Training Input-Output Recurrent Neural Networks through Spectral Methods

Hanie Sedghi* Anima Anandkumar†

June 3, 2016

Abstract

We consider the problem of training input-output recurrent neural networks (RNN) for sequence labeling tasks. We propose a novel spectral approach for learning the network parameters. It is based on decomposition of the cross-moment tensor between the output and a non-linear transformation of the input, based on score functions. We guarantee consistent learning with polynomial sample and computational complexity under transparent conditions such as non-degeneracy of model parameters, polynomial activations for the neurons, and a Markovian evolution of the input sequence. We also extend our results to Bidirectional RNN which uses both previous and future information to output the label at each time point, and is employed in many NLP tasks such as POS tagging.

Keywords: Recurrent neural networks, sequence labeling, spectral methods, score function.

1 Introduction

Learning with sequential data is widely encountered in domains such as natural language processing, genomics, speech recognition, video processing, financial time series analysis, and so on. Recurrent neural networks (RNN) are a flexible class of sequential models which can memorize past information, and selectively pass it on across sequence steps on multiple scales. However, training RNNs is challenging in practice, and backpropagation suffers from exploding and vanishing gradients as the length of the training sequence grows. To overcome this, either RNNs are trained over short sequences or incorporate more complex architectures such as long short-term memories (LSTM). For a detailed overview of RNNs, see [1]. Figure 1 contrasts the RNN with a feedforward neural network which has no memory.

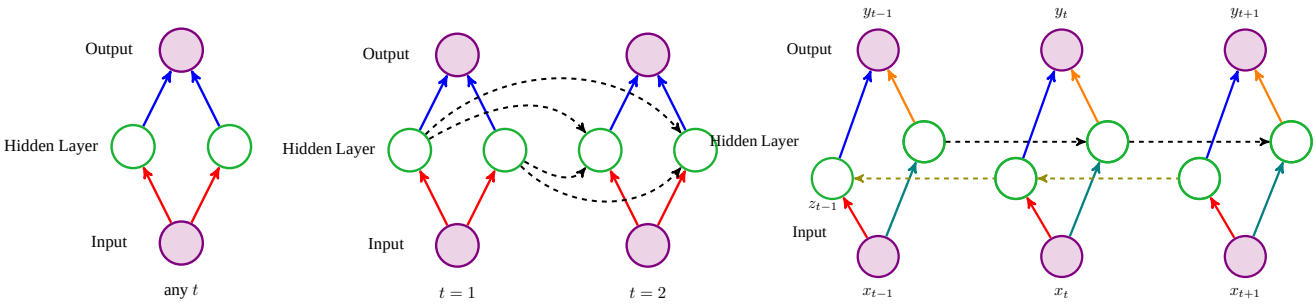
On the theoretical front, understanding of RNNs is at best rudimentary. With the current techniques, it is not tractable to analyze the highly non-linear state evolution in RNNs. Analysis of backpropagation is also intractable due to non-convexity of the loss function, and in general, reaching the global optimum is hard. Here, we take the first steps towards addressing these challenging issues.

We consider the class of input-output RNN or IO-RNN models, where each input in the sequence x_t has an output label y_t. These are useful for sequence labeling tasks, which has many applications such as parts of speech (POS) tagging and named-entity recognition (NER) in NLP [2], motif finding in protein analysis [3], action recognition in videos [4], and so on.

In addition, we also consider an extension of IO-RNN, viz., the bi-directional RNN or BRNN, first proposed by [5]. This includes two classes of hidden neurons: the first class receives recurrent connection from previous states, and the second class receives it from next steps. See Figure 1(c). BRNN is useful in NLP tasks such as POS tagging, where both previous and next words in a sentence have an effect on labeling the current word.

In this paper, we develop novel spectral methods for training IO-RNN and BRNN models. Spectral methods have previously been employed for unsupervised learning of a range of latent variable models such as hidden Markov models (HMM), topic models, network community models, and so on [6]. The idea is to decompose moment matrices

*Allen institute For Artificial Intelligence. Email: hanies@allenai.org
†University of California, Irvine. Email: a.anandkumar@uci.edu



(a) NN (b) IO-RNN (c) BRNN

Figure 1: Graphical representation of a Neural Network (NN) versus an Input-Output Recurrent Neural Network (IO-RNN) and a Bidirectional Recurrent Neural Network (BRNN)

and tensors using computationally efficient algorithms. The recovered components of the tensor decomposition yield consistent estimates of the model parameters. However, a direct application of these techniques is ruled out due to non-linearity of the activations in the RNN.

