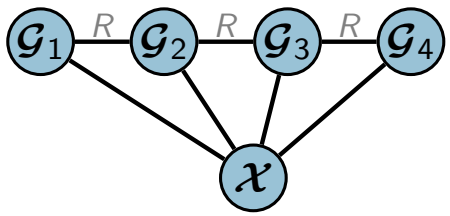
- We can parameterize linear classification models with MPS [Stoudenmire & Schwab, 2016]:

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

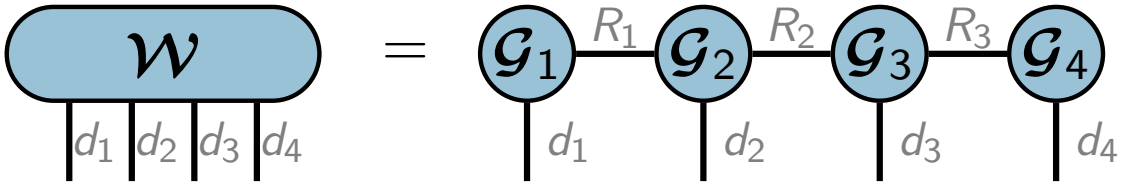


- We can also model probability distributions with MPS [Han et al., 2018]:

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

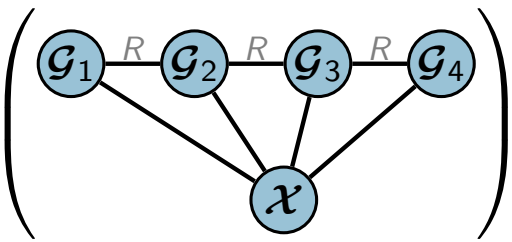


Tensor Train / Matrix Product States



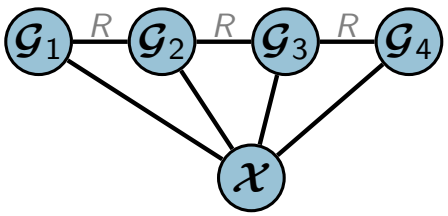
- We can parameterize linear classification models with MPS [Stoudenmire & Schwab, 2016]:

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

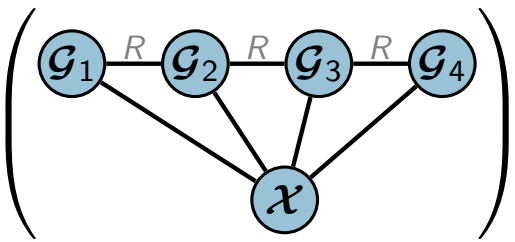


- We can also model probability distributions with MPS [Han et al., 2018]:

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



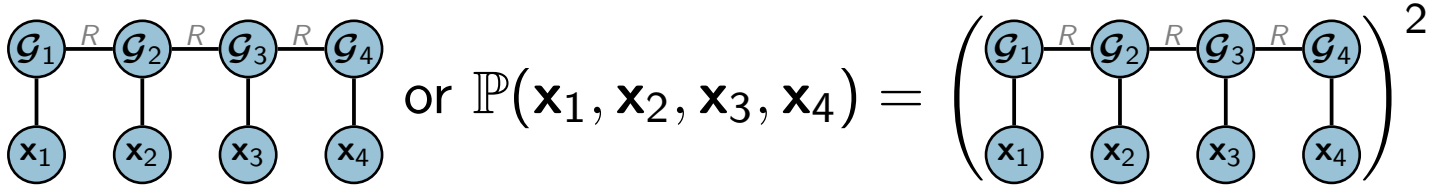
or $\quad \mathbb{P}(\boldsymbol{\mathcal{X}}) =$



