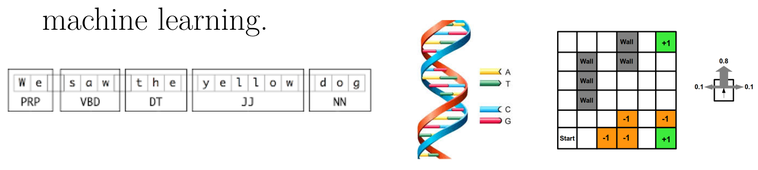
Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning

Guillaume Rabusseau*†§ Tianyu Li*‡ Doina Precup*‡§

* Mila † Université de Montréal ‡ McGill University § Canada CIFAR AI Chair

Introduction

- Sequence data is ubiquitous in computer science and machine learning.

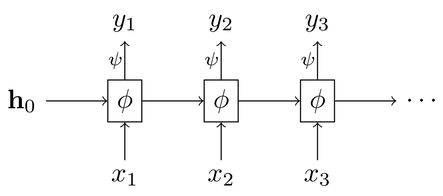


- Weighted Automata (WAs): **tractable** models for sequences of *discrete* data Vs. RNNs: powerful and expressive models.
- This work: connecting WAs and RNNs for fun and profit.

Overview of the Results

- We show the exact equivalence of WAs and 2nd order RNNs (2-RNNs) with linear activation functions.
- 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.

Recurrent Sequential Models



A recurrent sequential model maps sequences of inputs to sequences of outputs

$$x_1, x_2, \ldots, x_k \in \mathcal{X} \quad \longmapsto^{f} \quad y_1, y_2, \ldots, y_k \in \mathcal{Y}$$

by computing a sequence of hidden/latent vectors

$$\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_k \in \mathbb{R}^n, \qquad \mathbf{h}_t = \phi(\mathbf{h}_{t-1}, x_t)$$

for some *recurrent map* $\phi : \mathbb{R}^n \times \mathcal{X} \to \mathbb{R}^n$ and *initial state* $\mathbf{h}_0$. The sequence of outputs is computed with

$$y_t = \psi(\mathbf{h}_t)$$

for some *output map* $\psi : \mathbb{R}^n \to \mathcal{Y}$.

Weighted Automata ≡ linear 2-RNNs

- Vanilla RNN: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$

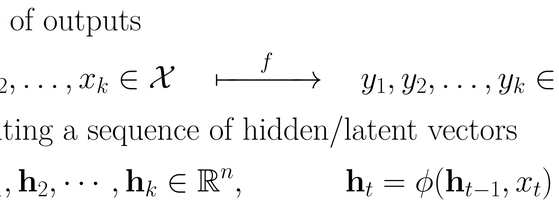
$$\phi(\mathbf{h}_{t-1}, \mathbf{x}_t) = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t) \qquad \psi(\mathbf{h}_t) = h(\mathbf{w}^\top \mathbf{x}_t)$$

where $\mathbf{U} \in \mathbb{R}^{n \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$, $\mathbf{W} \in \mathbb{R}^n$

- Second-order[a] RNN (2-RNN): $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$

$$\phi(\mathbf{h}_{t-1}, \mathbf{x}_t) = g(\mathcal{A} \times_1 \mathbf{h}_{t-1} \times_2 \mathbf{x}_t) \qquad \psi(\mathbf{h}_t) = h(\mathbf{w}^\top \mathbf{x}_t)$$

where $\mathcal{A} \in \mathbb{R}^{n \times d \times n}$.

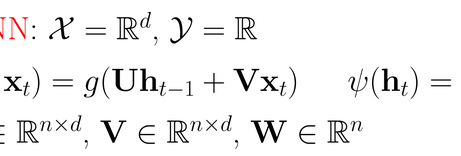


- Weighted Automaton (WA): $\mathcal{X} = \Sigma$, $\mathcal{Y} = \mathbb{R}$

$$\phi(\mathbf{h}_{t-1}, \sigma) = \mathbf{A}^\sigma \, \mathbf{h}_{t-1} \qquad \psi(\mathbf{h}_t) = \mathbf{w}^\top \mathbf{x}_t$$

where $\mathbf{A}^\sigma$ is the *transition matrix* associated with symbol σ for each $\sigma \in \Sigma$.

We can rewrite this as



where $\mathcal{A} \in \mathbb{R}^{n \times \Sigma \times n}$ is defined by

$$\mathcal{A}_{:,\sigma,:} = \mathbf{A}^\sigma \quad \text{for each } \sigma \in \Sigma$$

[a] *Second-order* refers to the order-2 interactions involved in the computation of the latent state: $[\mathcal{A} \times_1 \mathbf{h}_{t-1} \times_2 \mathbf{x}_t]_j = \sum_{i_1, i_2} \mathcal{A}_{i_1, i_2, j} [\mathbf{h}_{t-1}]_{i_1} [\mathbf{x}_t]_{i_2}$

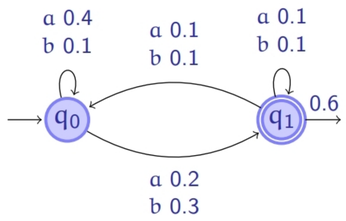
see e.g. [Lee, 86], [Giles, 91], [Pollack, 91], ..., [Wu et al., NIPS'16]

Result 1

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

Weighted Automata

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



Operator Representation

$$\boldsymbol{\alpha}_0 = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} \quad \mathbf{A}^a = \begin{bmatrix} 0.4 & 0.2 \\ 0.1 & 0.1 \end{bmatrix}$$

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

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

$$= \boldsymbol{\alpha}_0^\top \mathbf{A}_a \mathbf{A}_b \boldsymbol{\alpha}_\infty$$

Learning Weighted Automata

The Hankel matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ associated with a function $f : \Sigma^* \to \mathbb{R}$ is defined by $(\mathbf{H})_{u,v} = f(uv)$ for all $u, v \in \Sigma^*$.