Recently, Janzamin et al. [7] derived a new framework for training input-output models in the supervised framework. It is based on spectral decomposition of moment tensors, obtained after certain non-linear transformations of the input. These non-linear transformations take the form of *score functions*, which depend on generative models of the input. This provides a new approach for transferring generative information, obtained through unsupervised learning, into discriminative training on labeled samples. Based on the aforementioned approach, Janzamin et al. [8] provided guaranteed risk bounds for training two-layer feedforward neural network models with polynomial sample and computational bounds. The conditions for obtaining the risk bounds are mild: a small approximation error for the target function under the given class of neural networks, a generative input model with a continuous distribution, and general sigmoidal activations at the neurons.

In this paper, we consider new spectral approaches for training IO-RNNs in both classification and regression settings. The previous score function approach for training feedforward networks (as described above) does not immediately extend, and there are some non-trivial challenges: (i) Non-linearity in a RNN is propagated along multiple steps in the sequence, while in the two-layer feedforward network, non-linearity is applied only once to the input. It is not immediately clear on how to "untangle" all these non-linearities and obtain guaranteed estimates of the network weights. (ii) Learning bidirectional RNNs is even more challenging since recursive non-linearities are applied in both the directions, and (iii) Assumption of i.i.d. input and output training samples is no longer applicable, and analyzing concentration bounds for samples generated from a RNN with non-linear state evolution is challenging. We address all these challenges concretely in this paper.

1.1 Summary of Results

Our main contributions are: (i) novel approaches for training input-output RNN and bidirectional RNN models using tensor decomposition methods, (ii) guaranteed recovery of network parameters with polynomial computational and sample complexity, and (iii) transparent conditions for successful recovery based on non-degeneracy of model parameters and bounded evolution of hidden states.

Score function transformations: Training input-output neural networks under arbitrary input is computationally hard. On the other hand, we show that training becomes tractable through spectral methods when the input is generated from a probabilistic model on a continuous state space. This paper can be considered as study of what it takes to uncover the nonlinear dynamics in the system. Since learning under arbitrary input is extremely challenging, we seek to discover under what functions/information of the input the problem becomes tractable. Although this differs from usual approach in training IO-RNNs, this is a promising first step towards demystifying these widely-used models. We show that with some knowledge of the input distribution, we can solve the extremely hard problem of training nonlinear IO-RNNs. We assume knowledge of score function forms, which correspond to normalized

derivatives of the input probability density function (p.d.f). For instance, if the input is standard Gaussian, score functions are given by the Hermite polynomials. There are many unsupervised approaches for estimating the score function efficiently, see Appendix E.1. To estimate the score function, one does not need to estimate the density, and this distinction is especially crucial for models where the normalizing constant or the *partition function* is intractable to compute. Guarantees have been derived for estimating score functions of many flexible model classes such as infinite dimensional exponential families [9]. In addition, in many settings, we have control over designing the input distribution and our method is directly applicable.

In this work, we assume a Markovian model for the input sequence $\{x_1, \ldots, x_n\}$ on a continuous state space, e.g., autoregressive process of order one or their kernel counterparts [10]. For a Markovian model, the score function only depends on the Markov kernel, and has a compact representation, as seen in Section 2.4.

Tensor decomposition: We form cross-moments between the output label and score functions of the input. For a vector input, the first order score function is a vector, second order is a matrix, and higher orders corresponds to tensors. Hence, the empirical moments are tensors, and we perform a CP tensor decomposition to obtain the rank-1 components. Efficient algorithms for tensor decomposition have been proposed before, based on simple iterative updates such as tensor power method [6]. After some simple manipulations on the components, we provide estimates of the network parameters of RNN models. The overall algorithm involves simple linear and multilinear steps, is embarrassingly parallel, and is practical to implement [11].

Recovery guarantees: We guarantee consistent recovery under (a low order) polynomial and computational complexity. We consider the realizable setting when samples are generated by a IO-RNN or a BRNN under the following transparent conditions: (i) one hidden layer of neurons with a polynomial activation function, (ii) Markovian input sequence, (iii) full rank weight matrices on input, hidden and output layers, and (iv) spectral norm bounds on the weight matrices to ensure bounded state evolution.

Currently, the question of approximation bounds by a RNN with a fixed number of neurons is not satisfactorily resolved [12] and it is valid to first consider the realizeable setting for this complex problem. The polynomial activations are a departure from the usual sigmoidal units, but they can also capture non-linear signal evolution, and have been employed in different applications, e.g., [13], [14]. The Markovian assumption on the input limits the extent of dependence and allows us to derive concentration bounds for our empirical moments. The full rank conditions on the weight matrices imply non-degeneracy in the neural representation: the weights for any two neurons cannot linearly combine to generate the weight of another neuron. Such conditions have previously been derived for spectral learning of HMMs and other latent variable models [6]. Moreover, it can be easily relaxed by considering higher order moment tensors, and is relevant when we want to have more neurons than the input dimension in our network. The rank assumption on the output weight matrix implies a vector output of sufficient dimensions, i.e., sufficient number of output classes. This can be relaxed to a scalar output, the details are given in Appendix E.2.

