Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning

Guillaume Rabusseau[*†] grabus@iro.umontreal.ca

Tianyu Li[*‡] tianyu.li@mail.mcgill.ca

Doina Precup[*‡] dprecup@cs.mcgill.ca

Abstract

In this paper, we unravel a fundamental connection between weighted finite automata (WFAs) and second-order recurrent neural networks (2-RNNs): in the case of sequences of discrete symbols, WFAs and 2-RNNs with linear activation functions are expressively equivalent. Motivated by this result, we build upon a recent extension of the spectral learning algorithm to vector-valued WFAs and propose the first provable learning algorithm for linear 2-RNNs defined over sequences of continuous input vectors. This algorithm relies on estimating low rank subblocks of the so-called Hankel tensor, from which the parameters of a linear 2-RNN can be provably recovered. The performances of the proposed method are assessed in a simulation study.

1 Introduction

Many tasks in natural language processing, computational biology, reinforcement learning, and time series analysis rely on learning with sequential data, i.e. estimating functions defined over sequences of observations from training data. Weighted finite automata (WFAs) and recurrent neural networks (RNNs) are two powerful and flexible classes of models which can efficiently represent such functions. On the one hand, WFAs are tractable, they encompass a wide range of machine learning models (they can for example compute any probability distribution defined by a hidden Markov model (HMM) [12] and can model the transition and observation behavior of partially observable Markov decision processes [43]) and they offer appealing theoretical guarantees. In particular, the so-called *spectral methods* for learning HMMs [22], WFAs [4, 5] and related models [18, 7], provide an alternative to Expectation-Maximization based algorithms that is both computationally efficient and consistent. On the other hand, RNNs are remarkably expressive models — they can represent any computable function [41] — and they have successfully tackled many practical problems in speech and audio recognition [19, 31, 15], but their theoretical analysis is difficult. Even though recent work provides interesting results on their expressive power [24, 48] as well as alternative training algorithms coming with learning guarantees [40], the theoretical understanding of RNNs is still limited.

In this work, we bridge a gap between these two classes of models by unraveling a fundamental connection between WFAs and second-order RNNs (2-RNNs): *when considering input sequences of discrete symbols, 2-RNNs with linear activation functions and WFAs are one and the same*, i.e. they are expressively equivalent and there exists a one-to-one mapping between the two classes (moreover, this mapping conserves model sizes). While connections between finite state machines (e.g. deterministic finite automata) and recurrent neural networks have been noticed and investigated in the past (see e.g. [16, 32]), to the best of our knowledge this is the first time that such a rigorous equivalence between linear 2-RNNs and *weighted* automata is explicitly formalized. More precisely, we pinpoint exactly the class of recurrent neural architectures to which weighted automata are equivalent, namely second-order RNNs with linear activation functions. This result naturally leads to the observation that linear 2-RNNs are a natural generalization of WFAs (which take sequences of *discrete* observations as inputs) to sequences of *continuous vectors*, and raises the question of whether the spectral learning algorithm for WFAs can be extended to linear 2-RNNs. The second contribution of this paper is to show that the answer is in the positive: building upon the spectral learning algorithm for vector-valued WFAs introduced recently in [37], *we propose the first*

Proceedings of the 22[nd] International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

[*] Mila [†] Université de Montréal [‡] McGill University

provable learning algorithm for second-order RNNs with linear activation functions. Our learning algorithm relies on estimating sub-blocks of the so-called Hankel tensor, from which the parameters of a 2-linear RNN can be recovered using basic linear algebra operations. One of the key technical difficulties in designing this algorithm resides in estimating these sub-blocks from training data where the inputs are sequences of *continuous* vectors. We leverage multilinear properties of linear 2-RNNs and the fact that the Hankel sub-blocks can be reshaped into higher-order tensors of low tensor train rank (a result we believe is of independent interest) to perform this estimation efficiently using matrix sensing and tensor recovery techniques. As a proof of concept, we validate our theoretical findings in a simulation study on toy examples where we experimentally compare different recovery methods and investigate the robustness of our algorithm to noise and rank mis-specification. We also show that refining the estimator returned by our algorithm using stochastic gradient descent can lead to significant improvements.

Summary of contributions. We formalize a *strict equivalence between weighted automata and second-order RNNs with linear activation functions* (Section 3), showing that linear 2-RNNs can be seen as a natural extension of (vector-valued) weighted automata for input sequences of *continuous* vectors. We then propose a *consistent learning algorithm for linear 2-RNNs* (Section 4). The relevance of our contributions can be seen from two perspectives. First, while learning feed-forward neural networks with linear activation functions is a trivial task (it reduces to linear or reduced-rank regression), this is not at all the case for recurrent architectures with linear activation functions; to the best of our knowledge, our algorithm is the *first consistent learning algorithm for the class of functions computed by linear second-order recurrent networks.* Second, from the perspective of learning weighted automata, we propose a natural extension of WFAs to continuous inputs and *our learning algorithm addresses the long-standing limitation of the spectral learning method to discrete inputs.*

Related work. Combining the spectral learning algorithm for WFAs with matrix completion techniques (a problem which is closely related to matrix sensing) has been theoretically investigated in [6]. An extension of probabilistic transducers to continuous inputs (along with a spectral learning algorithm) has been proposed in [39]. The connections between tensors and RNNs have been previously leveraged to study the expressive power of RNNs in [24] and to achieve model compression in [48, 47, 44]. Exploring relationships between RNNs and automata has recently received a renewed interest [34, 9, 29]. In particular, such connections have been explored for interpretability purposes [45, 3] and the ability of RNNs to learn classes of formal languages has been investigated in [2]. Connections between the tensor train decomposition and WFAs have been previously noticed in [10, 11, 36]. The predictive state RNN model introduced in [13] is closely related to 2-RNNs and the authors propose to use the spectral learning algorithm for predictive state representations to initialize a gradient based algorithm; their approach however comes without theoretical guarantees. Lastly, a provable algorithm for RNNs relying on the tensor method of moments has been proposed in [40] but it is limited to first-order RNNs with quadratic activation functions (which do not encompass linear 2-RNNs).

The proofs of the results given in the paper can be found in the supplementary material.

