Spectral Learning of Weighted Automata and Connections with RNNs and Tensor Neworks.

Guillaume Rabusseau

Université de Montréal - Mila - CIFAR CCAI chair

April 1, 2020
MCQLL Meeting

Learning with Structured Data

Supervised Learning:

Learn $f : \mathcal{X} \to \mathcal{Y}$ from a sample $\{(x_1, y_1), \cdots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$.

Learning with Structured Data

Supervised Learning:

Learn $f : \mathcal{X} \to \mathcal{Y}$ from a sample $\{(x_1, y_1), \cdots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$.

- Classical learning algorithms assume $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^p$.
- How to handle input/output structured data?

Learning with Structured Data

Supervised Learning:

Learn $f : \mathcal{X} \to \mathcal{Y}$ from a sample $\{(x_1, y_1), \cdots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$.

- Classical learning algorithms assume $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^p$.
- How to handle input/output structured data?
  - Tensor structured data: Images, videos, spatio-temporal data, ...

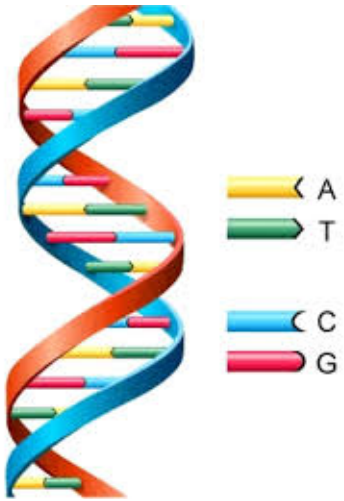
$\in \mathbb{R}^{32\times32\times3} \simeq \mathbb{R}^{3072}$

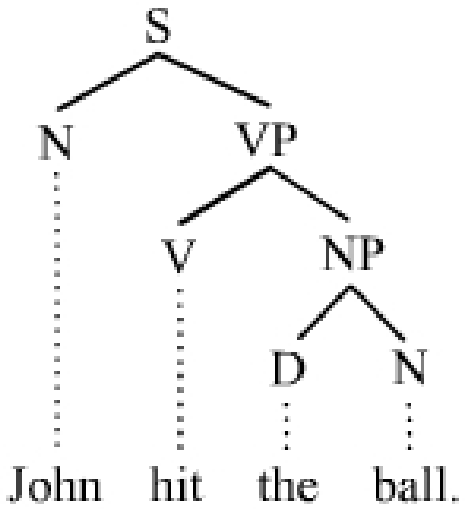
Learning with Structured Data

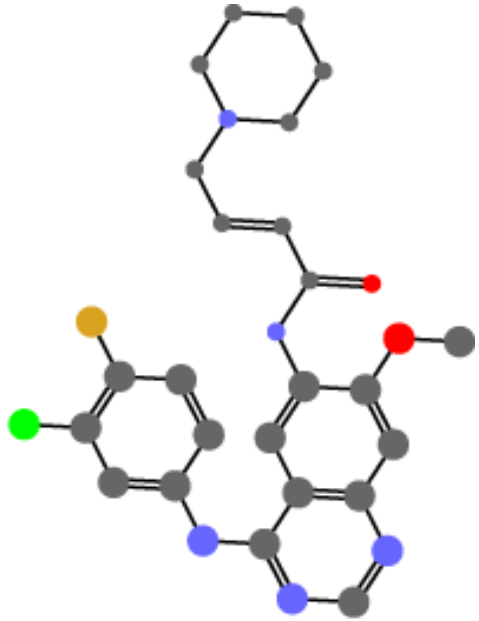
Supervised Learning:

Learn $f : \mathcal{X} \to \mathcal{Y}$ from a sample $\{(x_1, y_1), \cdots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$.

- Classical learning algorithms assume $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^p$.
- How to handle input/output structured data?
  - Tensor structured data: Images, videos, spatio-temporal data, ...
  - Discrete structured data: strings, trees, graphs, ...







Learning with Structured Data

Supervised Learning:

Learn $f : \mathcal{X} \to \mathcal{Y}$ from a sample $\{(x_1, y_1), \cdots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$.

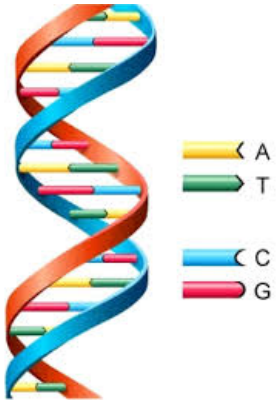
- Classical learning algorithms assume $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^p$.
- How to handle input/output structured data?
  - Tensor structured data: Images, videos, spatio-temporal data, ...
  - Discrete structured data: strings, trees, graphs, ...

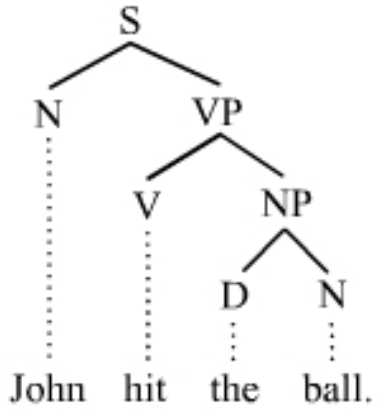
- In both cases, one can leverage linear and tensor algebra to design learning algorithms.

Outline

Introduction

- How can one learn with structured objects such as strings and trees?

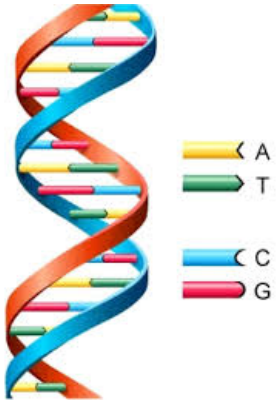


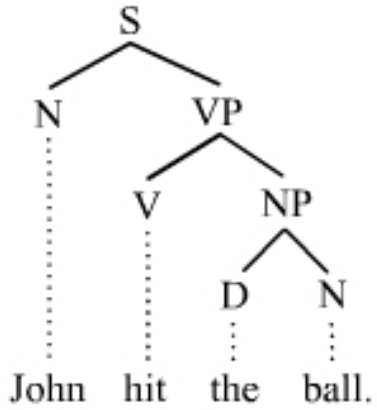


- Intersection of Theoretical Computer Science and Machine Learning...

Introduction

- How can one learn with structured objects such as strings and trees?



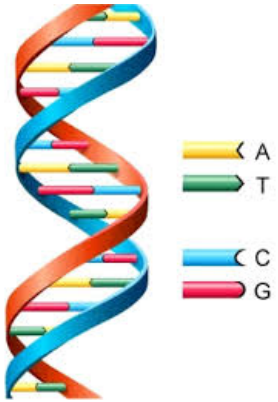


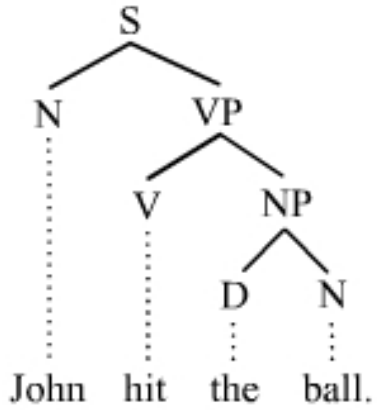
- Intersection of Theoretical Computer Science and Machine Learning...

$\rightarrow$ Weighted Automata: robust model to represent functions defined over structured objects (and in particular probability distributions).

Introduction

- How can one learn with structured objects such as strings and trees?





- Intersection of Theoretical Computer Science and Machine Learning...

→ Weighted Automata: robust model to represent functions defined over structured objects (and in particular probability distributions).

