Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning

Guillaume Rabusseau, Tianyu Li, Doina Precup

Université de Montréal - Mila - CIFAR CCAI chair

May 2, 2019

IACS - Stony Brook University

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, ...

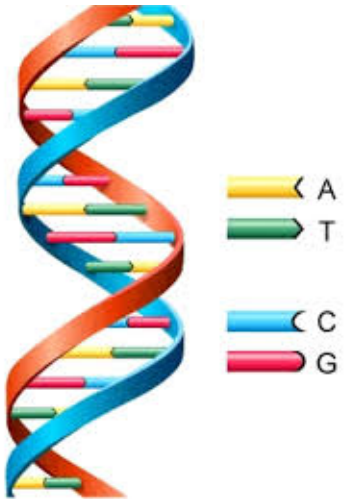
$\in \mathbb{R}^{32\times32\times3} \simeq \mathbb{R}^{3072}$

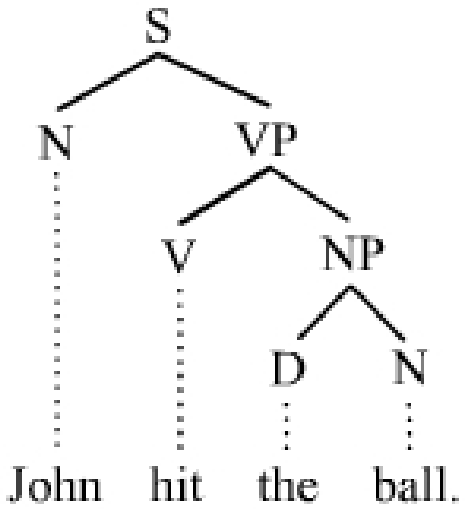
Learning with Structured Data

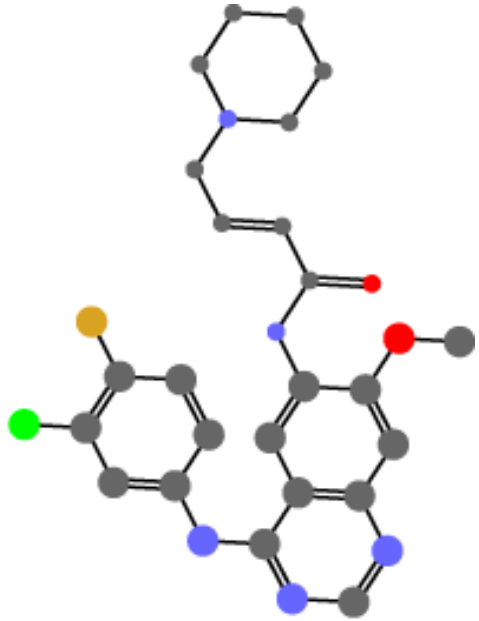
Supervised Learning:

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 $\{(x_1, y_1), \cdots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$.

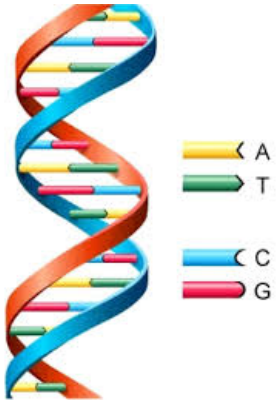
- Classical learning algorithms assume $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^p$.
- How to handle input/output structured data?
  - Tensor structured data: Images, videos, spatio-temporal data, ...
  - Discrete structured data: strings, trees, graphs, ...

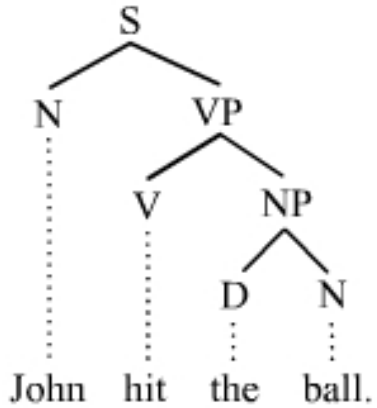
- In both cases, one can leverage linear and tensor algebra to design learning algorithms.

Outline

Introduction

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

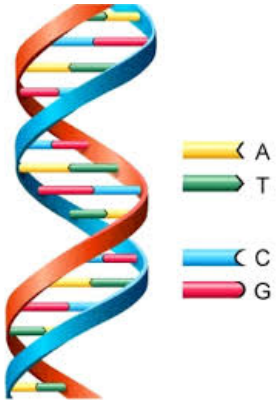


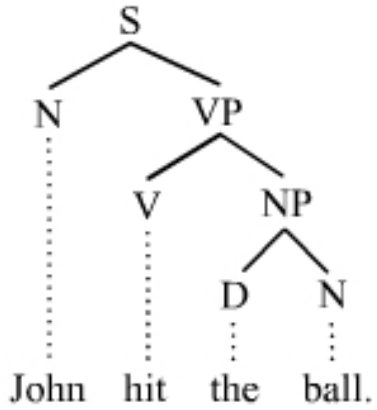


- Intersection of Theoretical Computer Science and Machine Learning...

Introduction

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



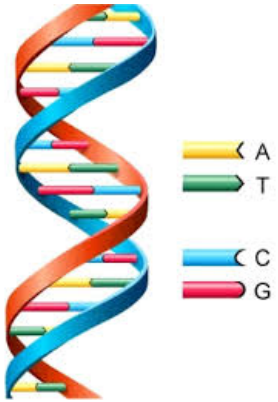


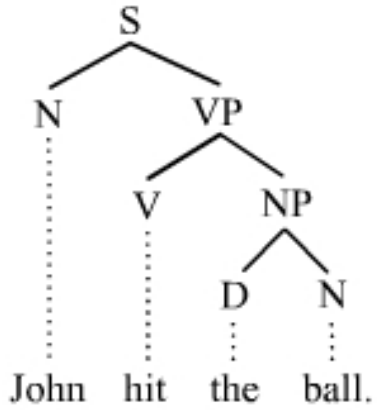
- Intersection of Theoretical Computer Science and Machine Learning...

→ 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?



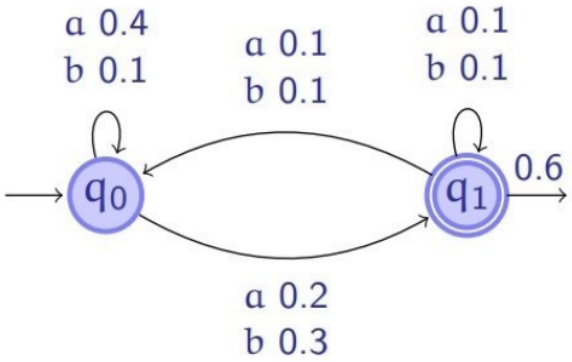


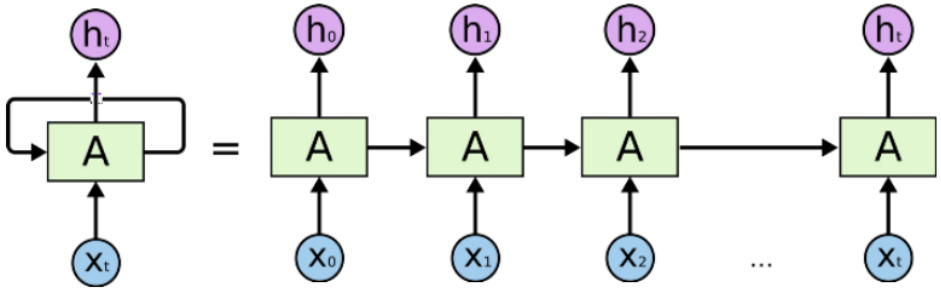
- Intersection of Theoretical Computer Science and Machine Learning...