2 Preliminaries

In this section, we first present basic notions of tensor algebra before introducing second-order recurrent neural network, weighted finite automata and the spectral learning algorithm. We start by introducing some notation. For any integer k we use $[k]$ to denote the set of integers from 1 to k. We use $\lceil l \rceil$ to denote the smallest integer greater or equal to l. For any set $\mathcal{S}$, we denote by $\mathcal{S}^* = \bigcup_{k \in \mathbb{N}} \mathcal{S}^k$ the set of all finite-length sequences of elements of $\mathcal{S}$ (in particular, Σ^* will denote the set of strings on a finite alphabet Σ). We use lower case bold letters for vectors (e.g. $\mathbf{v} \in \mathbb{R}^{d_1}$), upper case bold letters for matrices (e.g. $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$) and bold calligraphic letters for higher order tensors (e.g. $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$). We use $\mathbf{e}_i$ to denote the ith canonical basis vector of $\mathbb{R}^d$ (where the dimension d will always appear clearly from context). The $d \times d$ identity matrix will be written as $\mathbf{I}_d$. The ith row (resp. column) of a matrix $\mathbf{M}$ will be denoted by $\mathbf{M}_{i,:}$ (resp. $\mathbf{M}_{:,i}$). This notation is extended to slices of a tensor in the straightforward way. If $\mathbf{v} \in \mathbb{R}^{d_1}$ and $\mathbf{v}' \in \mathbb{R}^{d_2}$, we use $\mathbf{v} \otimes \mathbf{v}' \in \mathbb{R}^{d_1 \cdot d_2}$ to denote the Kronecker product between vectors, and its straightforward extension to matrices and tensors. Given a matrix $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$, we use $\mathrm{vec}(\mathbf{M}) \in \mathbb{R}^{d_1 \cdot d_2}$ to denote the column vector obtained by concatenating the columns of $\mathbf{M}$. The inverse of $\mathbf{M}$ is denoted by $\mathbf{M}^{-1}$, its Moore-Penrose pseudo-inverse by $\mathbf{M}^{\dagger}$, and the transpose of its inverse by $\mathbf{M}^{-\top}$; the Frobenius norm is denoted by $\|\mathbf{M}\|_F$ and the nuclear norm by $\|\mathbf{M}\|_*$.

Tensors. We first recall basic definitions of tensor algebra; more details can be found in [27]. A *tensor* $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$ can simply be seen as a multidimensional array ($\boldsymbol{\mathcal{T}}_{i_1,\cdots,i_p} : i_n \in [d_n], n \in [p]$). The

mode-n fibers of $\mathcal{T}$ are the vectors obtained by fixing all indices except the nth one, e.g. $\mathcal{T}_{:,i_2,\cdots,i_p} \in \mathbb{R}^{d_1}$. The *nth mode matricization* of $\mathcal{T}$ is the matrix having the mode-n fibers of $\mathcal{T}$ for columns and is denoted by $\mathcal{T}_{(n)} \in \mathbb{R}^{d_n \times d_1 \cdots d_{n-1} d_{n+1} \cdots d_p}$. The vectorization of a tensor is defined by $\text{vec}(\mathcal{T}) = \text{vec}(\mathcal{T}_{(1)})$. In the following $\mathcal{T}$ always denotes a tensor of size $d_1 \times \cdots \times d_p$.

The *mode-n matrix product* of the tensor $\mathcal{T}$ and a matrix $\mathbf{X} \in \mathbb{R}^{m \times d_n}$ is a tensor denoted by $\mathcal{T} \times_n \mathbf{X}$. It is of size $d_1 \times \cdots \times d_{n-1} \times m \times d_{n+1} \times \cdots \times d_p$ and is defined by the relation $\mathcal{Y} = \mathcal{T} \times_n \mathbf{X} \Leftrightarrow \mathcal{Y}_{(n)} = \mathbf{X}\mathcal{T}_{(n)}$. The *mode-n vector product* of the tensor $\mathcal{T}$ and a vector $\mathbf{v} \in \mathbb{R}^{d_n}$ is a tensor defined by $\mathcal{T} \bullet_n \mathbf{v} = \mathcal{T} \times_n \mathbf{v}^\top \in \mathbb{R}^{d_1 \times \cdots \times d_{n-1} \times d_{n+1} \times \cdots \times d_p}$. It is easy to check that the n-mode product satisfies $(\mathcal{T} \times_n \mathbf{A}) \times_n \mathbf{B} = \mathcal{T} \times_n \mathbf{BA}$ where we assume compatible dimensions of the tensor $\mathcal{T}$ and the matrices $\mathbf{A}$ and $\mathbf{B}$.

Given strictly positive integers $n_1, \cdots, n_k$ satisfying $\sum_i n_i = p$, we use the notation $(\mathcal{T})_{\langle\langle n_1, n_2, \cdots, n_k \rangle\rangle}$ to denote the kth order tensor obtained by reshaping $\mathcal{T}$ into a tensor[1] of size $(\prod_{i_1=1}^{n_1} d_{i_1}) \times (\prod_{i_2=1}^{n_2} d_{n_1+i_2}) \times \cdots \times (\prod_{i_k=1}^{n_k} d_{n_1+\cdots+n_{k-1}+i_k})$. In particular we have $(\mathcal{T})_{\langle\langle p \rangle\rangle} = \text{vec}(\mathcal{T})$ and $(\mathcal{T})_{\langle\langle 1, p-1 \rangle\rangle} = \mathcal{T}_{(1)}$.

A rank R *tensor train (TT) decomposition* [33] of a tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$ consists in factorizing $\mathcal{T}$ into the product of p core tensors $\mathcal{G}_1 \in \mathbb{R}^{d_1 \times R}, \mathcal{G}_2 \in \mathbb{R}^{R \times d_2 \times R}, \ldots, \mathcal{G}_{p-1} \in \mathbb{R}^{R \times d_{p-1} \times R}, \mathcal{G}_p \in \mathbb{R}^{R \times d_p}$, and is defined[2] by

$$\mathcal{T}_{i_1,\cdots,i_p} = \quad (\mathcal{G}_1)_{i_1,:}(\mathcal{G}_2)_{:,i_2,:} \cdots (\mathcal{G}_{p-1})_{:,i_{p-1},:}(\mathcal{G}_p)_{:,i_p}$$

for all indices $i_1 \in [d_1], \cdots, i_p \in [d_p]$; we will use the notation $\mathcal{T} = [\![\mathcal{G}_1, \cdots, \mathcal{G}_p]\!]$ to denote such a decomposition. A tensor network representation of this decomposition is shown in Figure 1. While the problem of finding the best approximation of TT-rank R of a given tensor is NP-hard [20], a quasi-optimal SVD based compression algorithm (TT-SVD) has been proposed in [33]. It is worth mentioning that the TT decomposition is invariant under change of basis: for any invertible matrix $\mathbf{M}$ and any core tensors $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_p$, we have $[\![\mathcal{G}_1, \cdots, \mathcal{G}_p]\!] = [\![\mathcal{G}_1 \times_2 \mathbf{M}^{-\top}, \mathcal{G}_2 \times_1 \mathbf{M} \times_3 \mathbf{M}^{-\top}, \cdots, \mathcal{G}_{p-1} \times_1 \mathbf{M} \times_3 \mathbf{M}^{-\top}, \mathcal{G}_p \times_1 \mathbf{M}]\!]$.

Second-order RNNs. A *second-order recurrent neural network* (2-RNN) [17, 35, 28][3] with n hidden units can be defined as a tuple $M = (\mathbf{h}_0, \mathcal{A}, \mathbf{\Omega})$ where

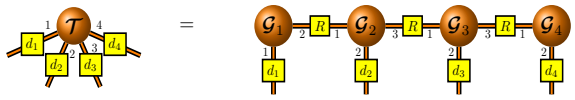

Figure 1: Tensor network representation of a rank R tensor train decomposition (nodes represent tensors and an edge between two nodes represents a contraction between the corresponding modes of the two tensors).