The spectral norm condition on the weight matrices arises in the analysis of concentration bounds for the empirical moments. Since we assume polynomial state evolution, it is important to ensure bounded values of the hidden states, and this entails a bound on the spectral norm of the weight matrices. We employ concentration bounds for functions of Markovian input from [15, 16] and combine it with matrix Azuma's inequality [17] to obtain concentration of empirical moment tensors. This implies learning RNNs with polynomial sample complexity.

Related work: The following works are directly relevant to this paper. **(a) Spectral approaches for sequence learning:** Previous guaranteed approaches for sequence learning mostly focus on the class of hidden Markov models (HMM). Anandkumar et al. [6] provide a tensor decomposition method for learning the parameters under non-degeneracy conditions, similar to ours. This framework is extended to more general HMMs in [18]. While in a HMM, the relationship between the hidden and observed variables can be modeled as a linear one, in a RNN it is non-linear. However, in a IO-RNN, we have both inputs and outputs, and that is helpful in handling the non-linearities. **(b) Input-output sequence models:** A rich set of models based on RNNs have been employed in practice in a wide range of applications. Lipton et al. [1] provides a nice overview of these various models. Balduzzi and Ghifari [19] recently apply physics based principles to design RNNs for stabilizing gradients and getting better training error. However, a rigorous analysis of these techniques is lacking.

2 Preliminaries

Let $[n] := \{1, 2, \ldots, n\}$, and $\langle u, v \rangle$ denote the inner product of vectors u and v. For sequence of n vectors $z_1, \ldots, z_n$, we use the notation $z[n]$ to denote the whole sequence. For vector v, v^{*m} refers to elementwise m^{th} power of v. For matrix $C \in \mathbb{R}^{d\times k}$, the j-th column is referred by C_j or c_j, $j \in [k]$, the j^{th} row is referred by $C^{(j)}$ or $c^{(j)}$, $j \in [d]$. Throughout this paper, $\nabla_x^{(m)}$ denotes the m^{th} order derivative operator w.r.t. variable x.

Tensor: A real m^{th} *order tensor* $T \in \bigotimes^m \mathbb{R}^d$ is a member of the outer product of Euclidean spaces $\mathbb{R}^d$. The different dimensions of the tensor are referred to as *modes*.

Tensor Reshaping: $T_2 = \text{Reshape}(T_1, v_1, \ldots, v_l)$ means that T_2 is a tensor of order l that is made by reshaping tensor T_1 such that the first mode of T_2 includes modes of T_1 that are shown in v_1, the second mode of T_2 includes modes of T_1 that are shown in v_2 and so on. For example if T_1 is a tensor of order 5, $T_2 = \text{Reshape}(T_1, [1\ 2], 3, [4\ 5])$ is a third order tensor, where its first mode is made by concatenation of modes $1, 2$ of T_1 and so on.

Tensor rank: A 3rd order tensor $T \in \mathbb{R}^{d\times d\times d}$ is said to be rank-1 if it can be written in the form $T = w \cdot a \otimes b \otimes c \Leftrightarrow T(i, j, l) = w \cdot a(i) \cdot b(j) \cdot c(l)$, where $\otimes$ represents the *outer product*, and $a, b, c \in \mathbb{R}^d$ are unit vectors. A tensor $T \in \mathbb{R}^{d\times d\times d}$ is said to have a CP (Candecomp/Parafac) *rank* k if it can be (minimally) written as the sum of k rank-1 tensors

$T = \sum_{i\in[k]} w_i a_i \otimes b_i \otimes c_i, \quad w_i \in \mathbb{R},\ a_i, b_i, c_i \in \mathbb{R}^d$. Note that $v^{\otimes p} = v \otimes v \otimes v \cdots \otimes v$, where v is repeated p times.

Definition 1 (Row-wise Kronecker product) *For matrices $A, B \in \mathbb{R}^{d\times k}$, the* Row-wise Kronecker product $\in \mathbb{R}^{d\times k^2}$ *is defined as follows.*

Let $a^{(i)}, b^{(i)}$ be rows of A, B respectively. Rows of $A \odot B$ are of the form $a^{(i)} \otimes b^{(i)}$. Note that our definition is different from usual definition of Khatri-Rao product which is a column-wise Kronecker product.