→ 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 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]

Overview of the Results

In this work, we answer the following questions:

To which extent Weighted Automata are linear RNNs?

Overview of the Results

In this work, we answer the following questions:

To which extent Weighted Automata are linear RNNs?

- We show the exact equivalence of WAs and 2nd order RNNs with linear activation functions (linear 2-RNNs).
- This leads to a natural extension of WAs for sequences of **continuous vectors**.
- We extend the spectral learning algorithm for WAs: First provable learning algorithm for linear 2-RNNs.

Weighted Automata (WA) and Recurrent Neural Networks (RNN)

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 $\sigma \in \Sigma$

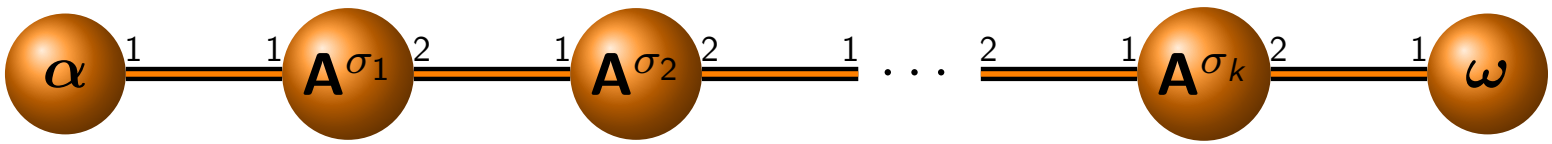
String Weighted Automata (WA)

- 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

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

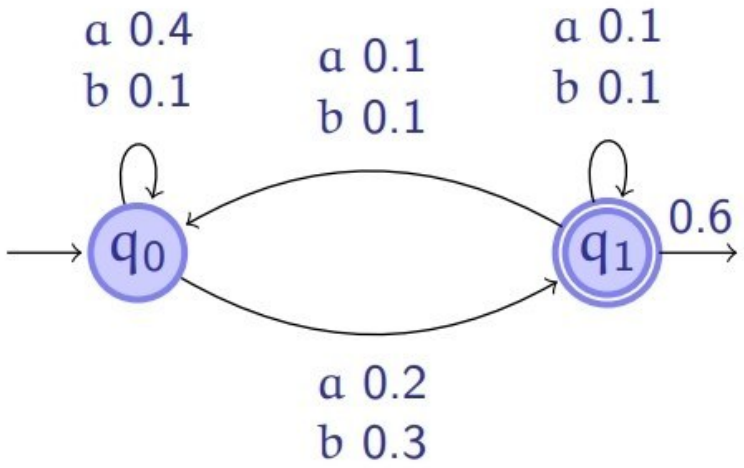
- A computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by

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



Weighted Automata: States and Transitions

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



Operator Representation

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

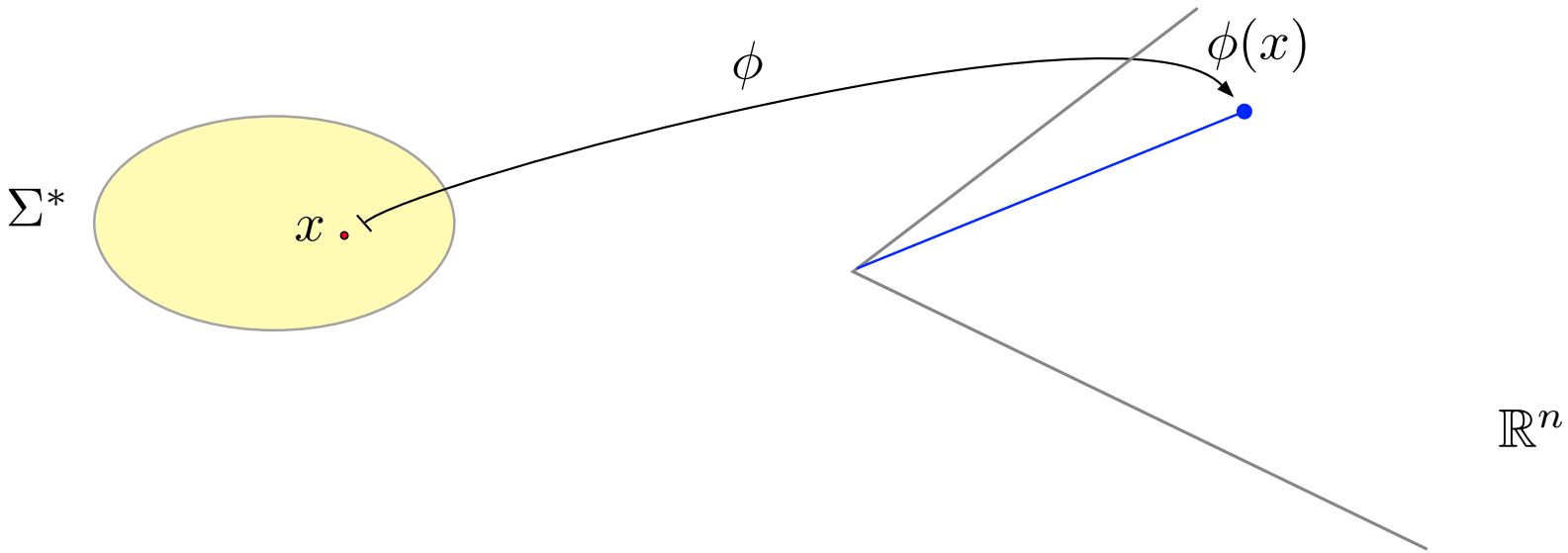
$$\boldsymbol{\omega} = \begin{bmatrix} 0.0 \\ 0.6 \end{bmatrix} \quad \mathbf{A}^{b} = \begin{bmatrix} 0.1 & 0.3 \\ 0.1 & 0.1 \end{bmatrix}$$

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

$$= \boldsymbol{\alpha}^{\top} \mathbf{A}^{a} \mathbf{A}^{b} \boldsymbol{\omega}$$