$\mathbf{h}_0 \in \mathbb{R}^n$ is the initial state, $\mathcal{A} \in \mathbb{R}^{n \times d \times n}$ is the transition tensor, and $\mathbf{\Omega} \in \mathbb{R}^{p \times n}$ is the output matrix, with d and p being the input and output dimensions respectively. A 2-RNN maps any sequence of inputs $\mathbf{x}_1, \cdots, \mathbf{x}_k \in \mathbb{R}^d$ to a sequence of outputs $\mathbf{y}_1, \cdots, \mathbf{y}_k \in \mathbb{R}^p$ defined for any $t = 1, \cdots, k$ by

$$\mathbf{y}_t = z_2(\mathbf{\Omega}\mathbf{h}_t) \text{ with } \mathbf{h}_t = z_1(\mathcal{A} \bullet_1 \mathbf{x}_t \bullet_2 \mathbf{h}_{t-1}) \qquad (1)$$

where $z_1 : \mathbb{R}^n \to \mathbb{R}^n$ and $z_2 : \mathbb{R}^p \to \mathbb{R}^p$ are activation functions. Alternatively, one can think of a 2-RNN as computing a function $f_M : (\mathbb{R}^d)^* \to \mathbb{R}^p$ mapping each input sequence $\mathbf{x}_1, \cdots, \mathbf{x}_k$ to the corresponding final output $\mathbf{y}_k$. While z_1 and z_2 are usually non-linear component-wise functions, we consider in this paper the case where both z_1 and z_2 are the identity, and we refer to the resulting model as a *linear 2-RNN*. For a linear 2-RNN M, the function f_M is multilinear in the sense that, for any integer l, its restriction to the domain $(\mathbb{R}^d)^l$ is multilinear. Another useful observation is that linear 2-RNNs are invariant under change of basis: for any invertible matrix $\mathbf{P}$, the linear 2-RNN $\tilde{M} = (\mathbf{P}^{-\top}\mathbf{h}_0, \mathcal{A} \times_1 \mathbf{P} \times_3 \mathbf{P}^{-\top}, \mathbf{P}\mathbf{\Omega})$ is such that $f_{\tilde{M}} = f_M$. A linear 2-RNN M with n states is called *minimal* if its number of hidden units is minimal (i.e. any linear 2-RNN computing f_M has at least n hidden units).

Weighted automata and spectral learning. *Vector-valued weighted finite automaton* (vv-WFA) have been introduced in [37] as a natural generalization of weighted automata from scalar-valued functions to vector-valued ones. A p-dimensional vv-WFA with n states is a tuple $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma \in \Sigma}, \mathbf{\Omega})$ where $\boldsymbol{\alpha} \in \mathbb{R}^n$ is the initial weights vector, $\mathbf{\Omega} \in \mathbb{R}^{p \times n}$ is the matrix of final weights, and $\mathbf{A}^\sigma \in \mathbb{R}^{n \times n}$ is the transition matrix for each symbol σ in a finite alphabet Σ. A vv-WFA A computes a function $f_A : \Sigma^* \to \mathbb{R}^p$ defined by

$$f_A(x) = \mathbf{\Omega}(\mathbf{A}^{x_1}\mathbf{A}^{x_2} \cdots \mathbf{A}^{x_k})^\top \boldsymbol{\alpha}$$

for each word $x = x_1 x_2 \cdots x_k \in \Sigma^*$. We call a vv-WFA *minimal* if its number of states is minimal. Given a function $f : \Sigma^* \to \mathbb{R}^p$ we denote by $\text{rank}(f)$ the number of states of a minimal vv-WFA computing f (which is set to ∞ if f cannot be computed by a vv-WFA).

The spectral learning algorithm for vv-WFAs relies on the following fundamental theorem relating the rank

[1] Note that the specific ordering used to perform matricization, vectorization and such a reshaping is not relevant as long as it is consistent across all operations.

[2] The classical definition of the TT-decomposition allows the rank R to be different for each mode, but this definition is sufficient for the purpose of this paper.

[3] Second-order reccurrent architectures have also been successfully used more recently, see e.g. [42] and [46].

of a function $f : \Sigma^* \to \mathbb{R}^d$ to its Hankel tensor $\mathcal{H} \in \mathbb{R}^{\Sigma^* \times \Sigma^* \times p}$, which is defined by $\mathcal{H}_{u,v,:} = f(uv)$ for all $u, v \in \Sigma^*$.

Theorem 1 ([37]). *Let $f : \Sigma^* \to \mathbb{R}^d$ and let $\mathcal{H}$ be its Hankel tensor. Then* $\text{rank}(f) = \text{rank}(\mathcal{H}_{(1)})$.

The vv-WFA learning algorithm leverages the fact that the proof of this theorem is constructive: one can recover a vv-WFA computing f from any low rank factorization of $\mathcal{H}_{(1)}$. In practice, a finite sub-block $\mathcal{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P} \times \mathcal{S} \times p}$ of the Hankel tensor is used to recover the vv-WFA, where $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ are finite sets of prefixes and suffixes forming a *complete basis* for f, i.e. such that $\text{rank}((\mathcal{H}_{\mathcal{P},\mathcal{S}})_{(1)}) = \text{rank}(\mathcal{H}_{(1)})$. More details can be found in [37].

3 A Fundamental Relation between WFAs and Linear 2-RNNs

We start by unraveling a fundamental connection between vv-WFAs and linear 2-RNNs: vv-WFAs and linear 2-RNNs are expressively equivalent for representing functions defined over sequences of discrete symbols. Moreover, both models have the same capacity in the sense that there is a direct correspondence between the hidden units of a linear 2-RNN and the states of a vv-WFA computing the same function. More formally, we have the following theorem.

Theorem 2. *Any function that can be computed by a vv-WFA with n states can be computed by a linear 2-RNN with n hidden units. Conversely, any function that can be computed by a linear 2-RNN with n hidden units on sequences of one-hot vectors (i.e. canonical basis vectors) can be computed by a WFA with n states.*

More precisely, the WFA $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\Omega})$ with n states and the linear 2-RNN $M = (\boldsymbol{\alpha}, \boldsymbol{\mathcal{A}}, \boldsymbol{\Omega})$ with n hidden units, where $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n \times \Sigma \times n}$ is defined by $\boldsymbol{\mathcal{A}}_{:,\sigma,:} = \mathbf{A}^\sigma$ for all $\sigma \in \Sigma$, are such that $f_A(\sigma_1 \sigma_2 \cdots \sigma_k) = f_M(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)$ for all sequences of input symbols $\sigma_1, \cdots, \sigma_k \in \Sigma$, where for each $i \in [k]$ the input vector $\mathbf{x}_i \in \mathbb{R}^\Sigma$ is the one-hot encoding of the symbol σ_i.

This result first implies that linear 2-RNNs defined over sequence of discrete symbols (using one-hot encoding) *can be provably learned using the spectral learning algorithm for WFAs/vv-WFAs*; indeed, these algorithms have been proved to return consistent estimators. Let us stress again that, contrary to the case of feed-forward architectures, learning recurrent networks with linear activation functions is not a trivial task. Furthermore, Theorem 2 reveals that linear 2-RNNs are a natural generalization of classical weighted automata to functions defined over sequences of continuous vectors (instead of discrete symbols). This spontaneously raises the question of whether the spectral learning algorithms for WFAs and vv-WFAs can be extended to the general setting of linear 2-RNNs; we show that the answer is in the positive in the next section.