Derivative: For function $g(x) : \mathbb{R}^d \to \mathbb{R}$ with vector input $x \in \mathbb{R}^d$, the m-th order derivative w.r.t. variable x is denoted by $\nabla_x^{(m)} g(x) \in \bigotimes^m \mathbb{R}^d$ (which is a m-th order tensor) such that

$$\left[\nabla_x^{(m)} g(x)\right]_{i_1,\ldots,i_m} := \frac{\partial g(x)}{\partial x_{i_1} \partial x_{i_2} \cdots \partial x_{i_m}}, \quad i_1, \ldots, i_m \in [d].$$

Tensor as multilinear form: We view a tensor $T \in \mathbb{R}^{d\times d\times d}$ as a multilinear form. Consider matrices $M_l \in \mathbb{R}^{d\times d_l}, l \in \{1, 2, 3\}$. Then tensor $T(M_1, M_2, M_3) \in \mathbb{R}^{d_1} \otimes \mathbb{R}^{d_2} \otimes \mathbb{R}^{d_3}$ is defined as

$$T(M_1, M_2, M_3)_{i_1,i_2,i_3} := \sum_{j_1,j_2,j_3\in[d]} T_{j_1,j_2,j_3} \cdot M_1(j_1, i_1) \cdot M_2(j_2, i_2) \cdot M_3(j_3, i_3).$$

In particular, for vectors $u, v, w \in \mathbb{R}^d$, we have [1]

$$T(I, v, w) = \sum_{j,l\in[d]} v_j w_l T(:, j, l) \ \in \mathbb{R}^d,$$

which is a multilinear combination of the tensor mode-1 fibers. Similarly $T(u, v, w) \in \mathbb{R}$ is a multilinear combination of the tensor entries, and $T(I, I, w) \in \mathbb{R}^{d\times d}$ is a linear combination of the tensor slices.

2.1 Problem Formulation

We consider a two-layer input-output RNN, that includes both regression and classification settings:

$$\mathbb{E}[y_t|h_t] = A_2^\top h_t, \quad h_t = \text{poly}_l(A_1 x_t + U h_{t-1}),$$

where $\text{poly}_l(\cdot)$ denotes an element-wise polynomial of order l, The input sequence x consists of the vectors $x_t \in \mathbb{R}^{d_x}$.$h_t \in \mathbb{R}^{d_h}$, $y_t \in \mathbb{R}^{d_y}$ and hence $A_1 \in \mathbb{R}^{d_h\times d_x}, U \in \mathbb{R}^{d_h\times d_h}$ and $A_2 \in \mathbb{R}^{d_h\times d_y}$. We can learn the parameters of

[1] Compare with the matrix case where for $M \in \mathbb{R}^{d\times d}$, we have $M(I, u) = Mu := \sum_{j\in[d]} u_j M(:, j) \in \mathbb{R}^d$.

the model using our method. Our algorithm is called GLOREE (Guaranteed Learning Of Recurrent nEural nEtworks) and is shown in Algorithm 1.

Throughout the paper, we assume that the p.d.f. of the input sequence vanishes in the boundary (i.e., when any coordinate of the input goes to infinity). This is also the assumption in [7]. We consider the case where input is a geometrically ergodic Markov chain. Then in order to have mixing and assure ergodicity for the output, we need to impose additional constraints on the model.

2.2 Review of Score functions

As mentioned in the introduction, our method builds on the method introduced by Janzamin et al. [7] called FEAST (Feature ExtrAction using Score function Tensors). The goal of FEAST is to extract discriminative directions using the cross-moment between the label and score function of input. Score function is the normalized (higher order) derivative of p.d.f. of the input.

Let $p(x)$ denote the joint probability density function of random vector $x \in \mathbb{R}^d$. Janzamin et al. [7] denote $\mathcal{S}_m(x)$ as the m^{th} order score function, given by

$$\mathcal{S}_m(x) = (-1)^m \frac{\nabla_x^{(m)} p(x)}{p(x)}, \tag{1}$$

where $\nabla_x^{(m)}$ denotes the m^{th} order derivative operator w.r.t. variable x. It can also be derived using the recursive form

$$\mathcal{S}_1(x) = -\nabla_x \log p(x), \quad \mathcal{S}_m(x) = -\mathcal{S}_{m-1}(x) \otimes \nabla_x \log p(x) - \nabla_x \mathcal{S}_{m-1}(x). \tag{2}$$

The importance of score function is that it provides a derivative operator. Janzamin et al. [7] proved that the cross-moment between the label and the score function of the input yields the information regarding derivative of the label w.r.t. the input.