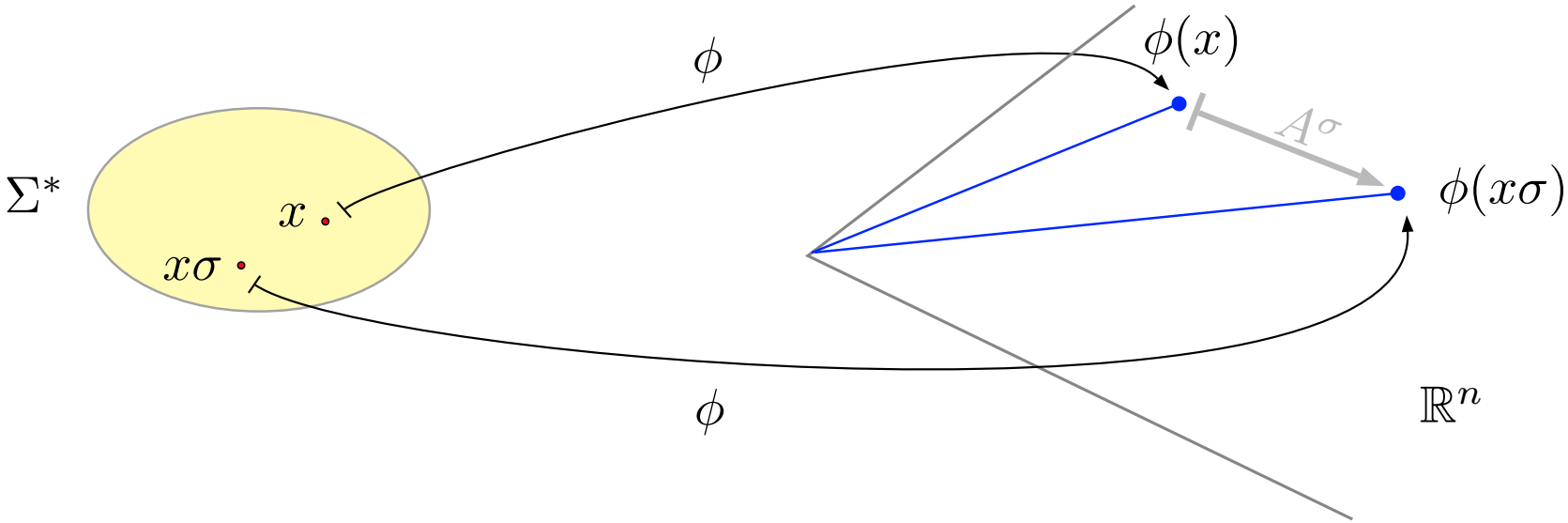
slide credits: B. Balle, X. Carreras, A. Quattoni - ENMLP'14 tutorial

Weighted Automata and Representation Learning



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

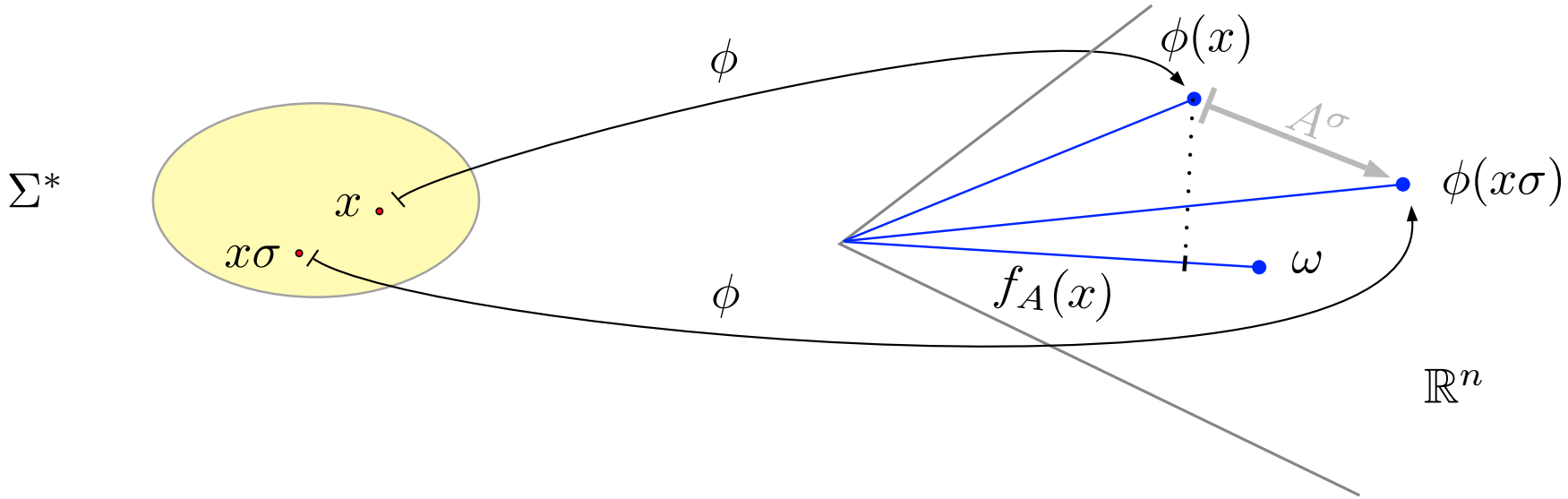
Weighted Automata and Representation Learning



- 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 $\phi(x)$.

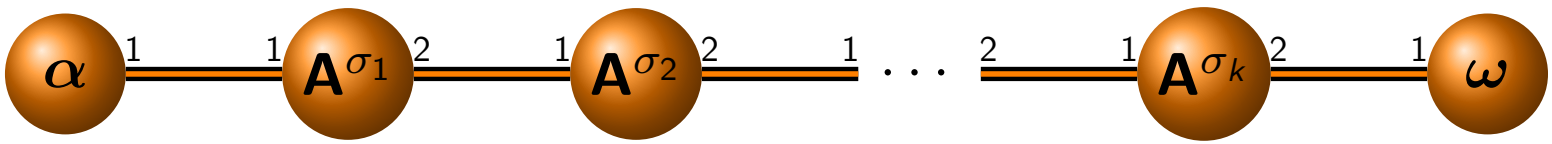
String Weighted Automata (WA)

- 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

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

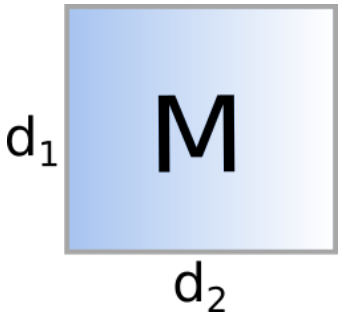
- A computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by

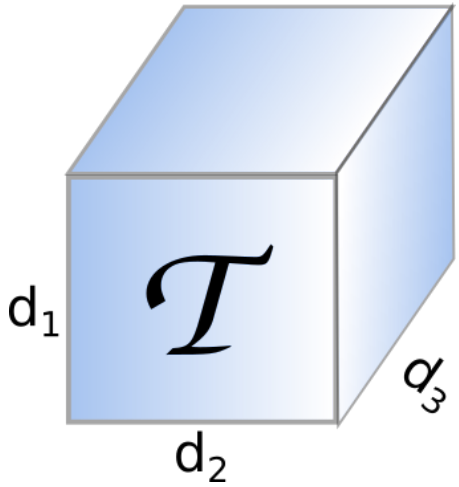
$$f_A(\sigma_1 \sigma_2 \cdots \sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1} \mathbf{A}^{\sigma_2} \cdots \mathbf{A}^{\sigma_k} \boldsymbol{\omega}$$