4 Spectral Learning of Linear 2-RNNs

In this section, we extend the learning algorithm for vv-WFAs to linear 2-RNNs, thus at the same time addressing the limitation of the spectral learning algorithm to discrete inputs and providing the first consistent learning algorithm for linear second-order RNNs.

4.1 Recovering 2-RNNs from Hankel Tensors

We first present an identifiability result showing how one can recover a linear 2-RNN computing a function $f : (\mathbb{R}^d)^* \to \mathbb{R}^p$ from observable tensors extracted from some Hankel tensor associated with f. Intuitively, we obtain this result by reducing the problem to the one of learning a vv-WFA. This is done by considering the restriction of f to canonical basis vectors; loosely speaking, since the domain of this restricted function is isomorphic to $[d]^*$, this allows us to fall back onto the setting of sequences of discrete symbols.

Given a function $f : (\mathbb{R}^d)^* \to \mathbb{R}^p$, we define its Hankel tensor $\mathcal{H}_f \in \mathbb{R}^{[d]^* \times [d]^* \times p}$ by

$$(\mathcal{H}_f)_{i_1 \cdots i_s, j_1 \cdots j_t, :} = f(\mathbf{e}_{i_1}, \cdots, \mathbf{e}_{i_s}, \mathbf{e}_{j_1}, \cdots, \mathbf{e}_{j_t}),$$

for all $i_1, \cdots, i_s, j_1, \cdots, j_t \in [d]$, which is infinite in two of its modes. It is easy to see that $\mathcal{H}_f$ is also the Hankel tensor associated with the function $\tilde{f} : [d]^* \to \mathbb{R}^p$ mapping any sequence $i_1 i_2 \cdots i_k \in [d]^*$ to $f(\mathbf{e}_{i_1}, \cdots, \mathbf{e}_{i_k})$. Moreover, in the special case where f can be computed by a linear 2-RNN, one can use the multilinearity of f to show that $f(\mathbf{x}_1, \cdots, \mathbf{x}_k) = \sum_{i_1, \cdots, i_k = 1}^{d} (\mathbf{x}_1)_{i_1} \cdots (\mathbf{x}_l)_{i_k} \tilde{f}(i_1 \cdots i_k)$, giving us some intuition on how one could learn f by learning a vv-WFA computing $\tilde{f}$ using the spectral learning algorithm. That is, given a large enough sub-block $\mathcal{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P} \times \mathcal{S} \times p}$ of $\mathcal{H}_f$ for some prefix and suffix sets $\mathcal{P}, \mathcal{S} \subseteq [d]^*$, one should be able to recover a vv-WFA computing $\tilde{f}$ and consequently a linear 2-RNN computing f using Theorem 2. Before devoting the remaining of this section to formalize this intuition (leading to Theorem 3), it is worth observing that while this approach is sound, it is not realistic since it requires observing entries of the Hankel tensor $\mathcal{H}_f$, which implies having access to input/output examples where the inputs are *sequences of canonical basis vectors*; This issue will be discussed in more details and addressed in the next section.

For the sake of clarity, we present the learning algorithm for the particular case where there exists an L

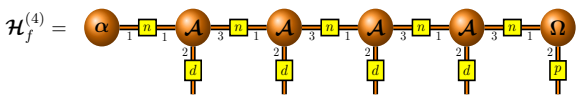

Figure 2: Tensor network representation of the TT decomposition of the Hankel tensor $\mathcal{H}_f^{(4)}$ induced by a linear 2-RNN $(\boldsymbol{\alpha}, \mathcal{A}, \boldsymbol{\Omega})$.

such that the prefix and suffix sets consisting of all sequences of length L, that is $\mathcal{P} = \mathcal{S} = [d]^L$, forms a complete basis for $\tilde{f}$ (i.e. the sub-block $\mathcal{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{[d]^L \times [d]^L \times p}$ of the Hankel tensor $\mathcal{H}_f$ is such that $\text{rank}((\mathcal{H}_{\mathcal{P},\mathcal{S}})_{(1)}) = \text{rank}((\mathcal{H}_f)_{(1)})$). This assumption allows us to present all the key elements of the algorithm in a simpler way, the technical details needed to lift this assumption are given in the supplementary material.

For any integer l, we define the finite tensor $\mathcal{H}_f^{(l)} \in \mathbb{R}^{d \times \cdots \times d \times p}$ of order $l+1$ by

$$(\mathcal{H}_f^{(l)})_{i_1,\cdots,i_l,:} = f(\mathbf{e}_{i_1}, \cdots, \mathbf{e}_{i_l}) \quad \text{for all } i_1, \cdots, i_l \in [d].$$

Observe that for any integer l, the tensor $\mathcal{H}_f^{(l)}$ can be obtained by reshaping a finite sub-block of the Hankel tensor $\mathcal{H}_f$. When f is computed by a linear 2-RNN, we have the useful property that, for any integer l,

$$f(\mathbf{x}_1, \cdots, \mathbf{x}_l) = \mathcal{H}_f^{(l)} \bullet_1 \mathbf{x}_1 \bullet_2 \cdots \bullet_l \mathbf{x}_l \tag{2}$$

for any sequence of inputs $\mathbf{x}_1, \cdots, \mathbf{x}_l \in \mathbb{R}^d$ (which can be shown using the multilinearity of f). Another fundamental property of the tensors $\mathcal{H}_f^{(l)}$ is that they are of low tensor train rank. Indeed, for any l, one can check that $\mathcal{H}_f^{(l)} = [\![\mathcal{A} \bullet_1 \boldsymbol{\alpha}, \underbrace{\mathcal{A}, \cdots, \mathcal{A}}_{l-1 \text{ times}}, \boldsymbol{\Omega}^\top]\!]$ (the tensor network representation of this decomposition is shown in Figure 2). This property will be particularly relevant to the learning algorithm we design in the following section, but it is also a fundamental relation that deserves some attention on its own: it implies in particular that, beyond the classical relation between the rank of the Hankel matrix $\mathbf{H}_f$ and the number states of a minimal WFA computing f, the Hankel matrix possesses a deeper structure intrinsically connecting weighted automata to the tensor train decomposition. We now state the main result of this section, showing that a (minimal) linear 2-RNN computing a function f can be exactly recovered from sub-blocks of the Hankel tensor $\mathcal{H}_f$.