Theorem 1 (Yielding differential operators [7]) *Let $x \in \mathbb{R}^{d_x}$ be a random vector with joint density function $p(x)$. Suppose the m^{th} order score function $\mathcal{S}_m(x)$ defined in* (1) *exists. Consider any continuously differentiable tensor function $G(x) : \mathbb{R}^{d_x} \to \bigotimes^r \mathbb{R}^{d_y}$. Then, under some mild regularity conditions, we have*

$$\mathbb{E}\left[G(x) \otimes \mathcal{S}_m(x)\right] = \mathbb{E}\left[\nabla_x^{(m)} G(x)\right].$$

2.3 Score function form for RNN

We now extend the notion of score function to handle sequence data with non i.i.d. samples. We denote the score function at each time step t in the sequence as $\mathcal{S}_m(z[n], t)$, where $z[n] := z_1, z_2, \ldots, z_n$, and it is defined below. Theorem 1 can be readily modified to

Lemma 2 (Score function form for input sequence) *For vector sequence $z[n] = \{z_1, z_2, \ldots, z_n\}$, let $p(z_1, z_2, \cdots, z_n)$ and $\mathcal{S}_m(z[n], [n])$ respectively denote the joint density function and the corresponding m^{th} order score function. Then, under some mild regularity conditions, for all continuously differentiable functions $G(z_1, z_2, \ldots, z_n)$, we have*

$$\mathbb{E}\left[G(z_1, z_2, \ldots, z_n) \otimes \mathcal{S}_m(z[n], t)\right] = \mathbb{E}\left[\nabla_{z_t}^{(m)} G(z_1, z_2, \ldots, z_n)\right],$$

where $\nabla_{z_t}^{(m)}$ denotes the m^{th} order derivative operator w.r.t. z_t,

$$\mathcal{S}_m(z[n], t) = (-1)^m \frac{\nabla_{z_t} p(z_1, z_2, \ldots, z_s)}{p(z_1, z_2, \ldots, z_s)}. \tag{3}$$

2.4 Score function form for Markov chains

We assume a Markovian model for the input sequence, and derive compact score function forms for (3). Note that this form can be readily expanded to higher-order Markov chains.

Lemma 3 (Score function form for first-order Markov Chains) *Let the input sequence $\{x_i\}_{i\in[n]}$ be a first-order Markov chain. The score function in* (3) *simplifies as*

$$\mathcal{S}_m(x[n],i) = (-1)^m \frac{\nabla_{x_i}^m \left[p(x_{i+1}|x_i)p(x_i|x_{i-1})\right]}{p(x_{i+1}|x_i)p(x_i|x_{i-1})}. \tag{4}$$

The proof follows the definition of first-order Markov chain and Equation (3).

Score function form for AR processes The Gaussian AR-1 process is a special case of the Markov chain: we have $x_t = \phi x_{t-1} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0,1), \ \ |\phi| < 1$, where index t represents the time and $|\phi| < 1$. For simplicity, we have considered scalar x, the form can be easily extended to vector x.

The conditional distribution is given by $x_t|x_1,\cdots,x_{t-1} \sim \mathcal{N}(\phi x_{t-1}, 1), t = 2,\cdots,n$, and [20]

$$p(x[n]) = \frac{1}{(2\phi)^{n/2}}|J|^{1/2}\exp\left(-\frac{1}{2}x[n]^\top Jx[n]\right), \quad J_{ij} = \begin{cases} -\phi & i = j+1, i = j-1 \\ 1+\phi^2 & i = j \neq 1 \\ 1 & i = j = 1 \\ 0 & o.w. \end{cases}$$

Expanding Equation (2) for any Gaussian AR process, we have that

$$\mathcal{S}_1(x[n],[n]) = x[n]^\top J, \quad \mathcal{S}_m(x[n],[n]) = \mathcal{S}_{m-1}(x[n],[n]) \otimes (x[n]^\top J) - \nabla_{x[n]}\mathcal{S}_{m-1}(x[n],[n]).$$

$$\text{i.e.,} \qquad \mathcal{S}_1(x[n],[n])_i = \begin{cases} x_1 - \phi x_2 & i = 1 \\ -\phi x_{i-1} + (1+\phi^2)x_i - \phi x_{i+1} & i \neq 1, n \\ -\phi x_{n-1} + x_n & i = n \end{cases} \tag{5}$$

3 Algorithm and Guarantees

In this paper, we have functions which map input sequence $x_1,\ldots x_n$ to an output sequence $y_1,\ldots,y_n$. By assuming a structured form of function mapping in terms of IO-RNN, we can hope to recover the function parameters efficiently. We exploit the score function forms derived above to compute partial derivatives of the output sequence.We first start with some simple intuitions.