MPS for sequence modeling

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

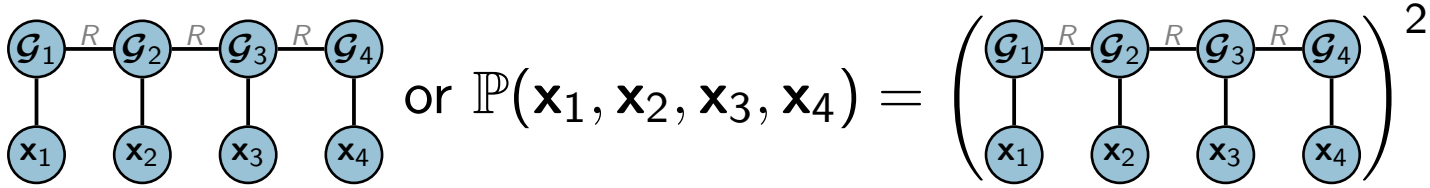
$$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \text{or } \mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$$



MPS for sequence modeling

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

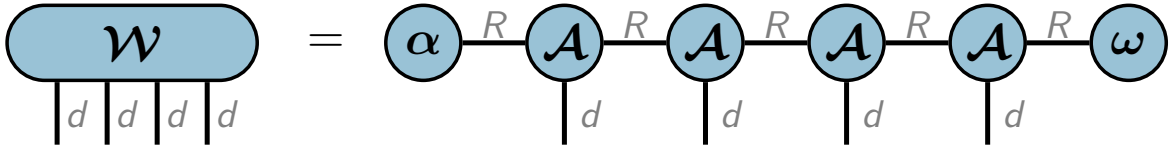
$$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \text{or } \mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) =$$



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

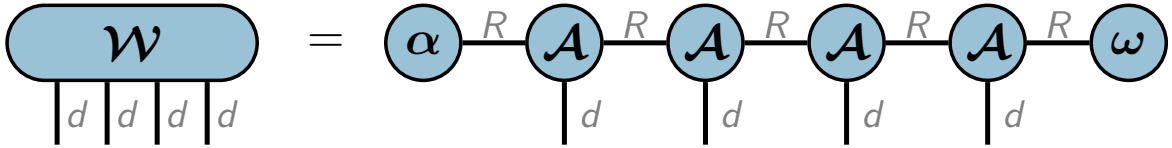
Uniform MPS

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



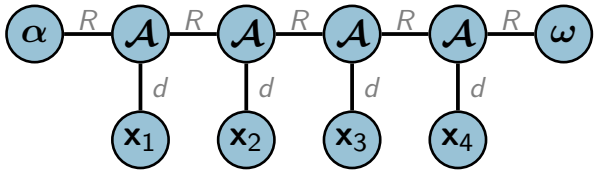
Uniform MPS

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

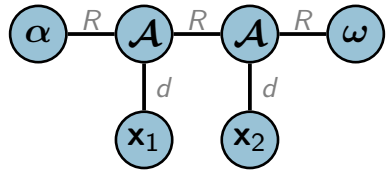


- With uMPS, we can model functions and distributions over variable length sequences:

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

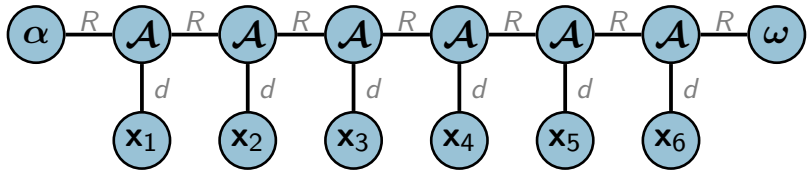


$$, \mathbb{P}(\mathbf{x}_1, \mathbf{x}_2) =$$



$$,$$

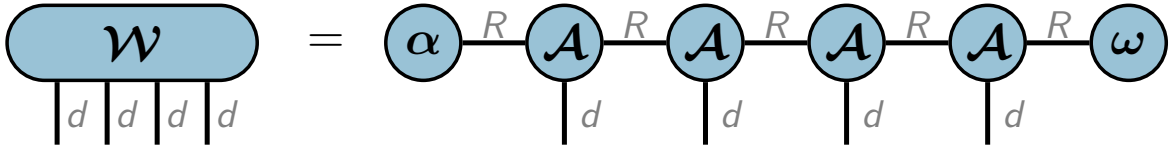
$$\mathbb{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6) =$$



$$, \cdots$$

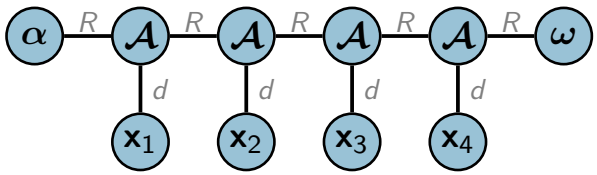
Uniform MPS

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

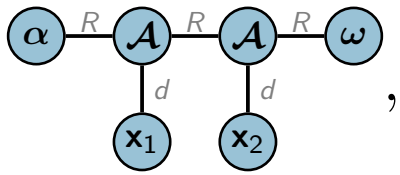


- With uMPS, we can model functions and distributions over variable length sequences:

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

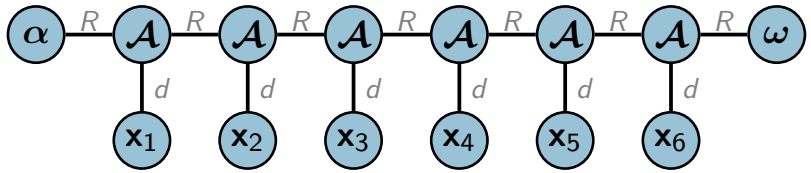


$$, \mathbb{P}(\mathbf{x}_1, \mathbf{x}_2) =$$



$$,$$

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



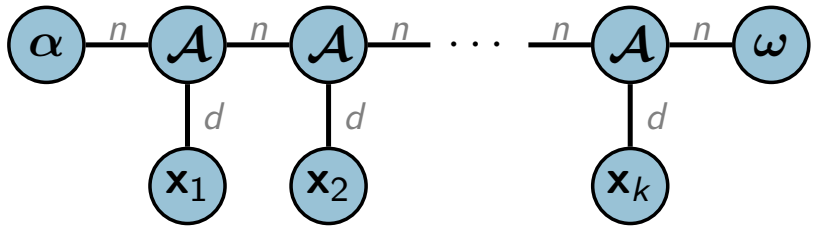
$$, \cdots$$

$\hookrightarrow$ Nothing else than the continuous WFA (aka linear 2-RNN) 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:

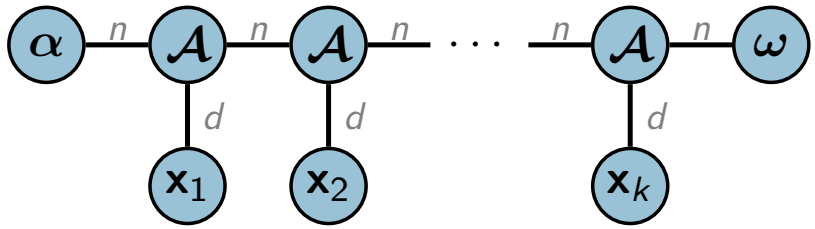
$$f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) =$$



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:

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

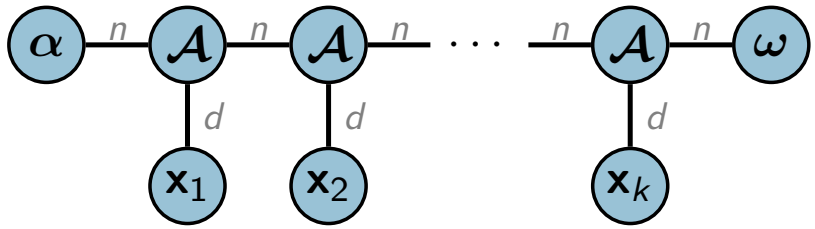


- If the inputs are one-hot encoding, uMPS $\equiv$ Weighted Automata
  - $\hookrightarrow$ If the probability of a sequence is $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)^2 \equiv$ Quadratic weighted automata (Bailly, 2011) / MPS from quantum physics

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:

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

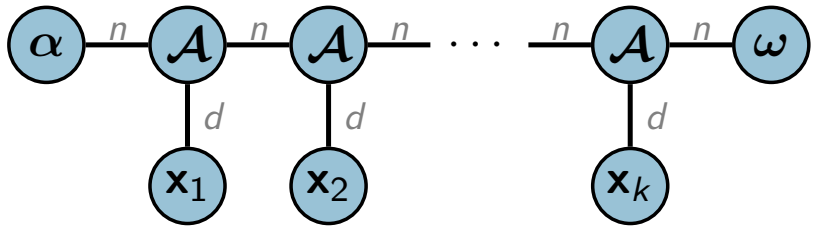


- If the inputs are one-hot encoding, uMPS $\equiv$ Weighted Automata
  - $\hookrightarrow$ If the probability of a sequence is $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)^2 \equiv$ Quadratic weighted automata (Bailly, 2011) / MPS from quantum physics
- Linear second order RNNs $\equiv$ uMPS

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
  - $\hookrightarrow$ 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 (ongoing 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).