- String Weighted Automata (WA): generalize *Hidden Markov Models*
- Weighted Tree Automata (WTA): closely related to *PCFGs*

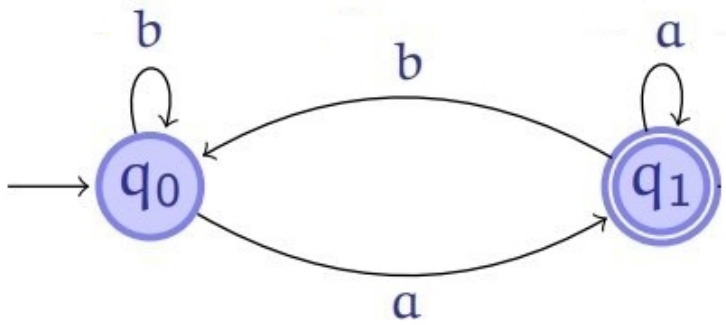
Weighted Automata (WA)

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)

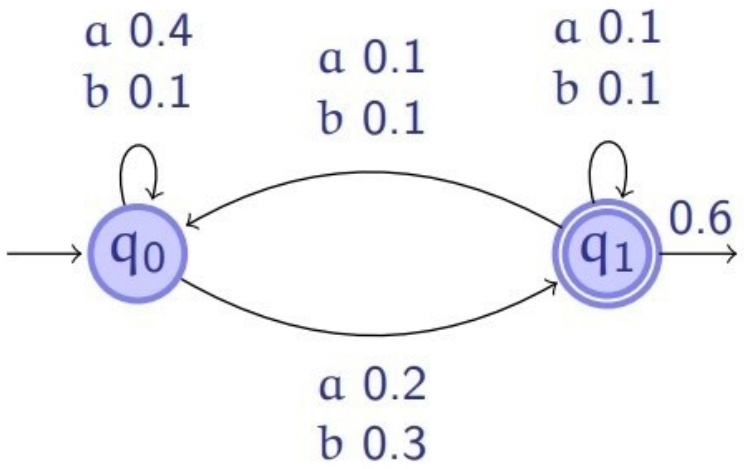
- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. *abba*), λ the empty string.
- Recall: a Deterministic Finite Automaton (DFA) recognizes a *language* (subset of Σ^*).



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

Weighted Automata: States and Weighted Transitions

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

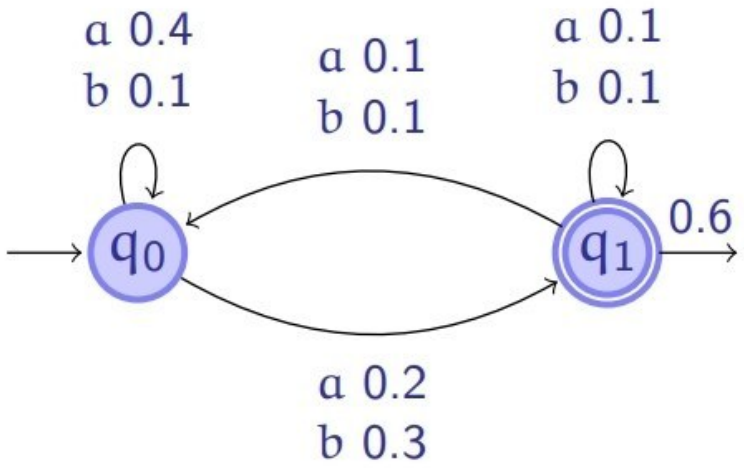


$$f(ab) = 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084$$

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 $\sigma \in \Sigma$

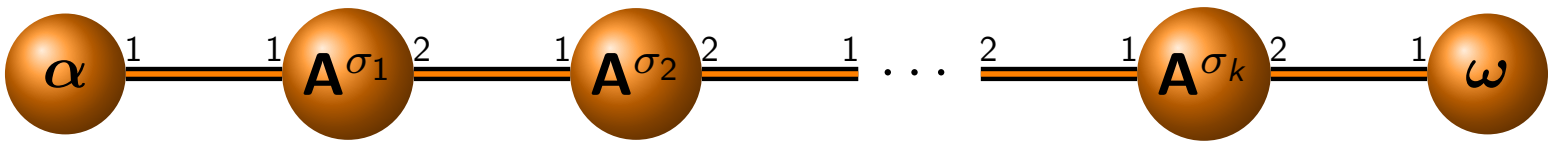
String Weighted Automata (WA)

- 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

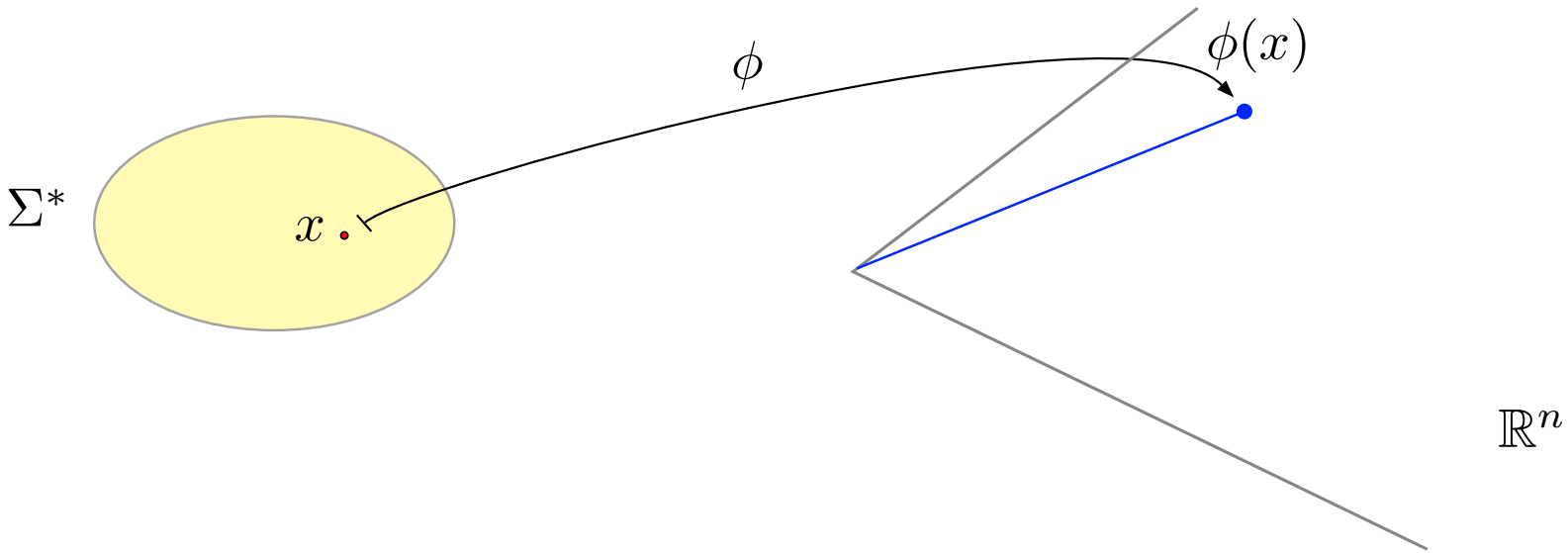
$$\begin{aligned} \boldsymbol{\alpha} &\in \mathbb{R}^n && \text{initial weights vector} \\ \boldsymbol{\omega} &\in \mathbb{R}^n && \text{final weights vector} \\ \mathbf{A}^\sigma &\in \mathbb{R}^{n\times n} && \text{transition weights matrix for each } \sigma \in \Sigma \end{aligned}$$

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

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

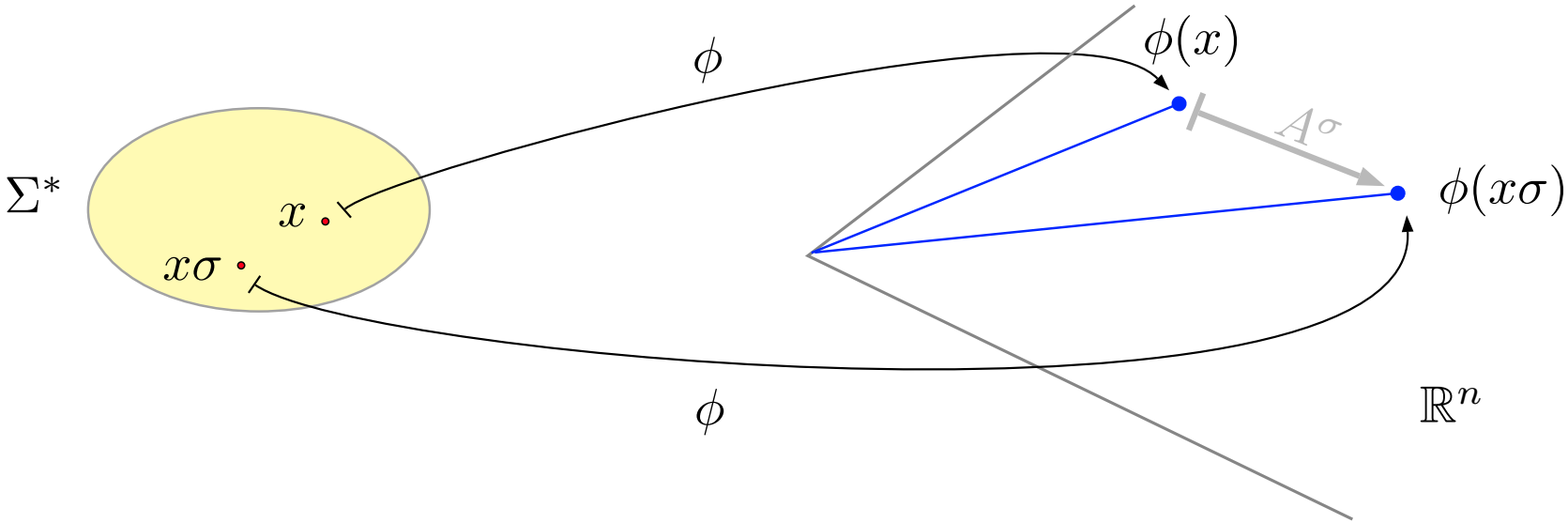


Weighted Automata and Representation Learning



- A WA induces a mapping $\phi : \Sigma^* \to \mathbb{R}^n$ ($\sim$ word embedding)