A Small Detour through Tensors and Tensor Networks

Tensors





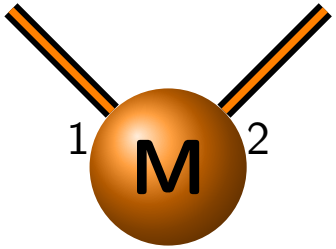
$\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$

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

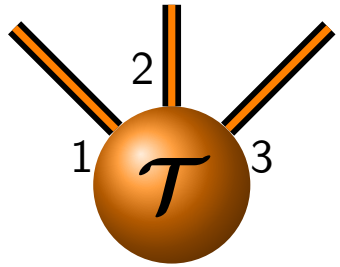
$\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$

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

Tensor Networks

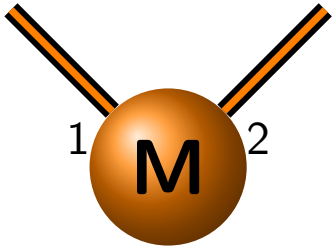


Matrix: $\mathbf{M}_{i_1 i_2}$

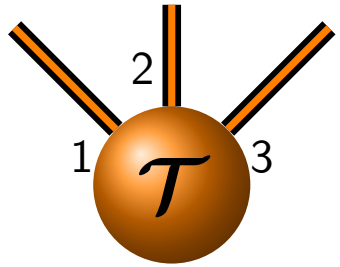


3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Tensor Networks

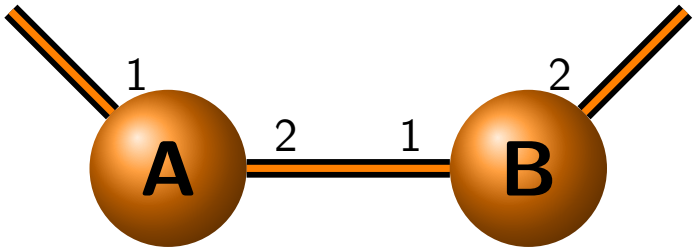


Matrix: $\mathbf{M}_{i_1 i_2}$



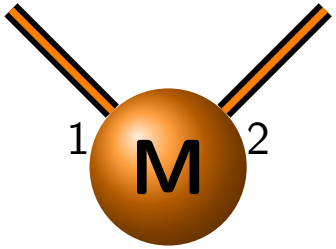
3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Matrix product:

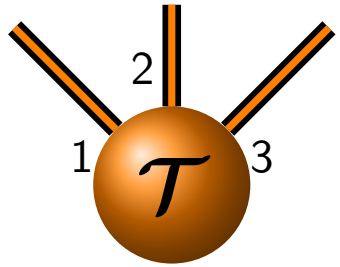


$$(\mathbf{AB})_{i_1,i_2} = \sum_k \mathbf{A}_{i_1 k} \mathbf{B}_{k i_2}$$

Tensor Networks

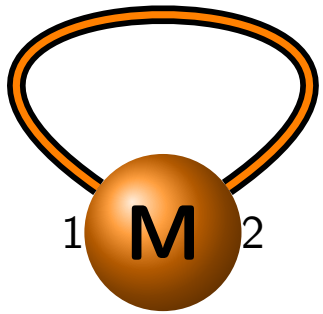


Matrix: $\mathbf{M}_{i_1 i_2}$



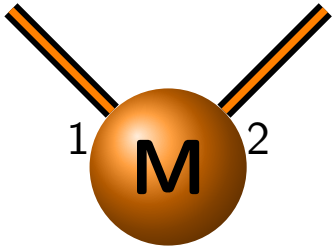
3rd order tensor: $\mathcal{T}_{i_1 i_2 i_3}$

Trace:

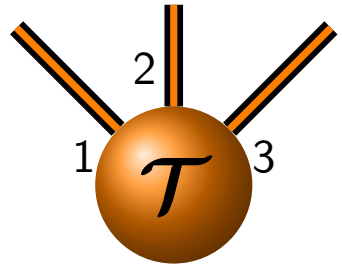


$$\mathrm{Tr}(\mathbf{M}) = \sum_i \mathbf{M}_{ii}$$

Tensor Networks

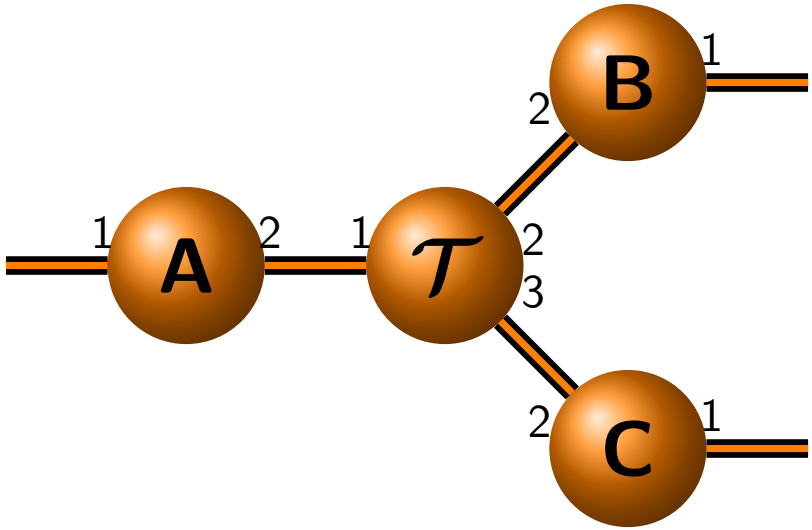


Matrix: $\mathbf{M}_{i_1 i_2}$



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

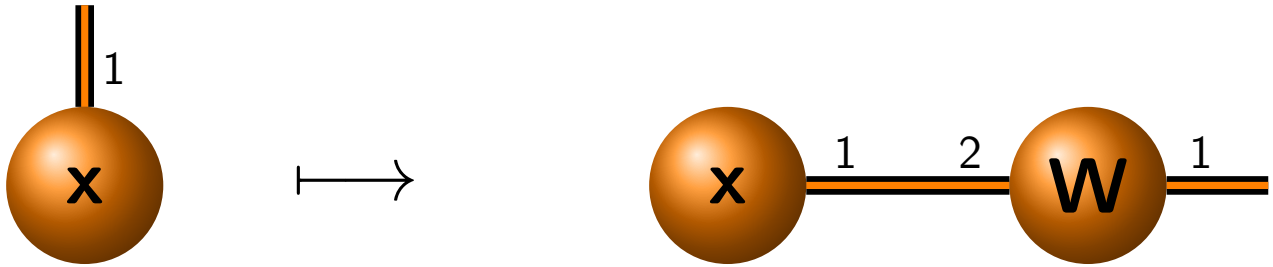
Tensor times matrices:



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

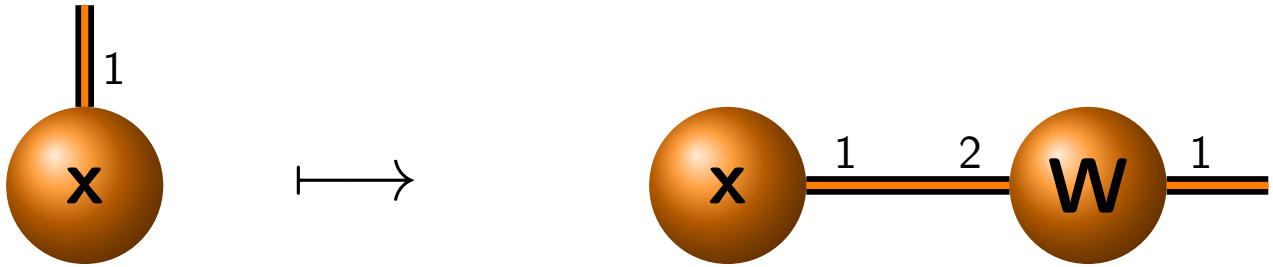
Multilinear Maps

- 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 $\mathbf{W} \in \mathbb{R}^{p \times d}$:

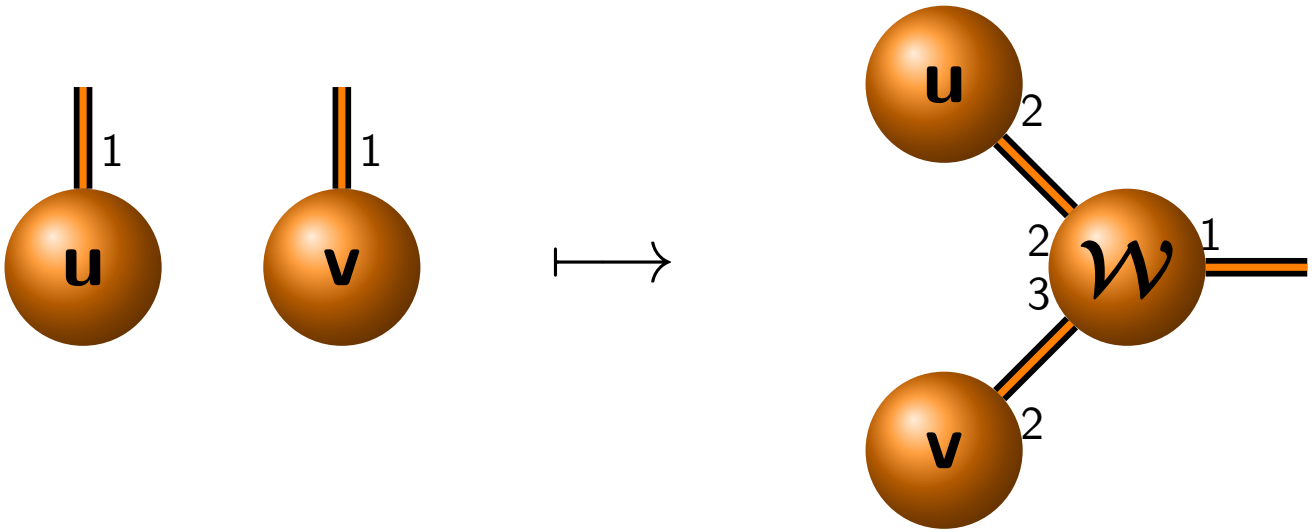


Multilinear Maps

- 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 $\mathcal{W} \in \mathbb{R}^{p \times d_1 \times d_2}$:



Example: Multilinear Maps in Higher-Order RNNs

- Recurrent Neural Network (RNN):

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

- Simple RNN:

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

Example: Multilinear Maps in Higher-Order RNNs

- Recurrent Neural Network (RNN):

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

- Simple 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]:

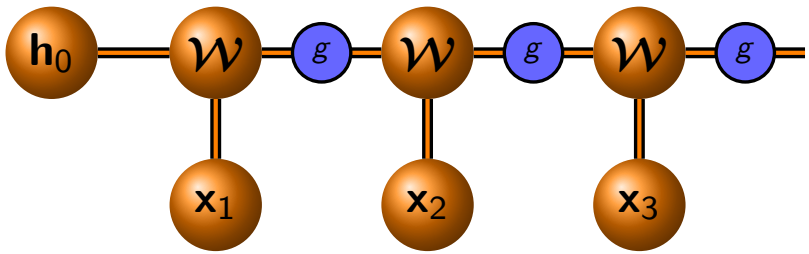
$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$

$\rightarrow$ 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

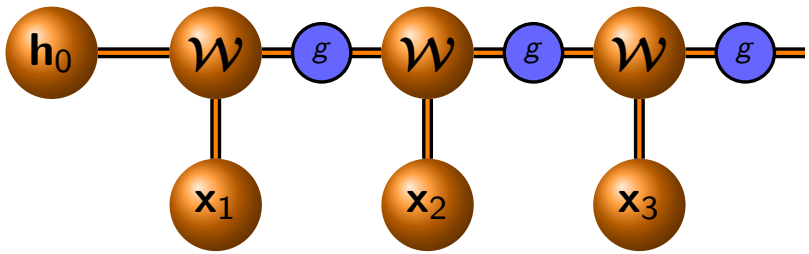
$$\mathbf{h}_t = g(\mathcal{W} \times_2 \mathbf{x}_t \times_3 \mathbf{h}_{t-1})$$



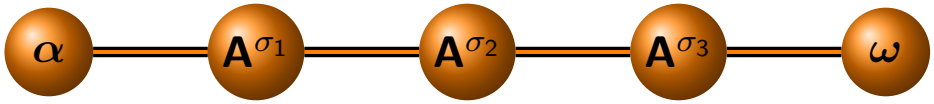
Weighted Automata and Recurrent Neural Networks

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

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



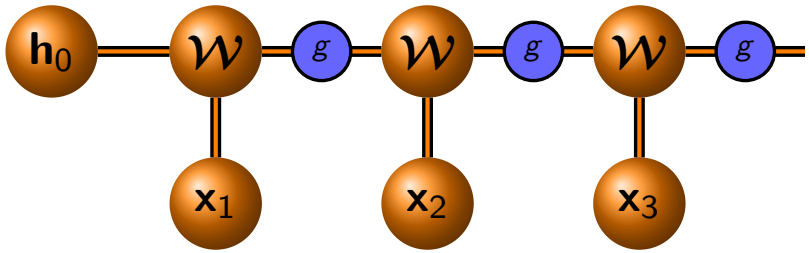
- The computation of a weighted automaton is very similar!