Theorem 3. *Let $f : (\mathbb{R}^d)^* \to \mathbb{R}^p$ be a function computed by a minimal linear 2-RNN with n hidden units and let L be an integer such that $\text{rank}((\mathcal{H}_f^{(2L)})_{\langle\langle L, L+1 \rangle\rangle}) = n$. Then, for any $\mathbf{P} \in \mathbb{R}^{d^L \times n}$ and $\mathbf{S} \in \mathbb{R}^{n \times d^L p}$ such that $(\mathcal{H}_f^{(2L)})_{\langle\langle L, L+1 \rangle\rangle} = \mathbf{PS}$, the linear 2-RNN $M = (\boldsymbol{\alpha}, \mathcal{A}, \boldsymbol{\Omega})$ defined by*

$$\boldsymbol{\alpha} = (\mathbf{S}^\dagger)^\top (\mathcal{H}_f^{(L)})_{\langle\langle L+1 \rangle\rangle}, \qquad \boldsymbol{\Omega}^\top = \mathbf{P}^\dagger (\mathcal{H}_f^{(L)})_{\langle\langle L,1 \rangle\rangle}$$

$$\mathcal{A} = ((\mathcal{H}_f^{(2L+1)})_{\langle\langle L,1,L+1 \rangle\rangle}) \times_1 \mathbf{P}^\dagger \times_3 (\mathbf{S}^\dagger)^\top$$

is a minimal linear 2-RNN computing f.

First observe that such an integer L exists under the assumption that $\mathcal{P} = \mathcal{S} = [d]^L$ forms a complete basis for $\tilde{f}$. It is also worth mentioning that a necessary condition for $\text{rank}((\mathcal{H}_f^{(2L)})_{\langle\langle L, L+1 \rangle\rangle}) = n$ is that $d^L \geq n$, i.e. L must be of the order $\log_d(n)$.

4.2 Hankel Tensors Recovery from Linear Measurements

We showed in the previous section that, given the Hankel tensors $\mathcal{H}_f^{(L)}$, $\mathcal{H}_f^{(2L)}$ and $\mathcal{H}_f^{(2L+1)}$, one can recover a linear 2-RNN computing f if it exists. This first implies that the class of functions that can be computed by linear 2-RNNs is learnable in Angluin's exact learning model [1] where one has access to an oracle that can answer membership queries (e.g. *what is the value computed by the target f on $(\mathbf{x}_1, \cdots, \mathbf{x}_k)$?*) and equivalence queries (e.g. *is my current hypothesis h equal to the target f?*). While this fundamental result is of significant theoretical interest, assuming access to such an oracle is unrealistic. In this section, we show that a stronger learnability result can be obtained in a more realistic setting, where we only assume access to randomly generated input/output examples $((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), \mathbf{y}^{(i)}) \in (\mathbb{R}^d)^* \times \mathbb{R}^p$ where $\mathbf{y}^{(i)} = f(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)})$.

The key observation is that such an input/output example $((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), \mathbf{y}^{(i)})$ can be seen as a *linear measurement* of the Hankel tensor $\mathcal{H}^{(l)}$. Indeed, we have

$$\begin{aligned}\mathbf{y}^{(i)} &= f(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}) = \mathcal{H}_f^{(l)} \bullet_1 \mathbf{x}_1 \bullet_2 \cdots \bullet_l \mathbf{x}_l \\ &= (\mathcal{H}^{(l)})_{\langle\langle l,1 \rangle\rangle}^\top \mathbf{x}^{(i)}\end{aligned}$$

where $\mathbf{x}^{(i)} = \mathbf{x}_1^{(i)} \otimes \cdots \otimes \mathbf{x}_l^{(i)} \in \mathbb{R}^{d^l}$. Hence, by regrouping N output examples $\mathbf{y}^{(i)}$ into the matrix $\mathbf{Y} \in \mathbb{R}^{N \times p}$ and the corresponding input vectors $\mathbf{x}^{(i)}$ into the matrix $\mathbf{X} \in \mathbb{R}^{N \times d^l}$, one can recover $\mathcal{H}^{(l)}$ by solving the linear system $\mathbf{Y} = \mathbf{X}(\mathcal{H}^{(l)})_{\langle\langle l,1 \rangle\rangle}$, which has a unique solution whenever $\mathbf{X}$ is of full column rank. This naturally leads to the following theorem, whose proof relies on the fact that $\mathbf{X}$ will be of full column rank whenever $N \geq d^l$

and the components of each $\mathbf{x}_j^{(i)}$ for $j \in [l], i \in [N]$ are drawn independently from a continuous distribution over $\mathbb{R}^d$ (w.r.t. the Lebesgue measure).

Theorem 4. *Let $(\mathbf{h}_0, \mathcal{A}, \mathbf{\Omega})$ be a minimal linear 2-RNN with n hidden units computing a function $f : (\mathbb{R}^d)^* \to \mathbb{R}^p$, and let L be an integer[4] such that $\text{rank}((\mathcal{H}_f^{(2L)})_{\langle\langle L, L+1\rangle\rangle}) = n$. Suppose we have access to 3 datasets $D_l = \{((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), \mathbf{y}^{(i)})\}_{i=1}^{N_l} \subset (\mathbb{R}^d)^l \times \mathbb{R}^p$ for $l \in \{L, 2L, 2L+1\}$ where the entries of each $\mathbf{x}_j^{(i)}$ are drawn independently from the standard normal distribution and where each $\mathbf{y}^{(i)} = f(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)})$.*

Then, if $N_l \geq d^l$ for $l = L,\ 2L,\ 2L+1$, the linear 2-RNN M returned by Algorithm 1 with the least-squares method satisfies $f_M = f$ with probability one.

A few remarks on this theorem are in order. The first observation is that the 3 datasets D_L, D_{2L} and D_{2L+1} can either be drawn independently or not (e.g. the sequences in D_L can be prefixes of the sequences in D_{2L} but it is not necessary). In particular, the result still holds when the datasets D_l are constructed from a unique dataset $S = \{((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_T^{(i)}), (\mathbf{y}_1^{(i)}, \mathbf{y}_2^{(i)}, \cdots, \mathbf{y}_T^{(i)}))\}_{i=1}^N$ of input/output sequences with $T \geq 2L+1$, where $\mathbf{y}_t^{(i)} = f(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_t^{(i)})$ for any $t \in [T]$. Observe that having access to such input/output training sequences is not an unrealistic assumption: for example when training RNNs for language modeling the output $\mathbf{y}_t$ is the conditional probability vector of the next symbol, and for classification tasks the output is the one-hot encoded label for all time steps. Lastly, when the outputs $\mathbf{y}^{(i)}$ are noisy, one can solve the least-squares problem $\|\mathbf{Y} - \mathbf{X}(\mathcal{H}^{(l)})_{\langle\langle l,1\rangle\rangle}\|_F^2$ to approximate the Hankel tensors; we will empirically evaluate this approach in Section 5 and we defer its theoretical analysis in the noisy setting to future work.

4.3 Leveraging the low rank structure of the Hankel tensors

While the least-squares method is sufficient to obtain the theoretical guarantees of Theorem 4, it does not leverage the low rank structure of the Hankel tensors $\mathcal{H}^{(L)}$, $\mathcal{H}^{(2L)}$ and $\mathcal{H}^{(2L+1)}$. We now propose three alternative recovery methods to leverage this structure, whose sample efficiency will be assessed in a simulation study in Section 5 (deriving improved sample complexity guarantees using these methods is left for future work). In the noiseless setting, we first propose to replace solving the linear system

[4]Note that the theorem can be adapted if such an integer L does not exists (see supplementary material).