3.1 Preliminary insights

Generalized linear model: Before considering the RNN, consider a simple generalized linear model with i.i.d. samples: $\mathbb{E}[y|x] = A_2^\top \sigma(A_1 x)$, where A_1 is the weight matrix and $\sigma(\cdot)$ is element-wise activation. Here, the partial derivative of $\mathbb{E}[y|x]$ w.r.t. x has a linear relationship with the weight matrices A_1 and A_2, i.e.,

$$\nabla_x \mathbb{E}[y|x] = \mathbb{E}[\nabla_x \sigma(A_1 x)] = A_2^\top \mathbb{E}[\sigma'(A_1 x)]A_1. \tag{6a}$$

$$\mathbb{E}[y \otimes \mathcal{S}_2(x)] = \nabla_x^2 \mathbb{E}[y|x] = \sum_{i\in d_h} \mu_i A_2^{(i)} \otimes A_1^{(i)} \otimes A_1^{(i)}. \tag{6b}$$

The first partial derivative is obtained by forming the cross moment $\mathbb{E}[y \otimes \mathcal{S}_1(x)]$, as given by Theorem 1. The form in (6a) yields A_1 and A_2 up to a linear transformation. But, by computing second order derivatives, we can recover A_1 and A_2, up to scaling of their rows. The second order derivative has the form in (6b). The tensor decomposition in [6] uniquely recovers A_1, A_2 up to scaling of rows, under full row rank assumptions.

Recovering input-output weight matrices in IO-RNN: The above intuition for GLM readily carries over to IO-RNN. Recall that the IO-RNN has the form

$$\mathbb{E}[y_t|h_t] = A_2^\top h_t, \quad h_t = \mathrm{poly}_l(A_1 x_t + U h_{t-1}),$$

Algorithm 1 GLOREE (Guaranteed Learning Of Recurrent nEural nEtworks) for vector input and quadratic function (General case shown in Algorithm 3 in Appendix C.1).

input Labeled samples $\{(x_i, y_i) : i \in [n]\}$ from IO-RNN model in Figure 1.

1: Compute 2nd-order score function $\mathcal{S}_2(x[n], i)$ of the input sequence as in Equation (4).
2: Compute $\widehat{T} := \widehat{\mathbb{E}}\left[y_i \otimes \mathcal{S}_2(x[n], i)\right]$. The empirical average is over a single sequence.
3: $\{\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\}$ = tensor decomposition($\widehat{T}$); see Appendix F.
4: $\hat{A}_2 = \hat{R}_1, \hat{R}_1 = \left(\hat{R}_2 + \hat{R}_3\right)/2$.
5: Compute 4th-order score function $\mathcal{S}_4(x[n], i)$ of the input sequence as in Equation (4).
6: Compute $\widehat{T}_2 = \hat{\mathbb{E}}\left[y_t \otimes \text{Reshape}\left((\mathcal{S}_4(x[n], t-1), [1\ \ 2], [3\ \ 4]\right)\right]$.
7: $\{\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\}$ = tensor decomposition($\widehat{T}_2$); see Appendix F.
8: $\hat{U} = \tilde{R}\left[\hat{A}_1 \odot \hat{A}_1\right]^\dagger$, $\tilde{R} = \left(\hat{R}_2 + \hat{R}_3\right)/2$. $\odot$ is row-wise Kronecker product as in Definition 1.
9: **return** $\hat{A}_1, \hat{A}_2, \hat{U}$.

where poly_l denotes any polynomial function of degree at most l.

Suppose we have access to partial derivatives $\nabla_{x_t}\mathbb{E}[y_t|h_t]$ and $\nabla^2_{x_t}\mathbb{E}[y_t|h_t]$, then they have the same forms as (6a) and (6b). This is because h_t does not depend on x_{t-1} given x_t, h_{t-1}. Thus, the weight matrices A_1 and A_2 can be easily recovered by forming $\mathbb{E}[y_t \otimes \mathcal{S}_2(x[n], t)]$, as given by (3), and it has a compact form for Markovian input in (4). Note that this intuition holds for any non-linear element-wise activation function, and we do not require it to be a polynomial at this stage.

Recovering hidden state transition matrix in IO-RNN: Recovering the transition matrix U is much more challenging since we do not have access to hidden state sequence $h[n]$. Thus, we cannot readily form partial derivatives of the form $\nabla^m_{h_{t-1}}\mathbb{E}[y_t|h_t]$. Moreover, the non-linearities get recursively propagated along the chain. Here, we provide an algorithm that works for any polynomial activation function of fixed degree l.

The main idea is that we attempt to recover U by considering partial derivatives $\nabla^m_{x_{t-1}}\mathbb{E}[y_t|h_t]$, i.e., how the previous input x_{t-1} affects current output y_t. At first glance, this appears complicated and indeed, the various terms are highly coupled and we do not have a nice CP tensor decomposition form as before. However, we can prove that when the derivative order m is sufficiently large, a nice CP tensor form emerges out, and this m depends on the degree l of the polynomial activation.

For simplicity, we provide intuitions for the quadratic activation function ($l = 2$). Now, y_t is a degree-4 polynomial function of x_{t-1}, since the activation function is applied twice. By considering fourth order derivative $\nabla^4_{x_{t-1}}\mathbb{E}[y_t|h_t]$, many coupled terms vanish since they correspond to polynomial functions of degree less than 4. The surviving term has a nice CP tensor form, and we can recover U efficiently from it. Note that this fourth order partial derivative can be computed efficiently using fourth order score function and forming the cross-moment $\mathbb{E}[y_t \otimes \mathcal{S}_4(x[n], t-1)]$. The precise algorithm is given in Algorithm 1.