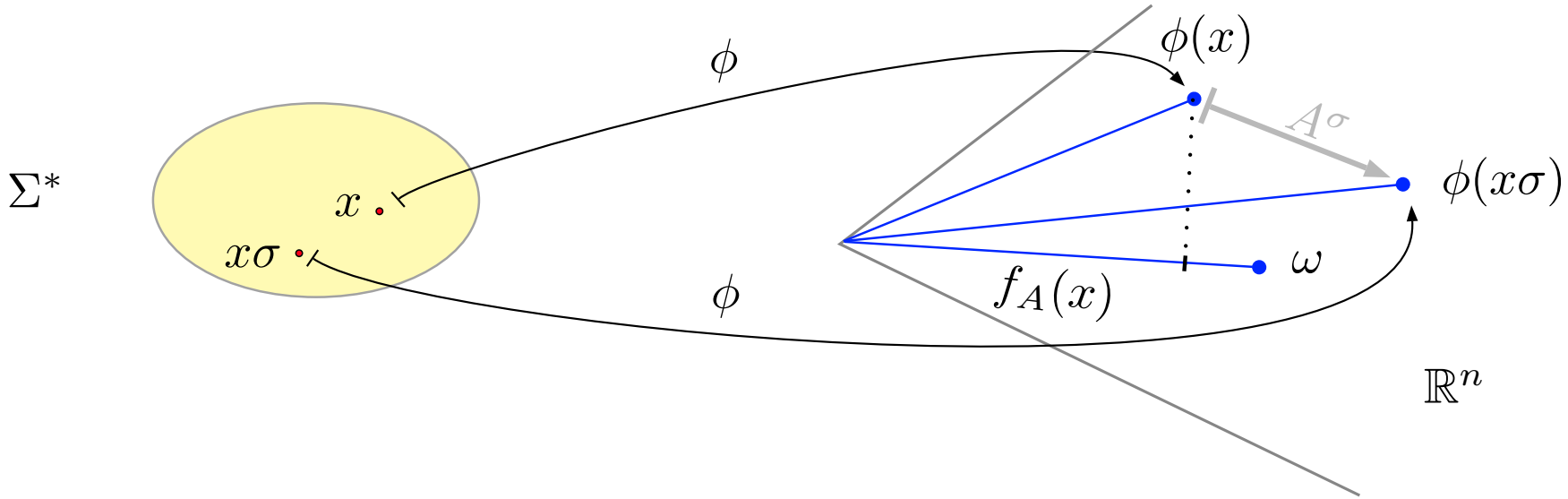
Weighted Automata and Representation Learning



- A WA induces a mapping $\phi : \Sigma^* \to \mathbb{R}^n$ ($\sim$ word embedding)
- The mapping ϕ is compositional:

$$\phi(\lambda) = \boldsymbol{\alpha}^\top, \;\; \phi(\sigma_1) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1}, \;\; \phi(\sigma_1\sigma_2) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1}\mathbf{A}^{\sigma_2} = \phi(\sigma_1)\mathbf{A}^{\sigma_2}, \;\; \ldots$$

Weighted Automata and Representation Learning



- A WA induces a mapping $\phi : \Sigma^* \to \mathbb{R}^n$ ($\sim$ word embedding)
- The mapping ϕ is compositional:

$$\phi(\lambda) = \boldsymbol{\alpha}^\top, \;\; \phi(\sigma_1) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1}, \;\; \phi(\sigma_1\sigma_2) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1}\mathbf{A}^{\sigma_2} = \phi(\sigma_1)\mathbf{A}^{\sigma_2}, \;\; \ldots$$

- The output $f_A(x) = \langle \phi(x), \boldsymbol{\omega} \rangle$ is linear in $\phi(x)$.

Spectral Learning of Weighted Automata

Hankel matrix

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

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

Hankel matrix and WA

Theorem (Fliess '74)
The rank of a *real* Hankel matrix H equals the minimal number of states of a WFA recognizing the weighted language of H

$$A(p_1 \cdots p_t s_1 \cdots s_{t'}) = \alpha^\top A_{p_1} \cdots A_{p_t} A_{s_1} \cdots A_{s_{t'}} \beta$$

$$\begin{matrix} & & s & & \\ p & \begin{bmatrix} \cdot & \cdot & A(ps) & \cdot & \cdot \end{bmatrix} \end{matrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \end{bmatrix}$$

H P S

slide credits: B. Balle

Hankel matrices

In addition to the Hankel matrix

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

For each $\sigma \in \Sigma$ we also define

$$\mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & aa & ab & \dots \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) & \dots & \dots & \dots \\ f(b\sigma a) & f(b\sigma b) & \dots & \dots & \dots \\ f(aa\sigma a) & f(aa\sigma b) & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

Hankel matrix: spectral learning

$$H_a(p,s) = A(pas)$$

$$A(p_1 \cdots p_t a s_1 \cdots s_{t'}) = \alpha^\top A_{p_1} \cdots A_{p_t} A_a A_{s_1} \cdots A_{s_{t'}} \beta$$

H_a P A_a S

$$H = P\ S \qquad H_a = P\ A_a\ S \qquad A_a = P^+\ H_a\ S^+$$

slide credits: B. Balle

Spectral Learning of Weighted Automata (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{H}$ and $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ for each $\sigma \in \Sigma$

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(aa) & f(ab) \\ f(ba) & f(bb) \\ f(aaa) & f(aab) \end{bmatrix} \end{array} \quad \mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) \\ f(b\sigma a) & f(b\sigma b) \\ f(aa\sigma a) & f(aa\sigma b) \end{bmatrix} \end{array}$$

3. Perform rank n decomposition $\mathbf{H} = \mathbf{P}\mathbf{S}$
4. WA with initial/final weights $\boldsymbol{\alpha} = \mathbf{P}_{\lambda,:}$, $\boldsymbol{\omega} = \mathbf{S}_{:,\lambda}$ and transition matrices $\mathbf{A}^\sigma = \mathbf{P}^\dagger \mathbf{H}_\sigma \mathbf{S}^\dagger$ is a minimal WFA for f.

$\rightarrow$ 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

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

$$\text{rank}(\mathbf{H}_{\mathcal{P} \times \mathcal{S}}) = \text{rank}(\mathbf{H}_{\Sigma^* \times \Sigma^*}) < \infty$$

then the spectral learning algorithm returns a minimal WA computing f.

Spectral Learning: when does it work?

Theorem

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

$$\text{rank}(\mathbf{H}_{\mathcal{P}\times\mathcal{S}}) = \text{rank}(\mathbf{H}_{\Sigma^*\times\Sigma^*}) < \infty$$

then the spectral learning algorithm returns a minimal WA computing f.

Suppose f is computed by a WA. By a continuity argument, if we are given noisy estimates $\hat{\mathbf{H}} = \mathbf{H} + \boldsymbol{\xi}$, $\hat{\mathbf{H}}_\sigma = \hat{\mathbf{H}}_\sigma + \boldsymbol{\xi}_\sigma$ we have

$$\lim_{\|\boldsymbol{\xi}\|\to 0,\ \|\boldsymbol{\xi}_\sigma\|\to 0} \hat{f} = f$$

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

Spectral Learning: when does it work?