Algorithm 1 `2RNN-SL`: Spectral Learning of linear 2-RNNs

Input: Three training datasets D_L, D_{2L}, D_{2L+1} with input sequences of length L, $2L$ and $2L+1$ respectively, a `recovery_method`, rank R and learning rate γ (for IHT/TIHT).

1: **for** $l \in \{L, 2L, 2L+1\}$ **do**
2: Use $D_l = \{((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), \mathbf{y}^{(i)})\}_{i=1}^{N_l} \subset (\mathbb{R}^d)^l \times \mathbb{R}^p$ to build $\mathbf{X} \in \mathbb{R}^{N_l \times d^l}$ with rows $\mathbf{x}_1^{(i)} \otimes \mathbf{x}_2^{(i)} \otimes \cdots \otimes \mathbf{x}_l^{(i)}$ for $i \in [N_l]$ and $\mathbf{Y} \in \mathbb{R}^{N_l \times p}$ with rows $\mathbf{y}^{(i)}$ for $i \in [N_l]$.
3: **if** `recovery_method` = "Least-Squares" **then**
4: $\mathcal{H}^{(l)} = \underset{\mathcal{T} \in \mathbb{R}^{d \times \cdots \times d \times p}}{\arg\min} \|\mathbf{X}(\mathcal{T})_{\langle\langle l,1\rangle\rangle} - \mathbf{Y}\|_F^2$.
5: **else if** `recovery_method` = "Nuclear Norm" **then**
6: $\mathcal{H}^{(l)} = \underset{\mathcal{T} \in \mathbb{R}^{d \times \cdots \times d \times p}}{\arg\min} \|(\mathcal{T})_{\langle\langle \lceil l/2 \rceil, l - \lceil l/2 \rceil + 1\rangle\rangle}\|_*$ subject to $\mathbf{X}(\mathcal{T})_{\langle\langle l,1\rangle\rangle} = \mathbf{Y}$.
7: **else if** `recovery_method` = "(T)IHT" **then**
8: Initialize $\mathcal{H}^{(l)} \in \mathbb{R}^{d \times \cdots \times d \times p}$ to $\mathbf{0}$.
9: **repeat**
10: $(\mathcal{H}^{(l)})_{\langle\langle l,1\rangle\rangle} = (\mathcal{H}^{(l)})_{\langle\langle l,1\rangle\rangle} + \gamma \mathbf{X}^\top (\mathbf{Y} - \mathbf{X}(\mathcal{H}^{(l)})_{\langle\langle l,1\rangle\rangle})$
11: $\mathcal{H}^{(l)} = \texttt{project}(\mathcal{H}^{(l)}, R)$ (using either SVD for IHT or TT-SVD for TIHT)
12: **until** convergence
13: Let $(\mathcal{H}^{(2L)})_{\langle\langle L, L+1\rangle\rangle} = \mathbf{PS}$ be a rank R factorization.
14: Return the linear 2-RNN $(\mathbf{h}_0, \mathcal{A}, \mathbf{\Omega})$ where

$$\boldsymbol{\alpha} = (\mathbf{S}^\dagger)^\top (\mathcal{H}_f^{(L)})_{\langle\langle L+1\rangle\rangle}, \qquad \mathbf{\Omega}^\top = \mathbf{P}^\dagger (\mathcal{H}_f^{(L)})_{\langle\langle L,1\rangle\rangle}$$
$$\mathcal{A} = ((\mathcal{H}_f^{(2L+1)})_{\langle\langle L,1,L+1\rangle\rangle}) \times_1 \mathbf{P}^\dagger \times_3 (\mathbf{S}^\dagger)^\top$$

$\mathbf{Y} = \mathbf{X}(\mathcal{H}^{(l)})_{\langle\langle l,1\rangle\rangle}$ with a nuclear norm minimization problem (see line 6 of Algorithm 1), thus leveraging the fact that $(\mathcal{H}^{(l)})_{\langle\langle \lceil l/2 \rceil, l - \lceil l/2 \rceil + 1\rangle\rangle}$ is potentially of low matrix rank. We also propose to use iterative hard thresholding (IHT) [23] and its tensor counterpart TIHT [38], which are based on the classical projected gradient descent algorithm and have shown to be robust to noise in practice. These two methods are implemented in lines 9-12 of Algorithm 1. There, the `project` method either projects $(\mathcal{H}^{(l)})_{\langle\langle \lceil l/2 \rceil, l - \lceil l/2 \rceil + 1\rangle\rangle}$ onto the manifold of low rank matrices using SVD (IHT) or projects $\mathcal{H}^{(l)}$ onto the manifold of tensors with TT-rank R (TIHT).

The low rank structure of the Hankel tensors can also be leveraged to improve the scalability of the learning algorithm. One can check that the computational complexity of Algorithm 1 is exponential in the maximum sequence length: indeed, building the matrix

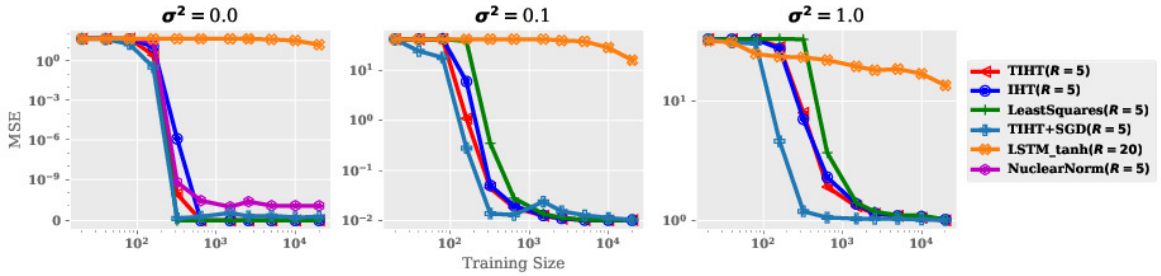

Figure 3: Average MSE as a function of the training set size for the first experiment (learning a random linear 2-RNN) for different values of output noise.

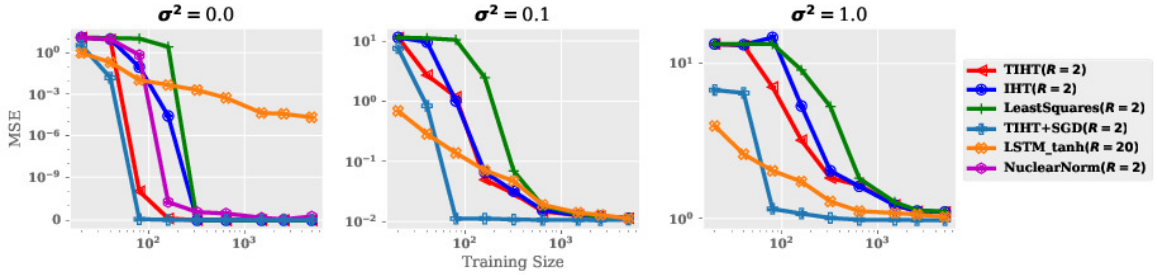

Figure 4: Average MSE as a function of the training set size for the second experiment (learning a simple arithmetic function) for different values of output noise.