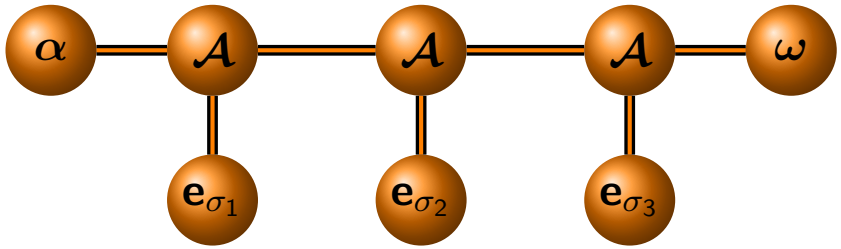
Weighted Automata and Recurrent Neural Networks

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

Our first result: WAs $\equiv$ linear 2-RNNs

Theorem

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

Our first result: 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

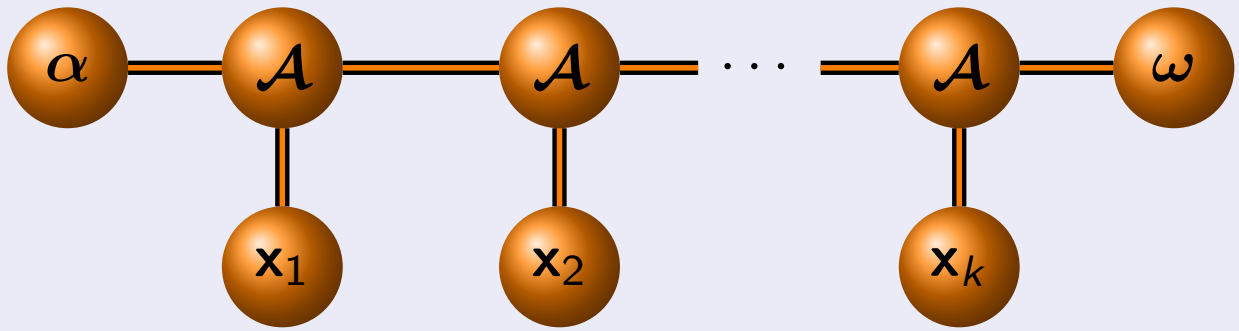
$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector

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

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



Our first result: 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!

Our first result: 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]

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

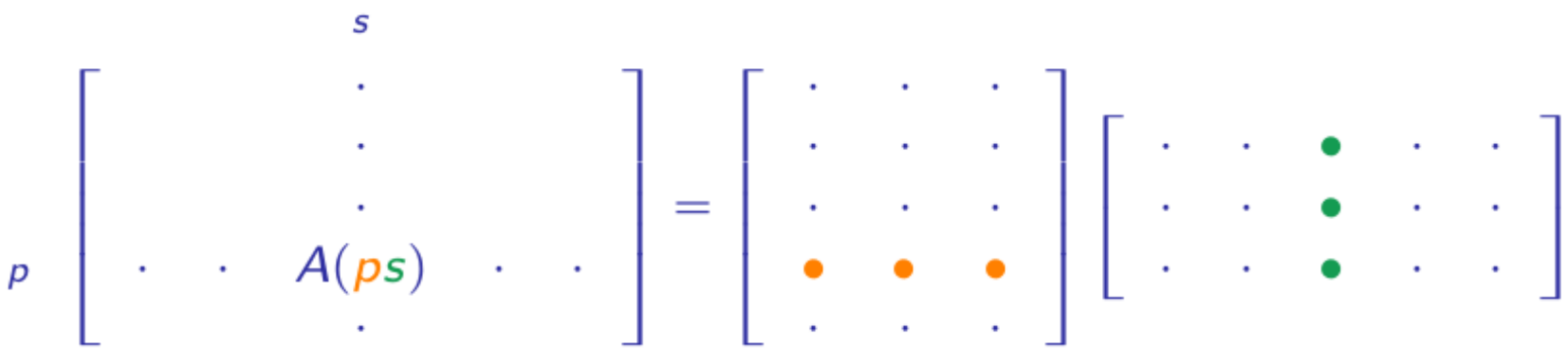
- Fundamental theorem [Carlyle and Paz, 1971; Fliess 1974]:

  $\text{rank}(\mathbf{H}_f) < \infty \iff f$ 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

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



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

Hankel matrix: spectral learning

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

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

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

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

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, ...].

Our second result: a consistent learning algorithm for linear 2-RNNs

Our second result: a consistent learning algorithm for linear 2-RNNs

Theorem

The spectral learning algorithm is a consistent learning algorithm for probability distributions over ***sequences of discrete symbols*** *computed by second-order RNNs with linear activation functions.*

Spectral Learning of Linear 2-RNNs

Learning linear 2-RNNs

Problem: learn a linear 2-RNNs from training data.

Learning linear 2-RNNs

Problem: learn a linear 2-RNNs from training data.

If inputs are one-hot encodings, we can use the spectral learning algorithm for WAs...

$\hookrightarrow$ What about sequences of **continuous** vectors?

Learning linear 2-RNNs

Problem: learn a linear 2-RNNs from training data.

If inputs are one-hot encodings, we can use the spectral learning algorithm for WAs...

$\hookrightarrow$ What about sequences of **continuous** vectors?

What would be the equivalent of the Hankel matrix for $f : (\mathbb{R}^d)^* \to \mathbb{R}$?

$$\mathbf{H} \in \mathbb{R}^{(\mathbb{R}^d)^* \times (\mathbb{R}^d)^*} \quad ?$$

Multi-linearity of linear 2-RNNs

Observation: Linear 2-RNNs are multilinear.

$$f(\mathbf{x}_1, \ldots, \sum_i \alpha_i \mathbf{u}_i, \ldots, \mathbf{x}_k) = \sum_i \alpha_i f(\mathbf{x}_1, \ldots, \mathbf{u}_i, \ldots, \mathbf{x}_k)$$

Multi-linearity of linear 2-RNNs

Observation: Linear 2-RNNs are multilinear.

$$f(\mathbf{x}_1, \ldots, \sum_i \alpha_i \mathbf{u}_i, \ldots, \mathbf{x}_k) = \sum_i \alpha_i f(\mathbf{x}_1, \ldots, \mathbf{u}_i, \ldots, \mathbf{x}_k)$$

$\Rightarrow$ learning the restriction of f to basis vectors is enough:

$$f(\mathbf{a}, \mathbf{b}) = f\left(\sum_i \alpha_i \mathbf{e}_i, \sum_j \beta_j \mathbf{e}_j\right) = \sum_{i,j} \alpha_i \beta_j f(\mathbf{e}_i, \mathbf{e}_j)$$

Multi-linearity of linear 2-RNNs

Observation: Linear 2-RNNs are multilinear.

$$f(\mathbf{x}_1, \ldots, \sum_i \alpha_i \mathbf{u}_i, \ldots, \mathbf{x}_k) = \sum_i \alpha_i f(\mathbf{x}_1, \ldots, \mathbf{u}_i, \ldots, \mathbf{x}_k)$$

$\Rightarrow$ learning the restriction of f to basis vectors is enough:

$$f(\mathbf{a}, \mathbf{b}) = f\left(\sum_i \alpha_i \mathbf{e}_i, \sum_j \beta_j \mathbf{e}_j\right) = \sum_{i,j} \alpha_i \beta_j f(\mathbf{e}_i, \mathbf{e}_j)$$

We only need to learn the function $\tilde{f} : \{1, 2, \cdots, d\}^* \to \mathbb{R}$

$$\tilde{f} : i_1 i_2 \cdots i_k \mapsto f(\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \ldots, \mathbf{e}_{i_k})$$

Idea: Use the spectral learning algorithm to learn $\tilde{f}$.