Theorem

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

$$\text{rank}(\mathbf{H}_{\mathcal{P}\times\mathcal{S}}) = \text{rank}(\mathbf{H}_{\Sigma^*\times\Sigma^*}) < \infty$$

then the spectral learning algorithm returns a minimal WA computing f.

Suppose f is computed by a WA. By a continuity argument, if we are given noisy estimates $\hat{\mathbf{H}} = \mathbf{H} + \boldsymbol{\xi}$, $\hat{\mathbf{H}}_\sigma = \hat{\mathbf{H}}_\sigma + \boldsymbol{\xi}_\sigma$ we have

$$\lim_{\|\boldsymbol{\xi}\|\to 0,\ \|\boldsymbol{\xi}_\sigma\|\to 0} \hat{f} = f$$

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

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

Spectral Learning: when does it work?

Theorem

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

$$\text{rank}(\mathbf{H}_{\mathcal{P}\times\mathcal{S}}) = \text{rank}(\mathbf{H}_{\Sigma^*\times\Sigma^*}) < \infty$$

then the spectral learning algorithm returns a minimal WA computing f.

Suppose f is computed by a WA. By a continuity argument, if we are given noisy estimates $\hat{\mathbf{H}} = \mathbf{H} + \boldsymbol{\xi}$, $\hat{\mathbf{H}}_\sigma = \hat{\mathbf{H}}_\sigma + \boldsymbol{\xi}_\sigma$ we have

$$\lim_{\|\boldsymbol{\xi}\|\to 0,\ \|\boldsymbol{\xi}_\sigma\|\to 0} \hat{f} = f$$

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

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

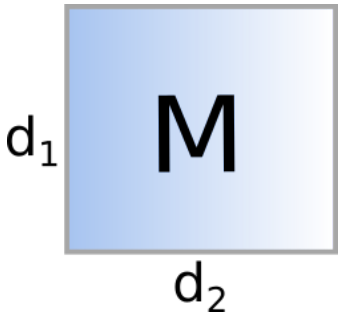
(if not, then first do matrix completion, then spectral method [Balle and Mohri, 2013])

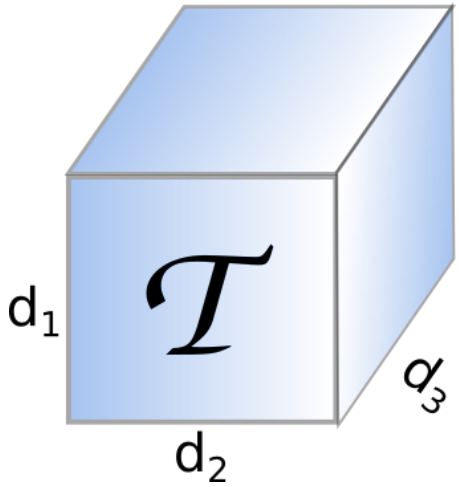
Open problems

- Selecting prefixes and suffixes $\mathcal{P}$ and $\mathcal{S}$ from the data
- Constraining the WA model (e.g. probabilistic automaton)
- Extension to semi-rings
- Extension to non-linear models
- Extension to graphs (trees done to some extent)
- Spectral method as an initialization to local optimization for non-linear models
- ...

A Small Detour through Tensors and Tensor Networks

Tensors





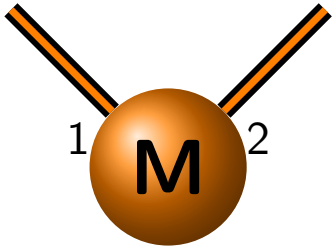
$\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$

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

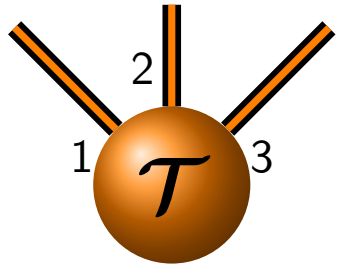
$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

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

Tensor Networks

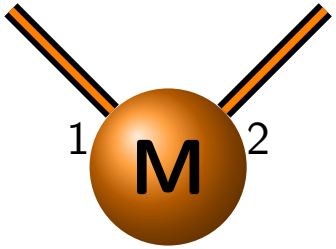


Matrix: $\mathbf{M}_{i_1 i_2}$

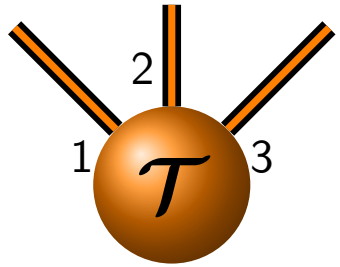


3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Tensor Networks

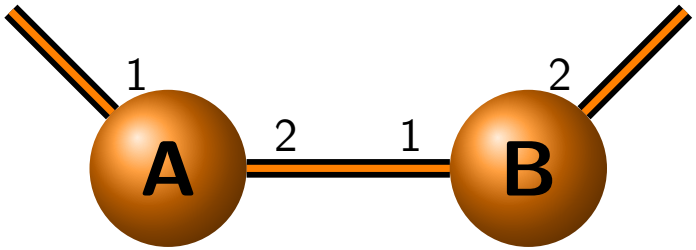


Matrix: $\mathbf{M}_{i_1 i_2}$



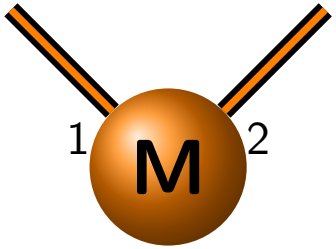
3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Matrix product:

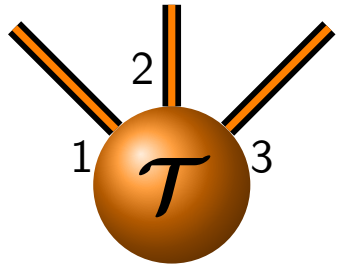


$$(\mathbf{AB})_{i_1,i_2} = \sum_k \mathbf{A}_{i_1 k} \mathbf{B}_{k i_2}$$

Tensor Networks

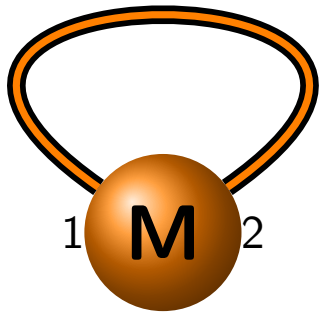


Matrix: $\mathbf{M}_{i_1 i_2}$



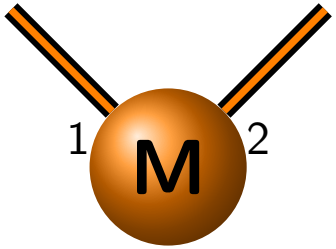
3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Trace:

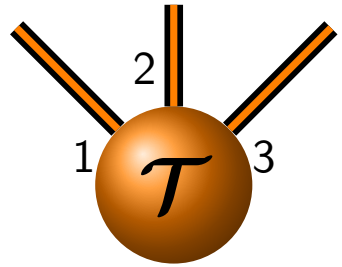


$$\mathrm{Tr}(\mathbf{M}) = \sum_i \mathbf{M}_{ii}$$

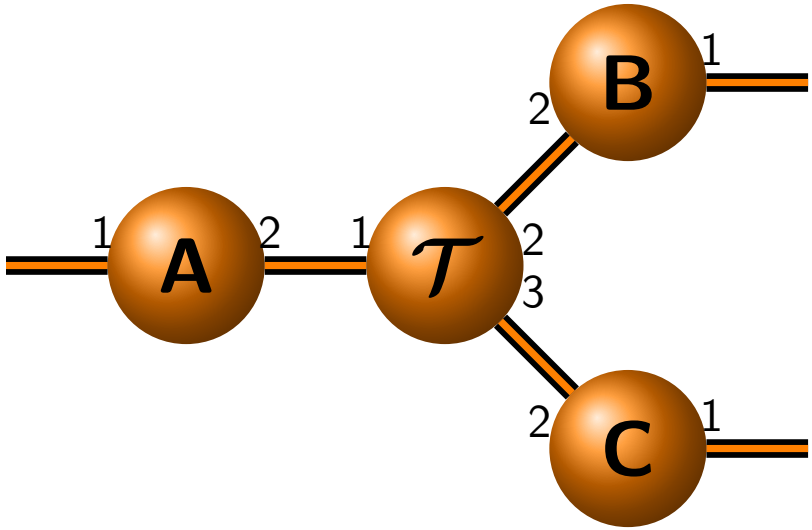
Tensor Networks



Matrix: $\mathbf{M}_{i_1 i_2}$



3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

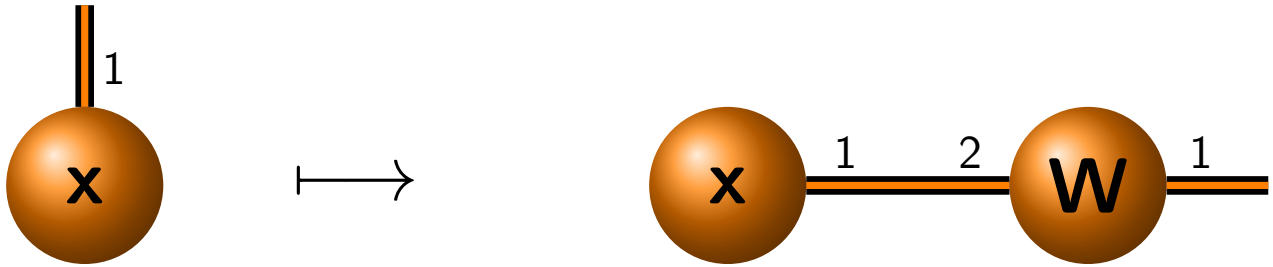


Tensor times matrices:

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

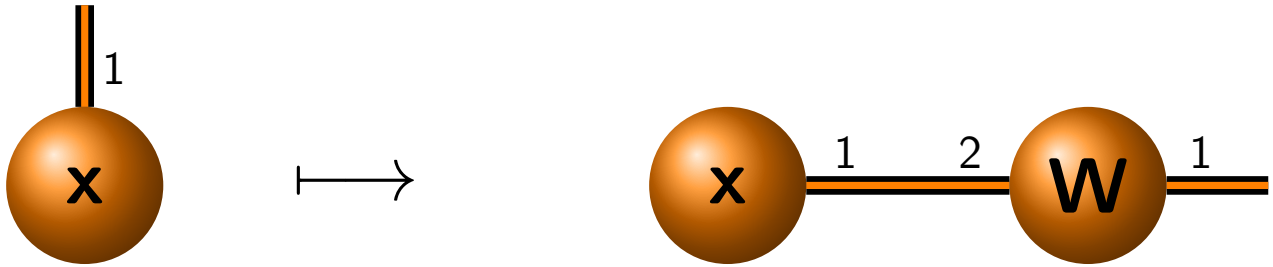
Multilinear Maps

- Liner map $f : \mathbb{R}^d \to \mathbb{R}^p$ maps $\mathbf{x}$ to $\mathbf{W}\mathbf{x} = \mathbf{W} \times_2 \mathbf{x}$ for some $\mathbf{W} \in \mathbb{R}^{p \times d}$:

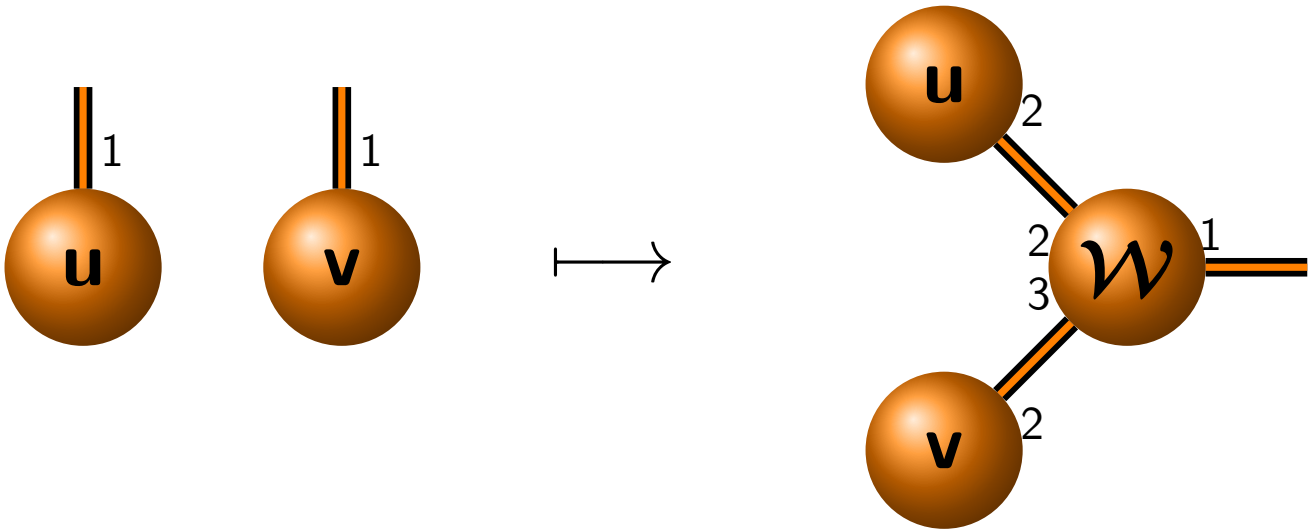


Multilinear Maps

- Liner map $f : \mathbb{R}^d \to \mathbb{R}^p$ maps $\mathbf{x}$ to $\mathbf{W}\mathbf{x} = \mathbf{W} \times_2 \mathbf{x}$ for some $\mathbf{W} \in \mathbb{R}^{p \times d}$:



- Multilinear map $g : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^p$ maps $(\mathbf{u}, \mathbf{v})$ to $\mathcal{W} \times_2 \mathbf{u} \times_3 \mathbf{v}$ for some $\mathcal{W} \in \mathbb{R}^{p \times d_1 \times d_2}$:



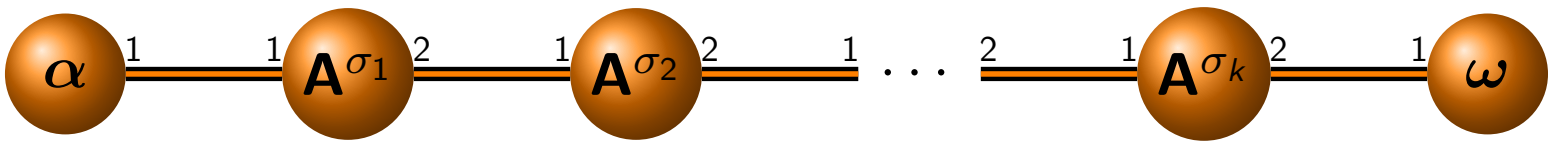
String Weighted Automata (WA)

- 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

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

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

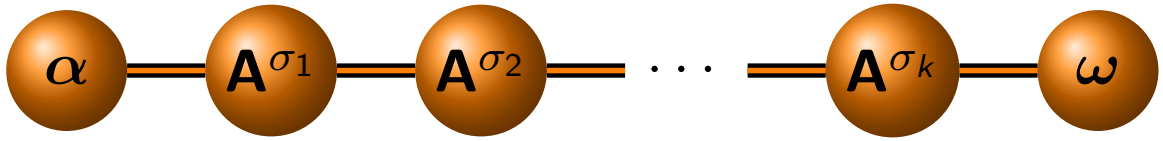
$$f_A(\sigma_1\sigma_2\cdots\sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1}\mathbf{A}^{\sigma_2}\cdots\mathbf{A}^{\sigma_k}\boldsymbol{\omega}$$



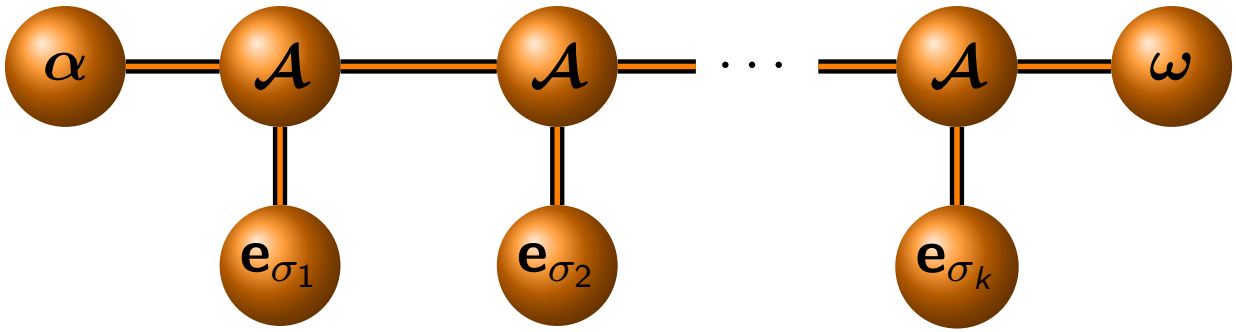
WA and Matrix Product States / Tensor Train