$\mathbf{X}$ in line 2 is already in $\mathcal{O}\left(N_l d^l\right)$, where l is in turn equal to L, $2L$ and $2L+1$. Focusing on the TIHT recovery method, a careful analysis shows that the computational complexity of the algorithm is in

$$\mathcal{O}\left(d^{2L+1}\left(p(TN+R)+R^2\right)+TL\max(p,d)^{2L+3}\right),$$

where $N = \max(N_L, N_{2L}, N_{2L+1})$ and T is the number of iterations of the loop on line 9. Thus, in its present form, our approach cannot scale to high dimensional inputs and long sequences. However, one can leverage the low tensor train rank structure of the Hankel tensors to circumvent this issue: by storing both the estimates of the Hankel tensors $\boldsymbol{\mathcal{H}}^{(l)}$ and the matrices $\mathbf{X}$ in TT format (with decompositions of ranks R and N respectively), all the operations needed to implement Algorithm 1 with the TIHT recovery method can be performed in time $\mathcal{O}\left(T(N+R)^3(Ld+p)\right)$ (more details can be found in the supplementary material). By leveraging the tensor train structure, one can thus lift the dependency on d^{2L+1} by paying the price of an increased cubic complexity in the number of examples N and the number of states R. While the dependency on the number of states is not a major issue (R should be negligible w.r.t. N), the dependency on N^3 can quickly become prohibitive for realistic application scenario. Fortunately, this issue can be dealt with by using mini-batches of training data for the gradient updates on line 10 instead of the whole dataset D_l, in which case the overall complexity of Algorithm 1 becomes $\mathcal{O}\left(T(M+R)^3(Ld+p)\right)$ where M is the mini-batch size (the overall algorithm in TT format is summarized in Algorithm 2 in the supplementary material).

5 Experiments

In this section, we perform experiments[5] on two toy examples to compare how the choice of the recovery method (`LeastSquares`, `NuclearNorm`, `IHT` and `TIHT`) affects the sample efficiency of Algorithm 1. Additional experiments on real data can be found in Appendix D. We also include comparisons with RNNs with long short term memory (LSTM) units [21] and report the performances obtained by refining the solution returned by our algorithm (with the TIHT recovery method) using stochastic gradient descent (`TIHT+SGD`).

We perform two experiments. In the first one, we randomly generate a linear 2-RNN with 5 units computing a function $f : \mathbb{R}^3 \to \mathbb{R}^2$ by drawing the entries of all parameters $(\mathbf{h}_0, \boldsymbol{\mathcal{A}}, \boldsymbol{\Omega})$ independently from the normal distribution $\mathcal{N}(0, 0.2)$. The training data consists of 3 independently drawn sets $D_l = \{((\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}), \mathbf{y}^{(i)})\}_{i=1}^{N_l} \subset (\mathbb{R}^d)^l \times \mathbb{R}^p$ for $l \in \{L, 2L, 2L+1\}$ with $L = 2$, where each $\mathbf{x}_j^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and where the outputs can be noisy, i.e. $\mathbf{y}^{(i)} = f(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_l^{(i)}) + \boldsymbol{\xi}^{(i)}$ where $\boldsymbol{\xi}^{(i)} \sim \mathcal{N}(0, \sigma^2)$ for some noise variance σ^2. In the second experiment, the goal is to learn a simple arithmetic function computing the sum of the running differences between the two components of a sequence of 2-dimensional vectors, i.e. $f(\mathbf{x}_1, \cdots, \mathbf{x}_k) = \sum_{i=1}^{k} \mathbf{v}^\top \mathbf{x}_i$ where $\mathbf{v}^\top = (-1 \quad 1)$. The 3 training datasets are generated using the same process as above and a constant entry equal to one is

[5] https://github.com/litianyu1993/learning_2RNN

added to all the input vectors to encode a bias term (one can check that the resulting function can be computed by a linear 2-RNN with 2 hidden units).

We run the experiments for different sizes of training data ranging from $N = 20$ to $N = 20,000$ (we set $N_L = N_{2L} = N_{2L+1} = N$) and we compare the different methods in terms of mean squared error (MSE) on a test set of $1,000$ sequences of length 6 generated in the same way as the training data (note that the training data only contains sequences of length up to 5).

We report the performances of (non-linear) RNNs with a single layer of LSTM with 20 hidden units (with tanh activation functions[6]) and one fully-connected output layer, trained using the Adam optimizer [25] with learning rate 0.001. We also use Adam with learning rate 0.001 to refine the models returned by TIHT with stochastic gradient descent SGD (we tried directly training a linear 2-RNN from random initializations using SGD as well but this approach always failed to return a good model). The IHT/TIHT methods sometimes returned aberrant models (due to numerical instabilities), we used the following scheme to circumvent this issue: when the training MSE of the hypothesis was greater than the one of the zero function, the zero function was returned instead (we applied this scheme to all other methods in the experiments).

The results are reported in Figure 3 and 4 where we see that all recovery methods of Algorithm 1 lead to consistent estimates of the target function given enough training data. This is the case even in the presence of noise (in which case more samples are needed to achieve the same accuracy, as expected). We can also see that `IHT` and `TIHT` are overall more sample efficient than the other methods (especially with noisy data), showing that taking the low rank structure of the Hankel tensors into account is profitable. Moreover, `TIHT` tends to perform better than its matrix counterpart, confirming our intuition that leveraging the tensor train structure is beneficial. While LSTMs obtain good performances on the addition task, they struggle to recover the random linear 2-RNN in the first task (despite our efforts at hyper-parameter tuning and architecture search). In the meantime, refining the TIHT models using SGD almost always leads to significant improvements (especially under the noisy setting), matching or outperforming the performances of RNNs on the two tasks. Lastly, we show the effect of rank mis-specification in Figure 5: as one can expect, when the rank parameter R is over-estimated Algorithm 1 still converges to the target function but it requires more samples (when the rank parameter was underestimated all algorithms did not learn at all).

[6] We also tried training LSTMs with linear recurrent activation functions on the two tasks but they always performed worse than non-linear ones.

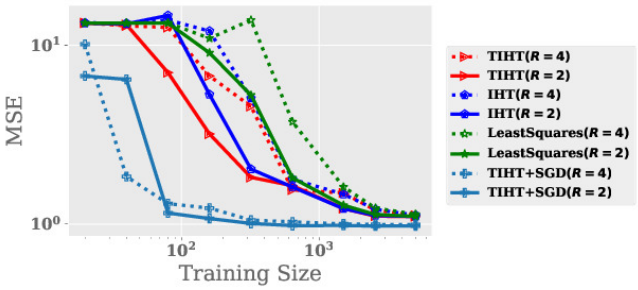

Figure 5: Comparison between different rank settings in terms of average MSE for the second experiment (learning a simple arithmetic function) in the noisy setting ($\sigma^2 = 1$).