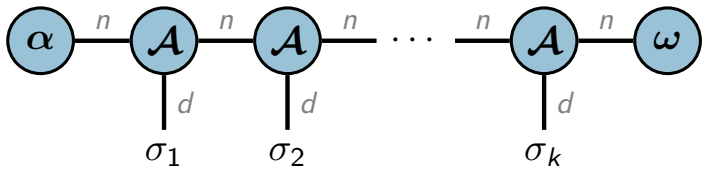
- Scale up learning to very large state spaces (ongoing work of Jacob Miller).
- Training uMPS/WFA with word embeddings for language modeling (ongoing work of Jacob Miller and Raphaëlle Tihon).

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 WFA computes a function $f : \Sigma^* \to \mathbb{R}$:

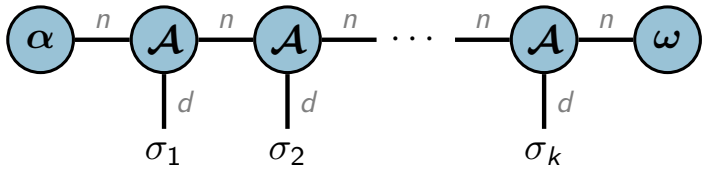
$$f(\sigma_1 \cdots \sigma_k) =$$



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 WFA 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 $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

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

A closer look at the Hankel matrix of a WFA

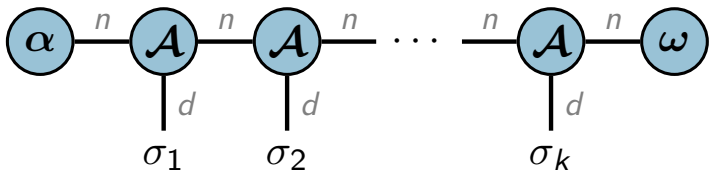
- Let $f : \Sigma^* \to \mathbb{R}$ be the function computed by a WFA $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$.
- Define the ℓth order Hankel tensor $\boldsymbol{\mathcal{H}}^{(\ell)} \in \mathbb{R}^{\Sigma \times \Sigma \times \cdots \times \Sigma}$ by

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

A closer look at the Hankel matrix of a WFA

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

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

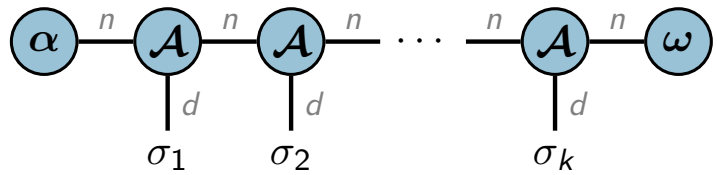


for all $\sigma_1, \cdots \sigma_\ell \in \Sigma$

A closer look at the Hankel matrix of a WFA

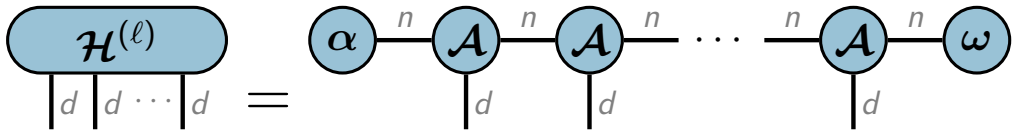
- Let $f : \Sigma^* \to \mathbb{R}$ be the function computed by a WFA $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$.
- Define the ℓth order Hankel tensor $\boldsymbol{\mathcal{H}}^{(\ell)} \in \mathbb{R}^{\Sigma \times \Sigma \times \cdots \times \Sigma}$ by