3.2 Training IO-RNNs

We now provide our algorithm for training IO-RNNs. In this paper we consider vector output and polynomial activation functions of any order $l \geq 2$. For simplicity of notation, we first discuss the case for quadratic activation function. Our algorithm is called GLOREE (Guaranteed Learning Of Recurrent nEural nEtworks) and is shown in Algorithm 1 for quadratic activation function. The general algorithm and analysis for $l \geq 2$ is shown in Appendix C.1. For completeness, we handle the linear case in Appendix E.3. We also cover the case where output y is a scalar (e.g. a binary label) in Appendix E.2.

Now, we consider an RNN with quadratic activation function and vector output. Let

$$\mathbb{E}[y_t|h_t] = A_2^\top h_t, \quad h_t = \text{poly}_2(A_1 x_t + U h_{t-1}),$$

where $x_t \in \mathbb{R}^{d_x}, h_t \in \mathbb{R}^{d_h}, y_t \in \mathbb{R}^{d_y}$ and $A_1 \in \mathbb{R}^{d_h \times d_x}, U \in \mathbb{R}^{d_h \times d_h}, A_2 \in \mathbb{R}^{d_h \times d_y}$. We can learn the parameters of the model using GLOREE. Let n be the window size for RNN.

Theorem 4 (Learning parameters of RNN for quadratic activation function) *We have the following properties for an IO-RNN:*

$$\mathbb{E}\left[y_t \otimes \mathcal{S}_2(x[n], t)\right] = 2 \sum_{i \in d_h} A_2^{(i)} \otimes A_1^{(i)} \otimes A_1^{(i)}. \tag{7}$$

Hence, we can recover A_1, A_2 via tensor decomposition, assuming that they are full row rank.

In order to learn U we form the tensor $\mathbb{E}\left[y_t \otimes \mathcal{S}_4(x_{t-1})\right]$, and under quadratic activations, we have

$$\mathbb{E}\left[y_t \otimes \textit{Reshape}(\mathcal{S}_4(x[n], t-1), [1\ 2], [3\ 4])\right] = \sum_{i \in d_h} A_2^{(i)} \otimes [U(A_1 \odot A_1)]^{(i)} \otimes [U(A_1 \odot A_1)]^{(i)}.$$

Hence, we can recover $U(A_1 \odot A_1)$ via tensor decomposition. Since A_1 is previously recovered using (7)*, and $(A_1 \odot A_1)^\dagger$ exists due to full rank assumption, we can recover U. Thus, Algorithm 1 (GLOREE) consistently recovers the parameters of IO-RNN under quadratic activations.*

For proof, see Appendix B.1.

3.3 Training Bidirectional RNNs

Bidirectional Recurrent Neural network was first proposed by Schuster and Paliwal [5]. Here there are two groups of hidden neurons.The first group receives recurrent connections from previous time steps while the other from the next time steps. The following equations describe a BRNN

$$\mathbb{E}[y_t | h_t, z_t] = A_2^\top \begin{bmatrix} h_t \\ z_t \end{bmatrix}, \quad h_t = f(A_1 x_t + U h_{t-1}), \quad z_t = g(B_1 x_t + V z_{t+1}), \tag{8}$$

where h_t and z_t denote the neurons that receive forward and backward connections, respectively. Note that BRNNs cannot be used in online settings as they require knowledge of the future steps. However, in various natural language processing applications such as part of speech (POS) BRNNs are effective models since they consider both past and future words in a sentence.

We can learn the parameters of bidirectional RNN by modifying our earlier algorithm. For notation simplicity, Algorithm 2 shows the case for quadratic activation functions $f(\cdot)$ and $g(\cdot)$. The general polynomial function is considered in Algorithm 4 in Appendix C.2.

Let us provide some intuitions. If we only had forward or backward connections, we would directly apply our previous method in GLOREE. For backward connections, the only difference would be to use derivatives of $\mathbb{E}[y_t|h_t, z_t]$ w.r.t. x_{t+1} to learn the transition matrix V. Now that we have both hidden neurons mixing to yield the output vector y_t, the cross-moment tensor $T = \mathbb{E}[y_i \otimes \mathcal{S}_2(x[n], i)]$ has a CP decomposition where the factor matrix for the first mode is A_2, i.e., the tensor has a specific form: $T = A_2^\top \begin{bmatrix} T_h \\ T_z \end{bmatrix}$, where T_h corresponds to the tensor incorporating columns of A_1 and T_z incorporates columns of B_1. Hence, under full rank assumption, as before, we can recover A_2. Next, we can invert A_2 to recover T_h and T_z. We decompose them to recover A_1 and B_1. The steps for recovering U and V remains the same as before in GLOREE. The only difference is to use derivatives of $\mathbb{E}[y_t|h_t, z_t]$ w.r.t. x_{t+1} to learn V.