We can rewrite

$$f(\sigma_1\sigma_2\cdots\sigma_k) =$$



as

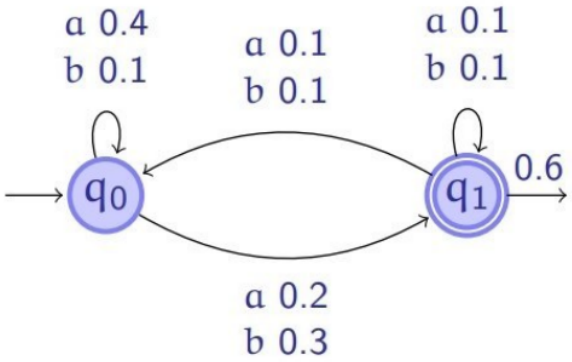


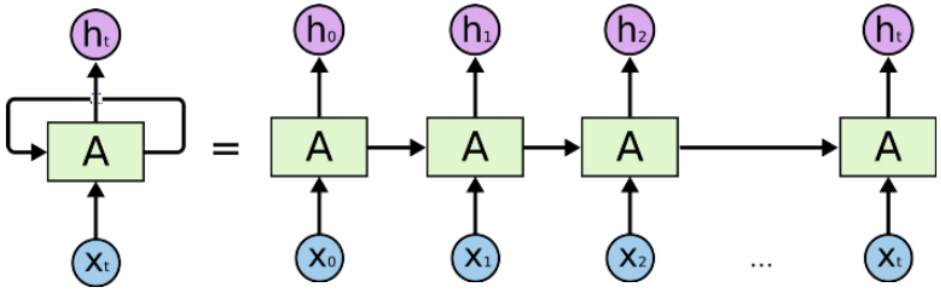
where $\mathcal{A} \in \mathbb{R}^{n\times\Sigma\times n}$ defined by $\mathcal{A}_{:,\sigma,:} = \mathbf{A}^\sigma$ and $\mathbf{e}_\sigma \in \mathbb{R}^\Sigma$ is the *one-hot encoding* of σ.

Connections between HMMs and MPS already noticed by Critch et al. around 2013 (and likely even before)

Weighted Automata Vs. RNNs

Weighted Automata Vs. Recurrent Neural Networks





- 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?

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]:

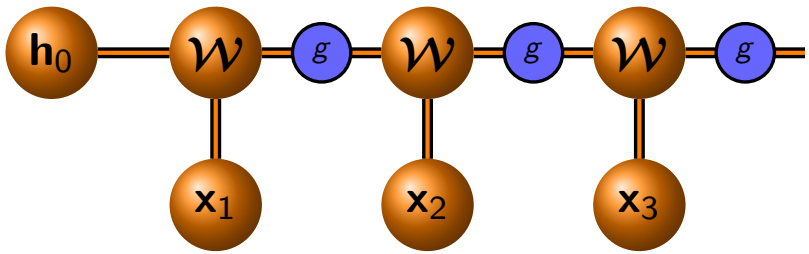
$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$

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

Weighted Automata and Recurrent Neural Networks

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

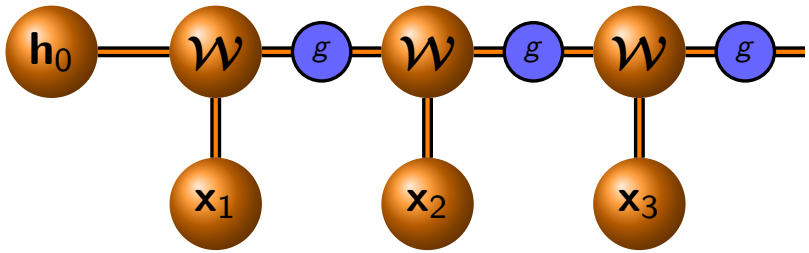
$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$



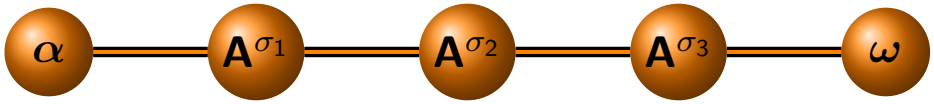
Weighted Automata and Recurrent Neural Networks

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

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



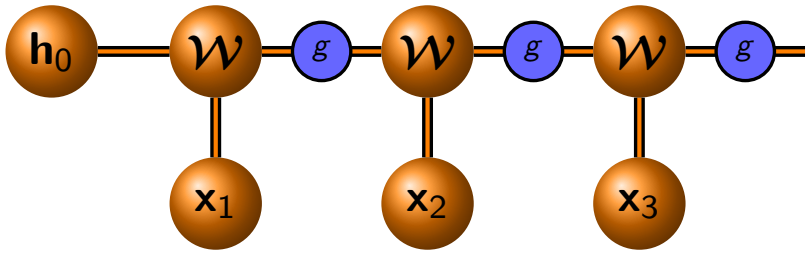
- The computation of a weighted automaton is very similar!



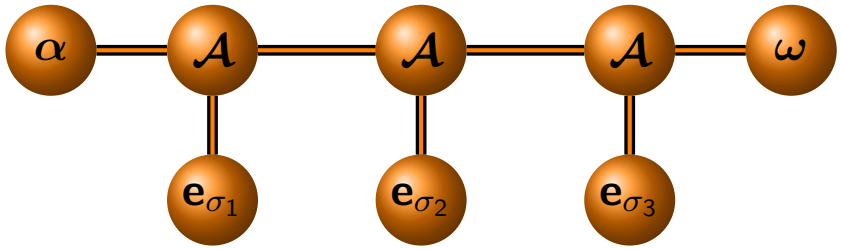
Weighted Automata and Recurrent Neural Networks

- 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

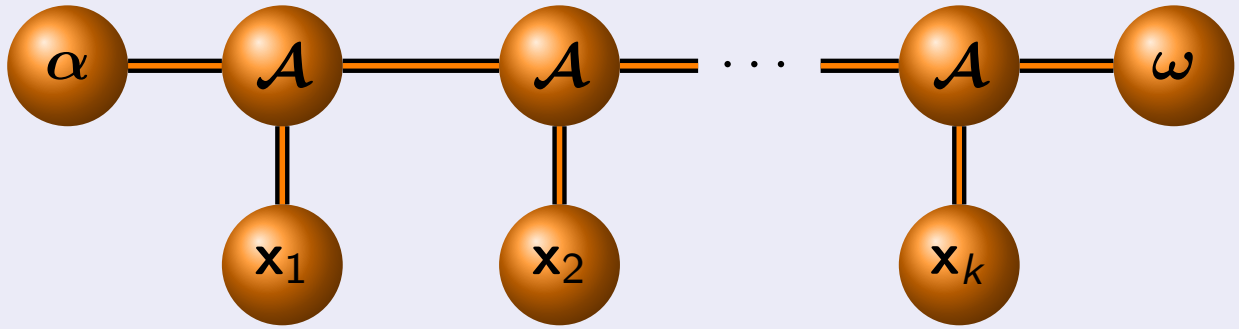
$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector

$\boldsymbol{\omega} \in \mathbb{R}^n$ 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?
  - $\star$ Over sequences of discrete symbols?