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



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

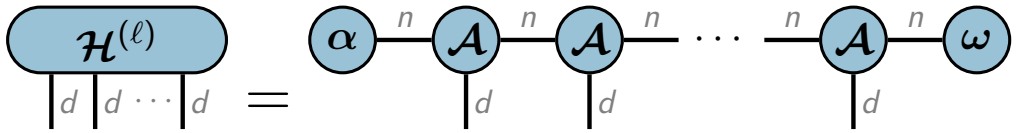
- For each ℓ, the tensor $\boldsymbol{\mathcal{H}}^{(\ell)}$ has low uniform MPS rank:



(2)

A closer look at the Hankel matrix of a uMPS

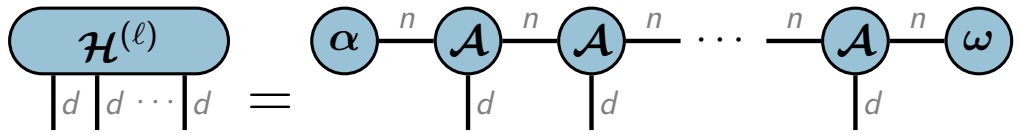
- For each ℓ, the tensor $\mathcal{H}^{(\ell)}$ (defined by $\mathcal{H}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell)$) has low uniform MPS rank:



(3)

A closer look at the Hankel matrix of a uMPS

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



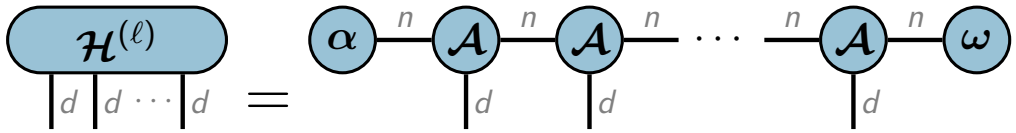
(3)

- 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 $\mathcal{H}^{(\ell)}$ (defined by $\mathcal{H}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell)$) 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	b	aa	ab	ba	bb	aaa	aab	⋯
a	$\mathcal{H}^{(2)}_{\Sigma\times\Sigma}$		$\mathcal{H}^{(3)}_{\Sigma\times\Sigma^2}$				$\mathcal{H}^{(4)}_{\Sigma\times\Sigma^3}$		⋯
b									
aa	$\mathcal{H}^{(3)}_{\Sigma^2\times\Sigma}$		$\mathcal{H}^{(4)}_{\Sigma^2\times\Sigma^2}$				$\mathcal{H}^{(5)}_{\Sigma^2\times\Sigma^3}$		⋯
ab									
ba									
bb									
aaa	$\mathcal{H}^{(4)}_{\Sigma^3\times\Sigma}$		$\mathcal{H}^{(5)}_{\Sigma^3\times\Sigma^2}$				$\mathcal{H}^{(6)}_{\Sigma^3\times\Sigma^3}$		⋯
aab									
⋮	⋮		⋮				⋮		

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

  $\to$ f can be a probability distribution, a score function 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

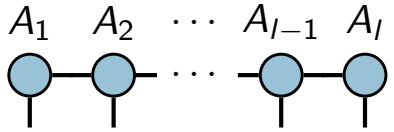
Spectral Learning $\equiv$ Conversion from MPS to uMPS

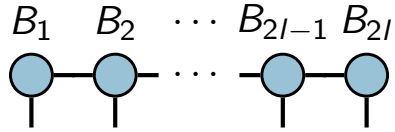
- Let $f : \Sigma^* \to \mathbb{R}$ be a function for which we have access to an MPS decomposition of the Hankel tensors $\mathcal{H}^{(\ell)}, \mathcal{H}^{(2\ell)}, \mathcal{H}^{(2\ell+1)}$.

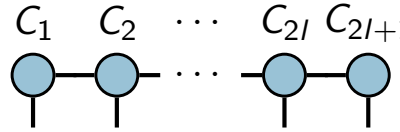
  $\to$ f can be a probability distribution, a score function or the wave function of a quantum system.