Hankel Matrix Recovery from Linear Measurements

Choosing $\mathcal{P} = \mathcal{S} = \{1, \cdots, d\}^L$, we need to estimate the Hankel matrix $\mathbf{H} \in \mathbb{R}^{d^L \times d^L}$ defined by

$$\mathbf{H}_{i_1 i_2 \cdots i_L, j_1 j_2 \cdots j_L} = f(\mathbf{e}_{i_1}, , \ldots, \mathbf{e}_{i_L}, \mathbf{e}_{j_1}, , \ldots, \mathbf{e}_{j_L})$$

$\hookrightarrow$ How to estimate $\mathbf{H}$ from input-output examples?

Hankel Matrix Recovery from Linear Measurements

Choosing $\mathcal{P} = \mathcal{S} = \{1, \cdots, d\}^L$, we need to estimate the Hankel matrix $\mathbf{H} \in \mathbb{R}^{d^L \times d^L}$ defined by

$$\mathbf{H}_{i_1 i_2 \cdots i_L, j_1 j_2 \cdots j_L} = f(\mathbf{e}_{i_1}, , \ldots, \mathbf{e}_{i_L}, \mathbf{e}_{j_1}, , \ldots, \mathbf{e}_{j_L})$$

$\hookrightarrow$ How to estimate $\mathbf{H}$ from input-output examples?

Given an input sequence $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{2L})$ and its output $y \simeq f(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{2L})$ we have

$$\begin{aligned} y &\simeq \sum_{i_1, \cdots, i_{2L}} [\mathbf{x}_1]_{i_1} \ldots [\mathbf{x}_{2L}]_{i_{2L}} f(\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_{2L}}) \\ &= (\mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_{2L})^\top \mathrm{vec}(\mathbf{H}) \end{aligned}$$

$\Rightarrow$ Each input-output example is a linear measurement of $\mathbf{H}$.

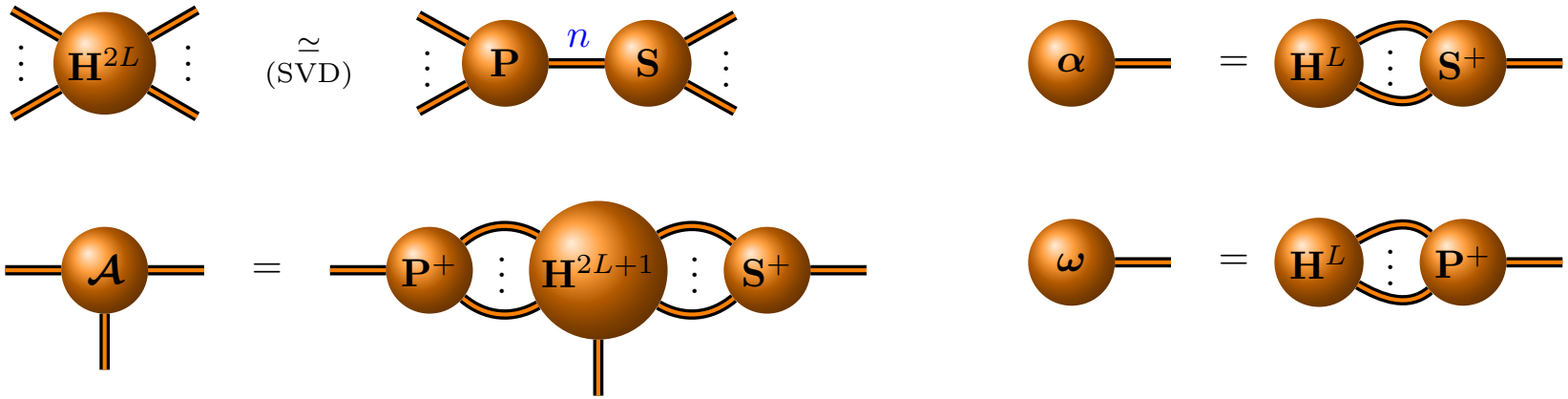
Learning Algorithm

Input: Train datasets D_L, D_{2L}, D_{2L+1}. Number of states n.

1: From $D_l = \{((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), y^{(i)})\}_{i=1}^{N_l} \subset (\mathbb{R}^d)^l \times \mathbb{R}$, solve

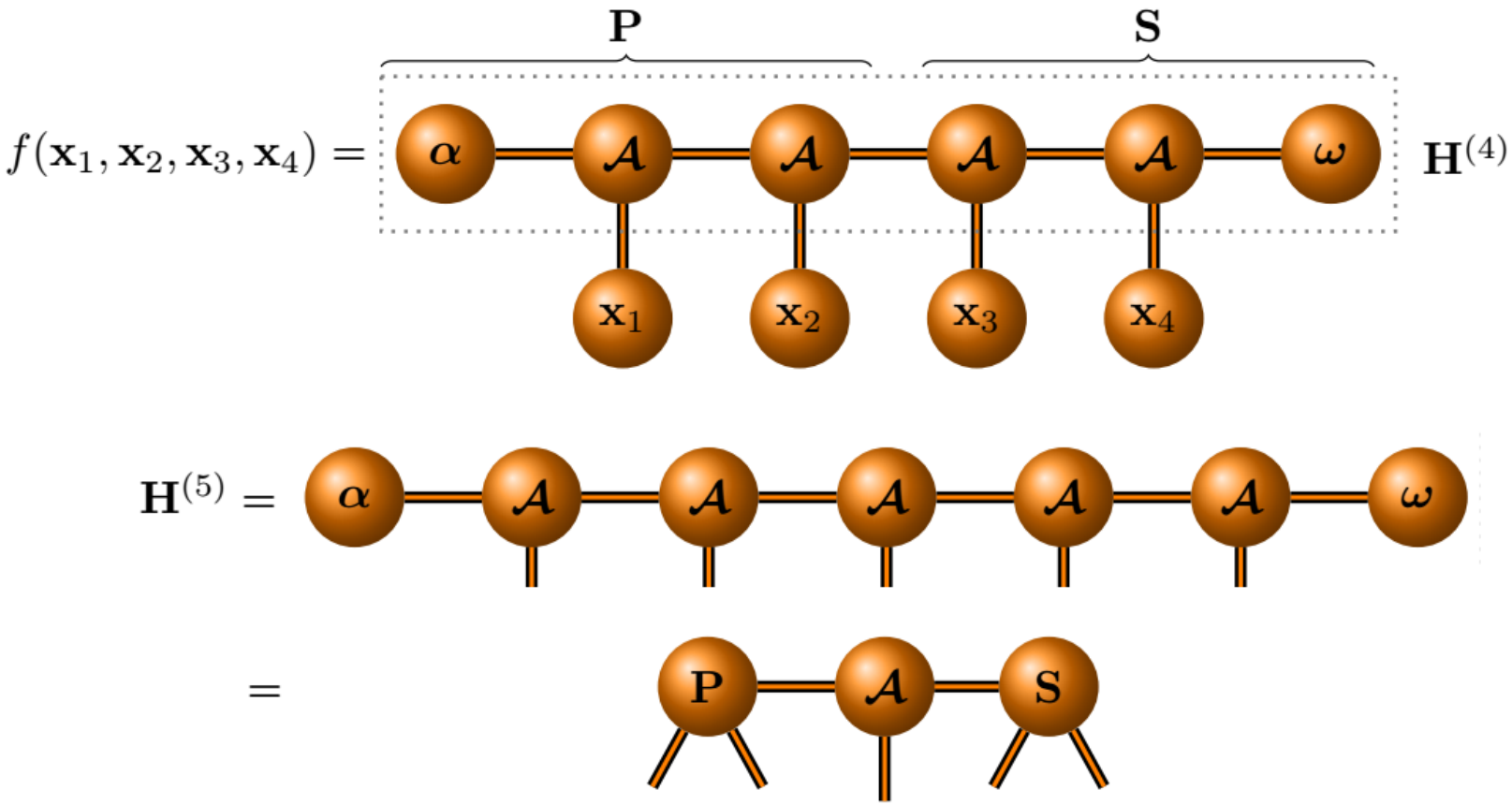
$$\mathbf{H}^{(l)} = \arg\min_{\mathbf{H}} \sum_{i=1}^{N_l} \left(\mathbf{x}_1^{(i)} \otimes \mathbf{x}_2^{(i)} \otimes \cdots \otimes \mathbf{x}_l^{(i)})^\top \mathrm{vec}(\mathbf{H}) - y^{(i)} \right)^2$$