6 Conclusion and Future Directions

We proposed the first provable learning algorithm for second-order RNNs with linear activation functions: we showed that linear 2-RNNs are a natural extension of vv-WFAs to the setting of input sequences of *continuous vectors* (rather than discrete symbol) and we extended the vv-WFA spectral learning algorithm to this setting. We believe that the results presented in this paper open a number of exciting and promising research directions on both the theoretical and practical perspectives. We first plan to use the spectral learning estimate as a starting point for gradient based methods to train non-linear 2-RNNs. More precisely, linear 2-RNNs can be thought of as 2-RNNs using LeakyRelu activation functions with negative slope 1, therefore one could use a linear 2-RNN as initialization before gradually reducing the negative slope parameter during training. The extension of the spectral method to linear 2-RNNs also opens the door to scaling up the classical spectral algorithm to problems with large discrete alphabets (which is a known caveat of the spectral algorithm for WFAs) since it allows one to use low dimensional embeddings of large vocabularies (using e.g. word2vec or latent semantic analysis). From the theoretical perspective, we plan on deriving learning guarantees for linear 2-RNNs in the noisy setting (e.g. using the PAC learnability framework). Even though it is intuitive that such guarantees should hold (given the continuity of all operations used in our algorithm), we believe that such an analysis may entail results of independent interest. In particular, analogously to the matrix case studied in [8], obtaining rate optimal convergence rates for the recovery of the low TT-rank Hankel tensors from rank one measurements is an interesting direction; such a result could for example allow one to improve the generalization bounds provided in [6] for spectral learning of general WFAs.

Acknowledgements

This work was done while G. Rabusseau was an IVADO postdoctoral scholar at McGill University.

References

[1] Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[2] Enes Avcu, Chihiro Shibata, and Jeffrey Heinz. Subregular complexity and deep learning. *CLASP Papers in Computational Linguistics*, page 20, 2017.

[3] Stéphane Ayache, Rémi Eyraud, and Noé Goudian. Explaining black boxes on sequential data using weighted automata. In *Proceedings of ICGI*, pages 81–103, 2018.

[4] Raphaël Bailly, François Denis, and Liva Ralaivola. Grammatical inference as a principal component analysis problem. In *Proceedings of ICML*, pages 33–40, 2009.

[5] Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. Spectral learning of weighted automata. *Machine learning*, 96(1-2):33–63, 2014.

[6] Borja Balle and Mehryar Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *Proceedings of NIPS*, pages 2159–2167, 2012.

[7] Byron Boots, Sajid M. Siddiqi, and Geoffrey J. Gordon. Closing the learning-planning loop with predictive state representations. *International Journal of Robotics Research*, 30(7):954–966, 2011.

[8] T Tony Cai, Anru Zhang, et al. Rop: Matrix recovery via rank-one projections. *The Annals of Statistics*, 43(1):102–138, 2015.

[9] Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. Recurrent neural networks as weighted language recognizers. In *Proceedings of NAACL-HLT*, pages 2261–2271, 2018.

[10] Andrew Critch. *Algebraic geometry of hidden Markov and related models*. PhD thesis, University of California, Berkeley, 2013.

[11] Andrew Critch and Jason Morton. Algebraic geometry of matrix product states. *SIGMA*, 10(095):095, 2014.

[12] François Denis and Yann Esposito. On rational stochastic languages. *Fundamenta Informaticae*, 86(1, 2):41–77, 2008.

[13] Carlton Downey, Ahmed Hefny, Byron Boots, Geoffrey J Gordon, and Boyue Li. Predictive state recurrent neural networks. In *Proceedings of NIPS*, pages 6055–6066, 2017.

[14] Herbert Federer. *Geometric measure theory*. Springer, 2014.

[15] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[16] C Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[17] C Lee Giles, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chun Lee, and Dong Chen. Higher order recurrent networks and grammatical inference. In *Proceedings of NIPS*, pages 380–387, 1990.

[18] Hadrien Glaude and Olivier Pietquin. PAC learning of probabilistic automaton based on the method of moments. In *Proceedings of ICML*, pages 820–829, 2016.

[19] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of ICASSP*, pages 6645–6649. IEEE, 2013.

[20] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] Daniel J. Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. In *Proceedings of COLT*, 2009.

[23] Prateek Jain, Raghu Meka, and Inderjit S Dhillon. Guaranteed rank minimization via singular value projection. In *Proceedings of NIPS*, pages 937–945, 2010.

[24] Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. In *Proceedings of ICLR*, 2018.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Stefan Klus, Patrick Gelß, Sebastian Peitz, and Christof Schütte. Tensor-based dynamic mode decomposition. *Nonlinearity*, 31(7):3359, 2018.

[27] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[28] YC Lee, Gary Doolen, HH Chen, GZ Sun, Tom Maxwell, HY Lee, and C Lee Giles. Machine learning using a higher order correlation network. *Physica D: Nonlinear Phenomena*, 22(1-3):276–306, 1986.

[29] Tianyu Li, Guillaume Rabusseau, and Doina Precup. Nonlinear weighted finite automata. In *Proceedings of AISTATS*, pages 679–688, 2018.

[30] Qin Lin, Christian Hammerschmidt, Gaetano Pellegrino, and Sicco Verwer. Short-term time series forecasting with regression automata. 2016.

[31] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Proceedings of ICASSP*, pages 5528–5531. IEEE, 2011.

[32] Christian W Omlin and C Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM (JACM)*, 43(6):937–972, 1996.

[33] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[34] Hao Peng, Roy Schwartz, Sam Thomson, and Noah A Smith. Rational recurrences. In *Proceedings of EMNLP*, pages 1203–1214, 2018.

[35] Jordan B Pollack. The induction of dynamical recognizers. In *Connectionist Approaches to Language Learning*, pages 123–148. Springer, 1991.

[36] Guillaume Rabusseau. *A Tensor Perspective on Weighted Automata, Low-Rank Regression and Algebraic Mixtures*. PhD thesis, Aix-Marseille Université, 2016.

[37] Guillaume Rabusseau, Borja Balle, and Joelle Pineau. Multitask spectral learning of weighted automata. In *Proceedings of NIPS*, pages 2585–2594, 2017.

[38] Holger Rauhut, Reinhold Schneider, and Željka Stojanac. Low rank tensor recovery via iterative hard thresholding. *Linear Algebra and its Applications*, 523:220–262, 2017.

[39] Adria Recasens and Ariadna Quattoni. Spectral learning of sequence taggers over continuous sequences. In *Proceedings of ECML*, pages 289–304, 2013.

[40] Hanie Sedghi and Anima Anandkumar. Training input-output recurrent neural networks through spectral methods. *arXiv preprint arXiv:1603.00954*, 2016.

[41] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of COLT*, pages 440–449. ACM, 1992.

[42] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of ICML*, pages 1017–1024, 2011.

[43] Michael Thon and Herbert Jaeger. Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework. *Journal of Machine Learning Research*, 16:103–147, 2015.

[44] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Compressing recurrent neural network with tensor train. In *Proceedings of IJCNN*, pages 4451–4458. IEEE, 2017.

[45] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of ICML*, pages 5244–5253, 2018.

[46] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R Salakhutdinov. On multiplicative integration with recurrent neural networks. In *Proceedings of NIPS*, pages 2856–2864, 2016.

[47] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of ICML*, pages 3891–3900, 2017.

[48] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using tensor-train rnns. *arXiv preprint arXiv:1711.00073*, 2017.