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

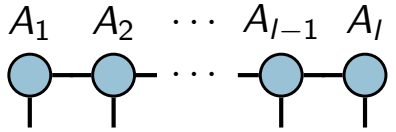
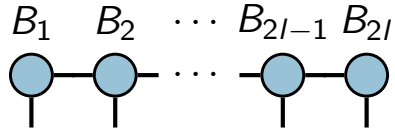
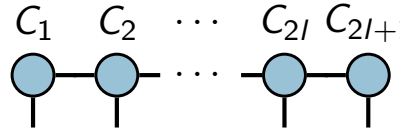
$\hookrightarrow$ if we know the value of f on words of length ℓ, 2ℓ and $2\ell + 1$, we can compute the value of f on sequences of arbitrary length!

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

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

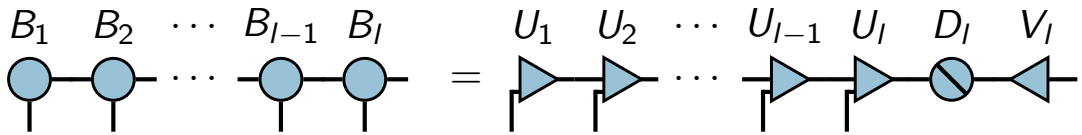
, $\mathcal{H}^{(2\ell+1)} =$ 

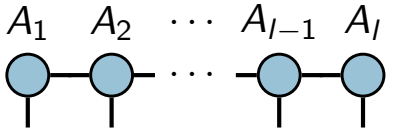
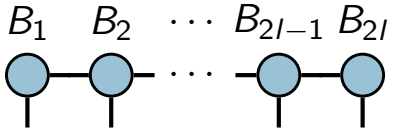
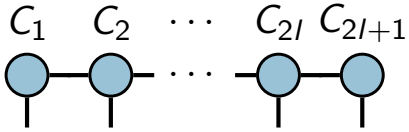
Output: uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ computing f

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

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

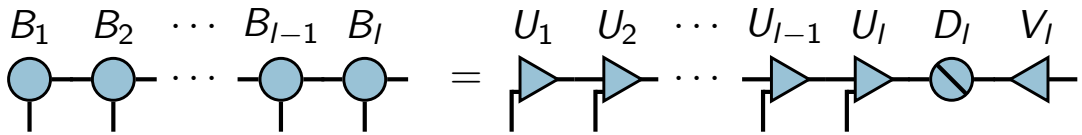
1. Left-orthonormalisation of $B_1, \cdots, B_\ell$ (first half of $\mathcal{H}^{(2\ell)}$)



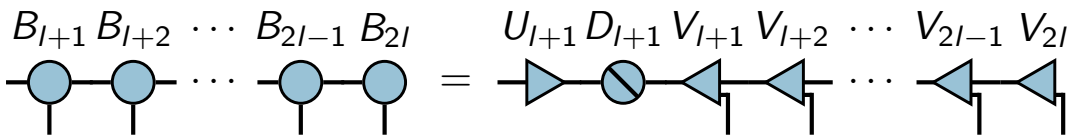
Input: $\mathcal{H}^{(\ell)} =$  , $\mathcal{H}^{(2\ell)} =$  , $\mathcal{H}^{(2\ell+1)} =$ 

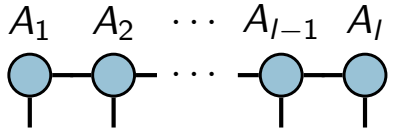
Output: uMPS $(\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\omega})$ computing f

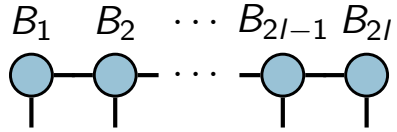
1. Left-orthonormalisation of $B_1, \cdots, B_\ell$ (first half of $\boldsymbol{\mathcal{H}}^{(2\ell)}$)