2: Rank n factorization and parameter estimation:



3: **return** Linear 2-RNN $\langle \mathbf{h}_0, \mathcal{A}, \mathbf{w} \rangle$.

Intuition on why this works



Our 3rd result: a provable learning algorithm for 2-RNNs

Our learning algorithm computes a consistent estimator for linear 2-RNNs:

Theorem

- *Let* $(\mathbf{h}_0, \boldsymbol{\mathcal{A}}, \mathbf{w})$ *be a minimal linear 2-RNN with* n *hidden units computing a function* $f : (\mathbb{R}^d)^* \to \mathbb{R}$
- *Let* L *be such that* $\text{rank}(\mathbf{H}^{(2L)}) = n$
- *Suppose the entries of* $\mathbf{x}_j^{(i)}$ *are drawn at random and each* $y^{(i)} = f(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)})$.

If $N_l \geq d^l$ *for* $l = L,\ 2L,\ 2L+1$, *the 2-RNN returned by our algorithm computes* f *with probability one.*

Experiments

Experiment on synthetic data

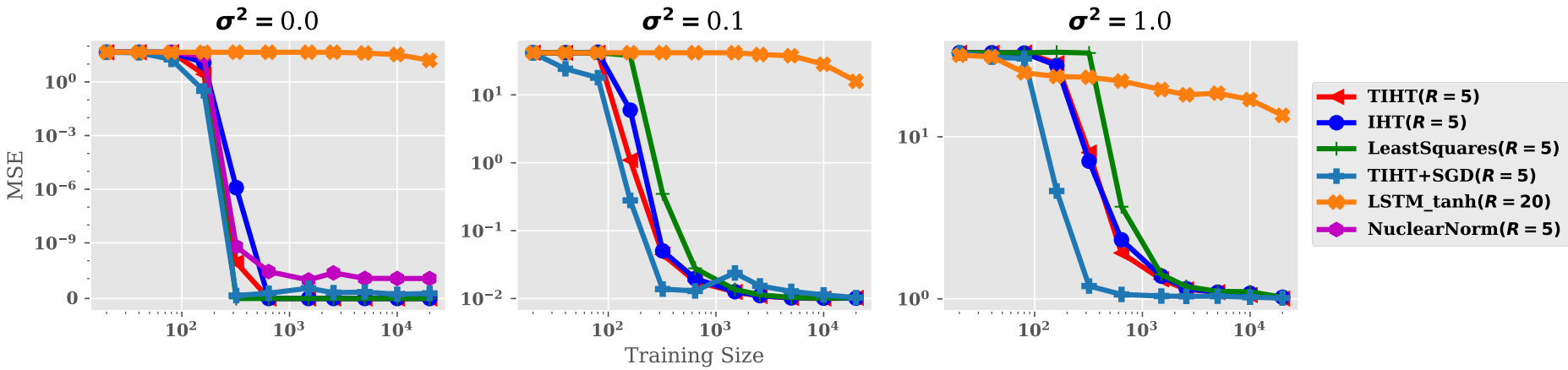


Figure: Learning a random 2-RNN from noisy data

Experiment on real data

Table 1: One-hour-ahead Speed Prediction Performance Comparisons

Method	TIHT	TIHT+SGD	Regression Automata	ARIMA	RNN	Persistence
RMSE	0.573	0.519	0.500	**0.496**	0.606	0.508
MAPE	21.35	18.79	**18.58**	18.74	24.48	18.61
MAE	0.412	0.376	0.363	**0.361**	0.471	0.367

Table 2: Three-hour-ahead Speed Prediction Performance Comparisons

Method	TIHT	TIHT+SGD	Regression Automata	ARIMA	RNN	Persistence
RMSE	0.868	**0.854**	0.872	0.882	1.002	0.893
MAPE	33.98	**31.70**	32.52	33.165	37.24	33.29
MAE	0.632	**0.624**	0.632	0.642	0.764	0.649

Figure: Wind Speed Prediction

Conclusion

Conclusion

- We proposed a natural extension of WA to the continuous case along with a consistent learning algorithm.

Conclusion

- We proposed a natural extension of WA to the continuous case along with a consistent learning algorithm.

- This potentially addresses the limitation of spectral learning to small alphabets (we can now use word embeddings with WA!)
- Opens up a lot of questions on the formal language theory side about continuous WAs.
- Leverage the tensor structure of the Hankel matrix: more structure than just a low rank matrix.
- Use spectral learning as an initialization to BPTT, even for non-linear RNNs!
- Extension to weighted tree automata, polynomial weighted automata, etc.

Conclusion

- We proposed a natural extension of WA to the continuous case along with a consistent learning algorithm.

- This potentially addresses the limitation of spectral learning to small alphabets (we can now use word embeddings with WA!)
- Opens up a lot of questions on the formal language theory side about continuous WAs.
- Leverage the tensor structure of the Hankel matrix: more structure than just a low rank matrix.
- Use spectral learning as an initialization to BPTT, even for non-linear RNNs!
- Extension to weighted tree automata, polynomial weighted automata, etc.

Thank you! Questions?