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

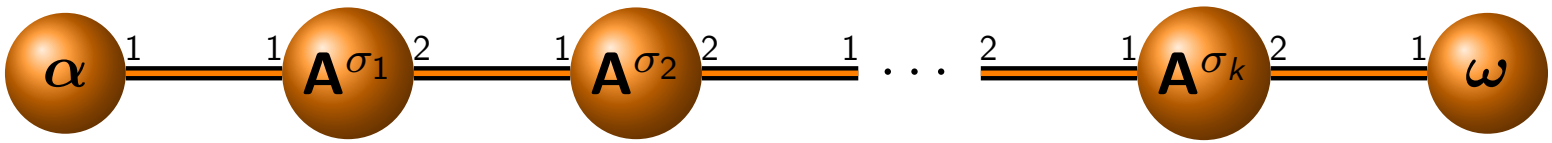
  $\hookrightarrow$ Yes: technical contribution of [AISTATS'19]

Beyond Strings and Trees: Graph Weighted Models

A Look Back on String Weighted Automata

A Weighted Automaton $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\omega})$ computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by

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

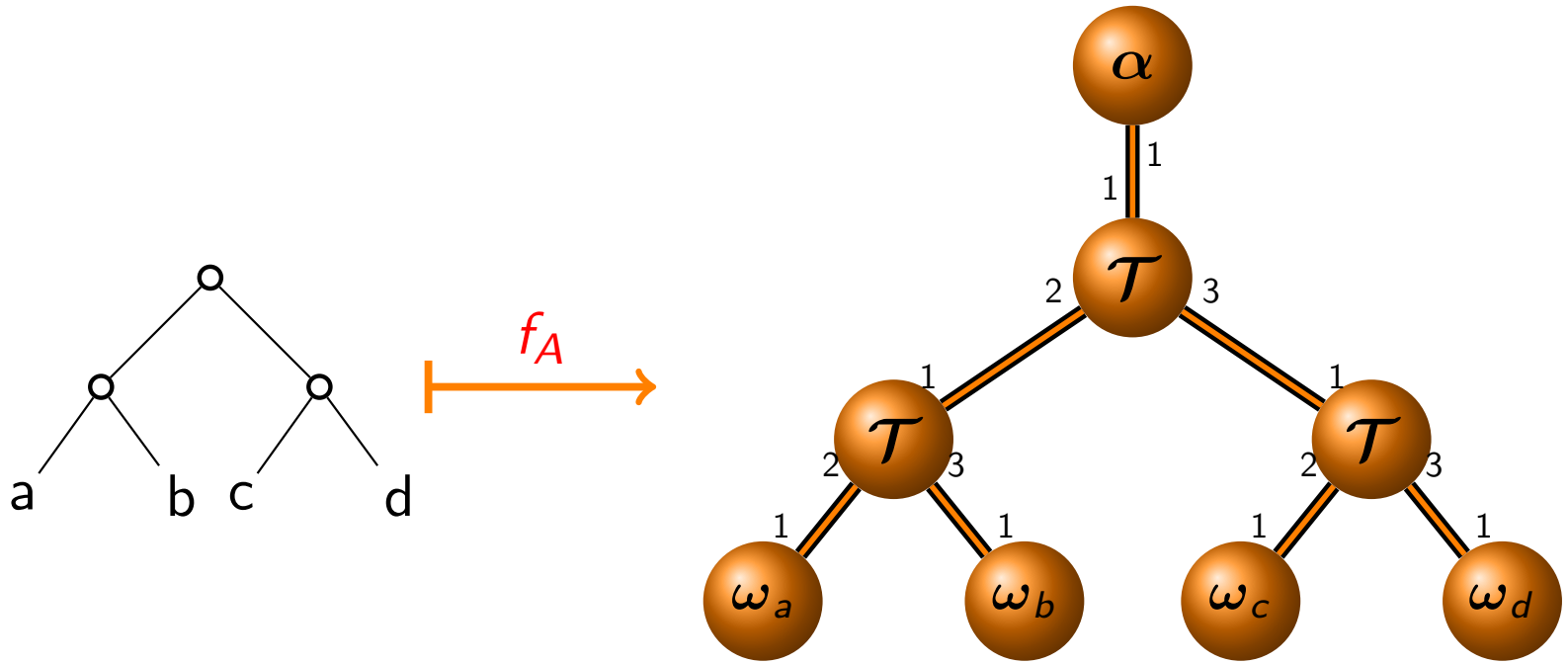


Beyond Strings: Weighted Tree Automata

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

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

- A WTA computes a function $f_A : \mathfrak{T}_\Sigma \to \mathbb{R}$.



Graph Weighted Models [R. Bailly*, **GR***, F. Denis, LATA'15/JCSS'18]

- $\mathcal{F} = \{a(\cdot), h(\cdot,\cdot), g(\cdot,\cdot,\cdot)\}$

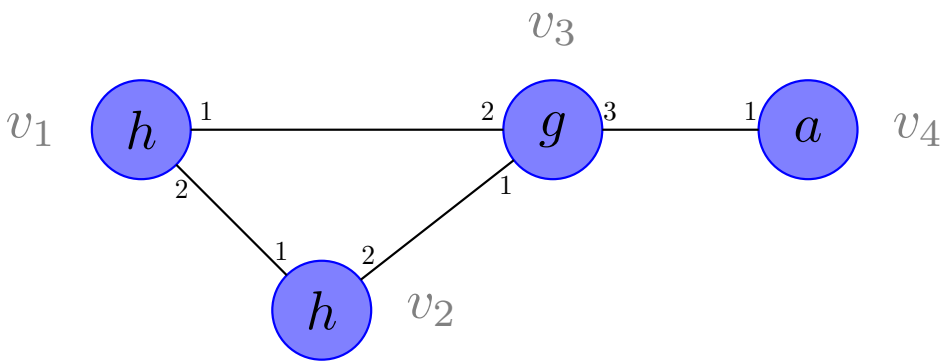


Figure: A graph on the ranked alphabet $\mathcal{F} = \{a(\cdot), h(\cdot,\cdot), g(\cdot,\cdot,\cdot)\}$.

Graph Weighted Models [R. Bailly*, **GR***, F. Denis, LATA'15/JCSS'18]

- $\mathcal{F} = \{a(\cdot), h(\cdot,\cdot), g(\cdot,\cdot,\cdot)\}$

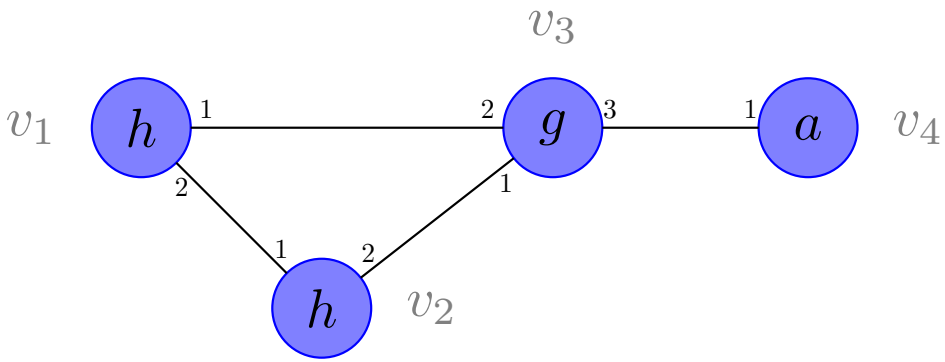
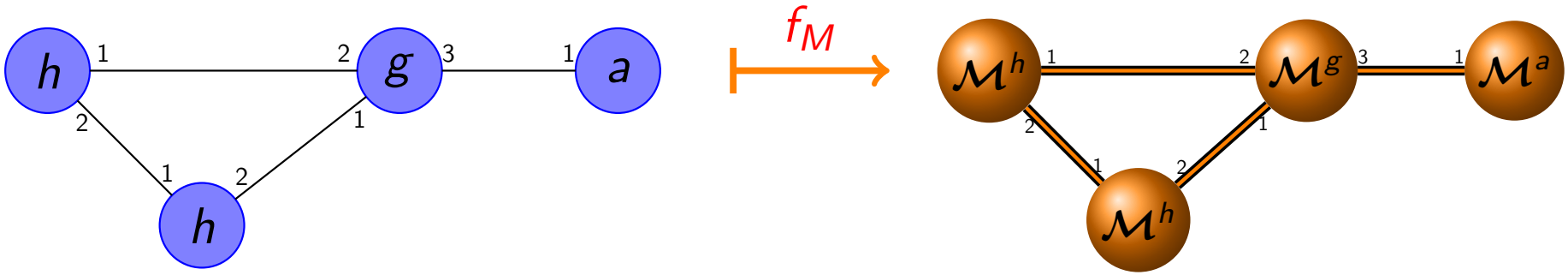


Figure: A graph on the ranked alphabet $\mathcal{F} = \{a(\cdot), h(\cdot,\cdot), g(\cdot,\cdot,\cdot)\}$.