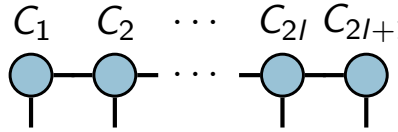


2. Right-orthonormalisation of $B_{\ell+1}, \cdots, B_{2\ell}$ (second half of $\boldsymbol{\mathcal{H}}^{(2\ell)}$)



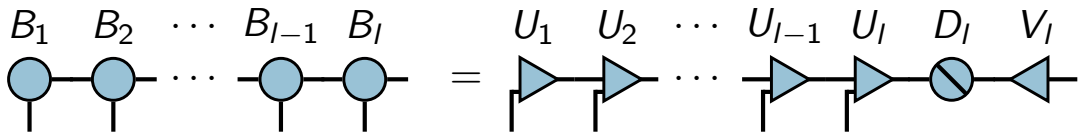
Input: $\mathcal{H}^{(\ell)} =$ 

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

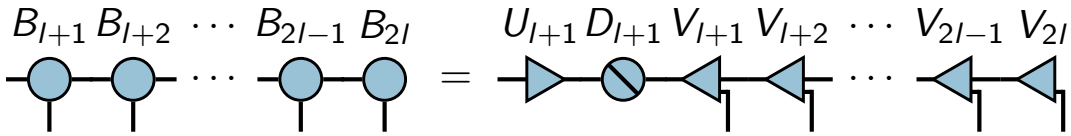
, $\mathcal{H}^{(2\ell+1)} =$ 

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

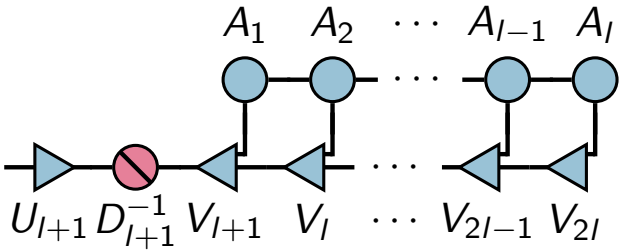
1. Left-orthonormalisation of $B_1, \cdots, B_\ell$ (first half of $\mathcal{H}^{(2\ell)}$)

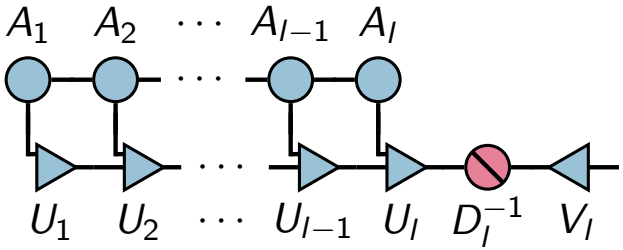


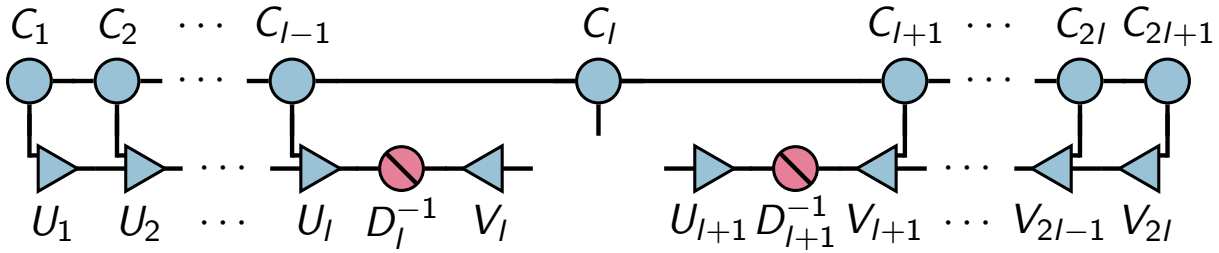
2. Right-orthonormalisation of $B_{\ell+1}, \cdots, B_{2\ell}$ (second half of $\mathcal{H}^{(2\ell)}$)



3. Computation of the uMPS parameters:

$\boldsymbol{\alpha} =$ 

$\boldsymbol{\omega} =$ 

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

Spectral Learning with Tensor Networks

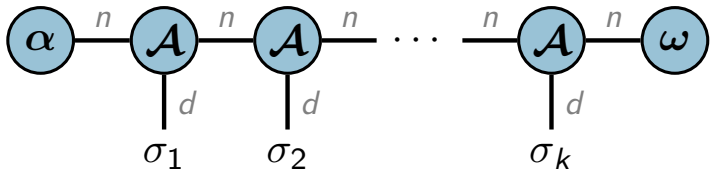
- More structure than low matrix rank in the Hankel matrix.
  - When $\mathcal{P} = \mathcal{S} = \Sigma^{\ell}$, the spectral learning algorithm can be performed efficiently in the MPS/TT format.