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} a & b & aa & \ldots \end{array} \\ \begin{bmatrix} f(aa) & f(ab) & \ldots & \ldots \\ f(ba) & f(bb) & \ldots & \ldots \\ f(aaa) & f(aab) & \ldots & \ldots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array} \quad \mathbf{H}^\sigma = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} a & b & aa & \ldots \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) & \ldots & \ldots \\ f(b\sigma a) & f(b\sigma b) & \ldots & \ldots \\ f(aa\sigma a) & f(aa\sigma b) & \ldots & \ldots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

Theorem [Fliess, 1974] *For any function $f : \Sigma^* \to \mathbb{R}$, $rank(\mathbf{H}_f)$ is finite iff f can be computed by a WA.*

Spectral learning of WAs (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 $\sigma \in \Sigma$, where $(\mathbf{H}^\sigma)_{u,v} = f(u\sigma v)$ for all u, v.
3. Perform rank n decomposition $\mathbf{H} = \mathbf{P}\mathbf{S}$
4. WA with initial/final weights $\mathbf{h}_0 = \mathbf{P}_{\lambda,:}$, $\mathbf{w} = \mathbf{S}_{:,\lambda}$ and transition matrices $\mathbf{A}^\sigma = \mathbf{P}^+ \mathbf{H}^\sigma \mathbf{S}^+$ is a minimal WFA for f.

Two observations to put together:

(i) The spectral learning algorithm is *consistent*.

(ii) Linear 2-RNNs over discrete sequences are WAs.

Result 2

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.

Extension to Continuous Sequences

Problem: learn a linear 2-RNNs from training data.
If inputs are one-hot encodings, we can use the spectral learning algorithm for WAs...

↪ What about sequences of *continuous* vectors?

Observation: Linear 2-RNNs are multilinear.

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

⇒ 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 $\tilde{f} : [d]^* \to \mathbb{R}$

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

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

Hankel Matrix Recovery from Linear Measurements

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

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

↪ 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

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

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

Learning Algorithm

Input: Three training datasets D_L, D_{2L}, D_{2L+1} with input sequences of length L, $2L$ and $2L+1$ respectively. Number of states n.

1: **for** $l \in \{L, 2L, 2L+1\}$ **do**

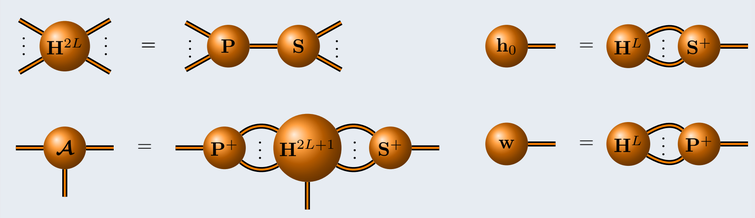
2: From $D_l = \{((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), y^{(i)})\}_{i=1}^{N_l} \subset (\mathbb{R}^d)^l \times \mathbb{R}$ build

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}_1^{(1)} \otimes \mathbf{x}_2^{(1)} \otimes \cdots \otimes \mathbf{x}_l^{(1)})^\top \\ \vdots \\ (\mathbf{x}_1^{(N)} \otimes \mathbf{x}_2^{(N)} \otimes \cdots \otimes \mathbf{x}_l^{(N)})^\top \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

3: $\mathbf{H}^{(l)} = \arg\min_{\mathbf{H}} \|\mathbf{X}\mathrm{vec}(\mathbf{H}) - \mathbf{y}\|_F^2$

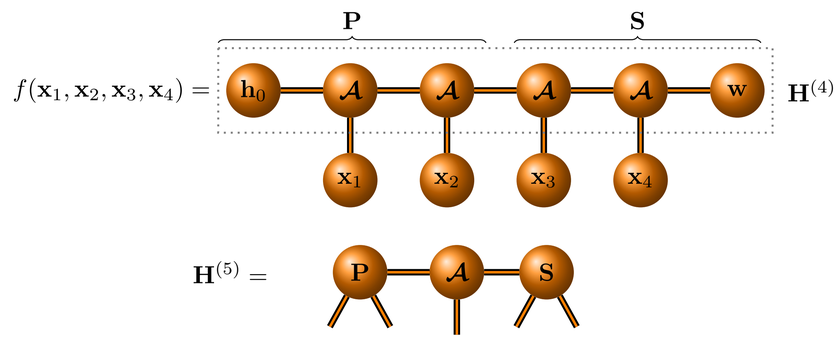
4: **end for**

5: Rank n factorization and parameter estimation:



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

Intuition on why this works



Result 3

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

Theorem

- Let $(\mathbf{h}_0, \mathcal{A}, \mathbf{w})$ 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 $\mathrm{rank}(\mathbf{H}^{(2L)}) = n$
- 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.

Experiment

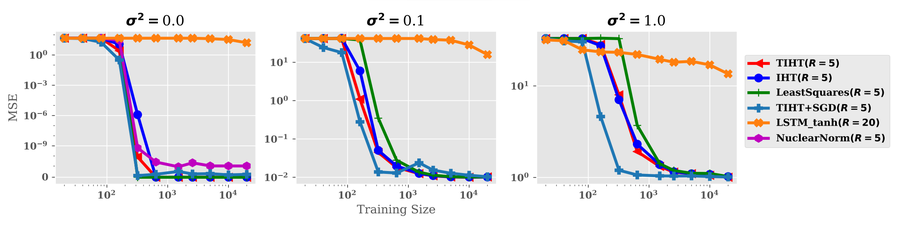


Figure 1: Learning a random 2-RNN from noisy data