- GWM: vector $\boldsymbol{\mathcal{M}}^a \in \mathbb{R}^n$, matrix $\boldsymbol{\mathcal{M}}^h \in \mathbb{R}^{n\times n}$, tensor $\boldsymbol{\mathcal{M}}^g \in \mathbb{R}^{n\times n\times n}$

Graph Weighted Models [R. Bailly*, **GR***, F. Denis, LATA'15/JCSS'18]

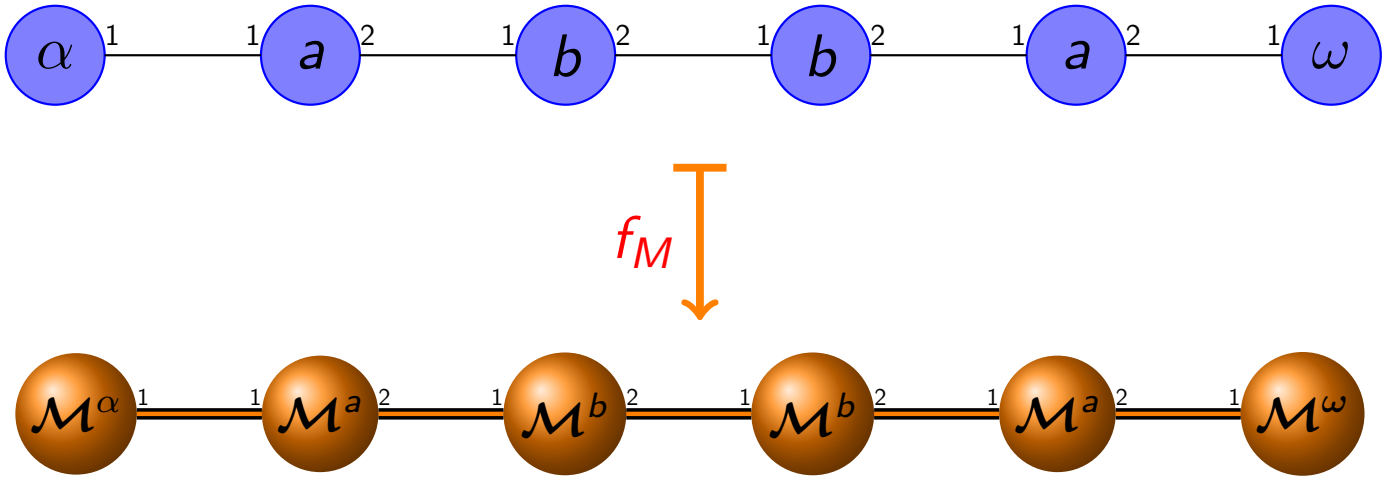
- $\mathcal{F} = \{a(\cdot), h(\cdot,\cdot), g(\cdot,\cdot,\cdot)\}$
- GWM: vector $\boldsymbol{\mathcal{M}}^a \in \mathbb{R}^n$, matrix $\boldsymbol{\mathcal{M}}^h \in \mathbb{R}^{n\times n}$, tensor $\boldsymbol{\mathcal{M}}^g \in \mathbb{R}^{n\times n\times n}$



$$f_M(G) = \sum_{i_1,i_2,i_3,i_4 \in [n]} \boldsymbol{\mathcal{M}}^h_{i_1,i_2} \boldsymbol{\mathcal{M}}^h_{i_2,i_3} \boldsymbol{\mathcal{M}}^g_{i_3,i_1,i_4} \boldsymbol{\mathcal{M}}^a_{i_4}$$

Graph Weighted Models [R. Bailly*, **GR***, F. Denis, LATA'15/JCSS'18]

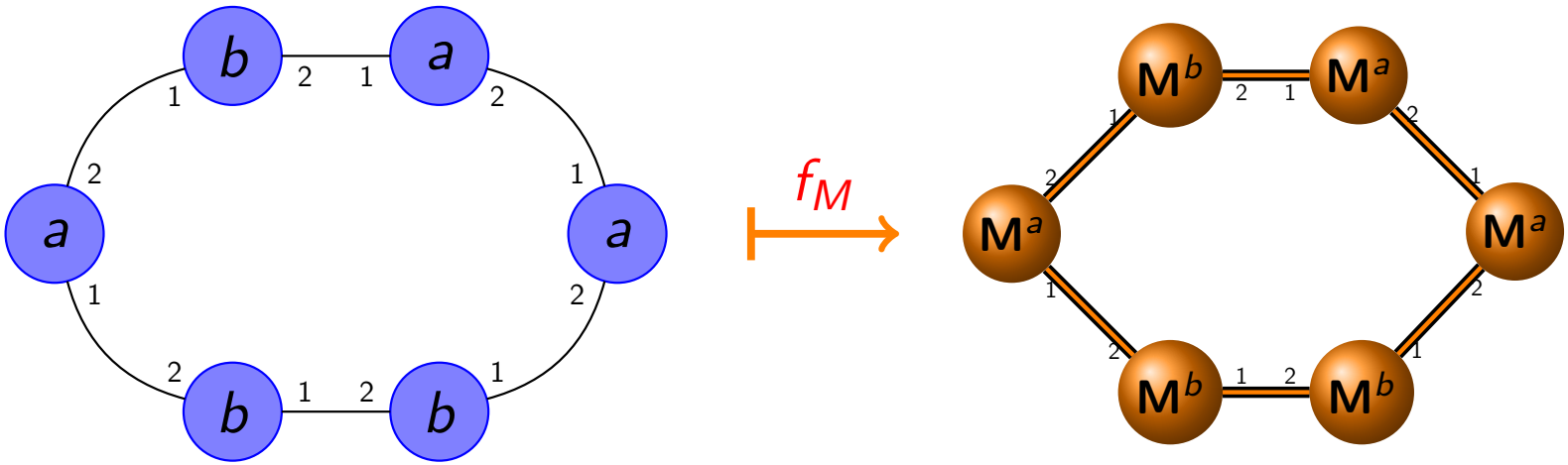
- $\mathcal{F} = \{\alpha(\cdot), a(\cdot,\cdot), b(\cdot,\cdot), \omega(\cdot)\}$
- GWM: $\boldsymbol{\mathcal{M}}^{\alpha}, \boldsymbol{\mathcal{M}}^{\omega} \in \mathbb{R}^{n}$, $\boldsymbol{\mathcal{M}}^{a}, \boldsymbol{\mathcal{M}}^{b} \in \mathbb{R}^{n \times n}$



$$f_M(G) = \sum_{i_1,i_2,i_3,i_4,i_5 \in [n]} \boldsymbol{\mathcal{M}}^{\alpha}_{i_1} \boldsymbol{\mathcal{M}}^{a}_{i_1,i_2} \boldsymbol{\mathcal{M}}^{b}_{i_2,i_3} \boldsymbol{\mathcal{M}}^{b}_{i_3,i_4} \boldsymbol{\mathcal{M}}^{a}_{i_4,i_5} \boldsymbol{\mathcal{M}}^{\omega}_{i_5}$$
$$= \boldsymbol{\alpha}^{\top} \mathbf{M}^{a} \mathbf{M}^{b} \mathbf{M}^{b} \mathbf{M}^{a} \boldsymbol{\omega}$$

Graph Weighted Models [R. Bailly*, **GR***, F. Denis, LATA'15/JCSS'18]

- $\mathcal{F} = \{a(\cdot,\cdot), b(\cdot,\cdot)\}$
- GWM: $\mathbf{M}^a, \mathbf{M}^b \in \mathbb{R}^{n\times n}$



$$f_M(G) = \sum_{i_1,i_2,i_3,i_4,i_5,i_6\in[n]} \mathbf{M}^a_{i_1,i_2}\mathbf{M}^b_{i_2,i_3}\mathbf{M}^a_{i_3,i_4}\mathbf{M}^a_{i_4,i_5}\mathbf{M}^b_{i_5,i_6}\mathbf{M}^b_{i_6,i_1}$$

$$= \mathrm{Tr}(\mathbf{M}^a\mathbf{M}^b\mathbf{M}^a\mathbf{M}^a\mathbf{M}^b\mathbf{M}^b)$$

Learning Graph Weighted Models

- Long term objective: extend the spectral learning algorithm to functions defined over graphs.
  - $\rightarrow$ learning general GWMs is very challenging.
- First step: study the problem of learning GWMs defined over simple families of graphs (circular strings, 2D grids).
- Minimization of GWMs over circular strings [**GR**, FoSSaCS'18]:
  - Minimizing WA $\leftrightarrow$ linear algebra
  - Minimizing GWMs $\leftrightarrow$ theory of finite dimensional algebras

Thank you! Questions?