  $\hookrightarrow$ Time complexity is reduced from $\mathcal{O}\left(n|\Sigma|^{2\ell} + n^2|\Sigma|^{\ell+1}\right)$ to $\mathcal{O}\left(n^3\ell|\Sigma|\right)$.
  - To learn arbitrary function, we can fill the missing entries of the Hankel matrix using tensor completion instead of structured matrix completion techniques.

Learning linear 2-RNN over continuous inputs

- To learn f from data, we "just" need access to $\mathcal{H}^{(\ell)}, \mathcal{H}^{(2\ell)}, \mathcal{H}^{(2\ell+1)}$.
- Recall, in the discrete case:

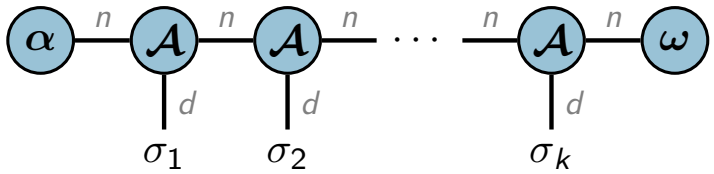
$$\mathcal{H}^{(\ell)}_{\sigma_1,\sigma_2,\cdots,\sigma_\ell} = f(\sigma_1\sigma_2\cdots\sigma_\ell) =$$



Learning linear 2-RNN over continuous inputs

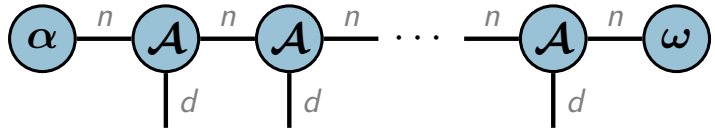
- To learn f from data, we "just" need access to $\mathcal{H}^{(\ell)}, \mathcal{H}^{(2\ell)}, \mathcal{H}^{(2\ell+1)}$.
- Recall, in the discrete case:

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



- In the continuous case, we define the Hankel tensor of a 2-RNN similarly:

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

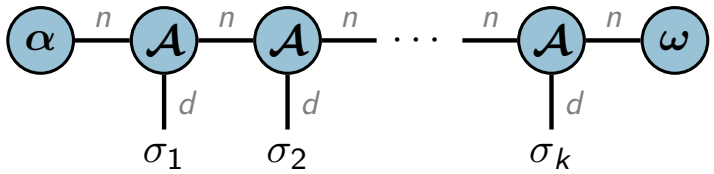


which implies that $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) = \langle \mathcal{H}^{(k)}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \cdots \otimes_k \mathbf{x}_k \rangle$

Learning linear 2-RNN over continuous inputs

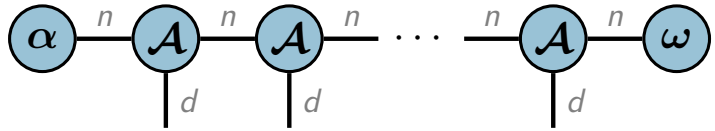
- To learn f from data, we "just" need access to $\mathcal{H}^{(\ell)}, \mathcal{H}^{(2\ell)}, \mathcal{H}^{(2\ell+1)}$.
- Recall, in the discrete case:

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



- In the continuous case, we define the Hankel tensor of a 2-RNN similarly:

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



which implies that $f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k) = \langle \mathcal{H}^{(k)}, \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \cdots \otimes_k \mathbf{x}_k \rangle$

$\hookrightarrow$ we can recover $\mathcal{H}^{(\ell)}$ from input-output examples of the form $((\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_\ell),\ y)$ using compressed sensing techniques! (related to tensor recovery from linear measurement recovery, quantum tomography and matrix sensing).

Future directions

- Extracting WFA from RNN defined over continuous inputs.
- Spectral learning of continuous WFA/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?