Theorem 5 (Training BRNN) *Consider the BRNN model as in* (8)*. Algorithm 2 correctly recovers parameters of this model.* For proof, see Appendix B.2.

3.4 Sample and Computational Complexity Analysis

Sample Complexity: In order to analyze the sample complexity, we first need to prove concentration bound for the cross-moment tensor, then we use analysis of the tensor decomposition to show the that sample complexity is a polynomial of $(d_x, d_y, d_h, G, \frac{1}{1-\theta}, \sigma_{\min}^{-1}(A_1), \sigma_{\min}^{-1}(A_2), \sigma_{\min}^{-1}(U))$. Deriving concentration bound for the cross-moment tensor is a bit more involved since we have a non-i.i.d. sequence. Moreover, since we assume a polynomial

Algorithm 2 GLOREE-B (Guaranteed Learning Of Recurrent nEural nEtworks-Bidirection case) for quadratic activation function (general case is shown in Algorithm 4).

input Labeled samples $\{(x_i, y_i) : i \in [n]\}$ from IO-RNN model in Figure 1(b).

input 2nd-order score function $\mathcal{S}_2(x[n], [n])$ of the input x; see Equation (1) for the definition.

1: Compute $\widehat{T} := \hat{\mathbb{E}}[y_i \otimes \mathcal{S}_2(x[n], i)]$. The empirical average is over a single sequence.

2: $\{(\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\widehat{T})$; see Appendix F.

3: $\hat{A}_2 = \hat{R}_1$.

4: Compute $\tilde{T} = \widehat{T}(((\hat{A}_2)^\top)^{-1}, I, I)$. For definition of multilinear form see Section 2.

5: $\{(\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\tilde{T})$;

6: $\hat{C} = (\hat{R}_2 + \hat{R}_3)/2$.

7: $\hat{C} = \begin{bmatrix} \hat{A}_1 \\ \hat{B}_1 \end{bmatrix}$.

8: Compute 4th-order score function $\mathcal{S}_4(x[n], t-1)$ of the input sequence as in Equation (4).

9: Compute $\widehat{T} = \hat{\mathbb{E}}\left[y_t \otimes \text{Reshape}\left((\mathcal{S}_4(x[n], t-1), [1\ 2], [3\ 4]\right)\right]$.

10: $\{\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\widehat{T})$; see Appendix F.

11: $\hat{U} = \tilde{R}\left[\hat{A}_1 \odot \hat{A}_1\right]^\dagger$, $\tilde{R} = \left(\hat{R}_2 + \hat{R}_3\right)/2$. $\odot$ is row-wise Kronecker product as in Definition 1.

12: Repeat lines (8)-(11) with $\mathcal{S}_4(x[n], t+1)$ instead of $\mathcal{S}_4(x[n], t-1)$ to recover $\hat{V}$.

13: **return** $\hat{A}_1, \hat{A}_2, \hat{B}_1, \hat{U}, \hat{V}$.

activation function, the hidden state h_t can grow in an unbounded manner. To avoid this, we impose additional assumptions on spectral norm of the model matrices and without loss of generality assume that ℓ_2 norm of each entry in the input sequence is bounded by 1. For detailed analysis, see Appendix D.

Computational Complexity: The computational complexity of our method is related to the complexity of the tensor decomposition methods. See [6, 8] for a detailed discussion. It is popular to perform the tensor decomposition in a stochastic manner by splitting the data into mini-batches and reducing computational complexity. Starting with the first mini-batch, we perform a small number of tensor power iterations, and then use the result as initialization for the next mini-batch, and so on. We assume that score function is given to us in an efficient form. Note that if we can write the cross-moment tensor in terms of summation of rank-1 components, we do not need to form the whole tensor explicitly. As an example, if input follows Gaussian distribution, the score function has a simple polynomial form, and the computational complexity of tensor decomposition is $O(nd_h d_x R)$, where n is the number of samples and R is the number of initializations for the tensor decomposition. Similar argument follows when the input is mixture of Gaussian distributions.

4 Conclusion

This work is a first step towards answering challenging questions in sequence modeling. Many of the assumptions can be relaxed, e.g., here we assumed IO-RNNs with aligned inputs and outputs which is not applicable to tasks such as machine translation, and we can relax this assumption to obtain more general RNNs. We have assumed polynomial activation functions at the neurons and our computational and sample complexity degrade exponentially in the degree of the polynomial. It is an open question to develop strategies to avoid this exponential blowup, and to extend it to the usual sigmoidal units in RNNs. Architectures such as long short-term memory (LSTM) have much more complicated non-linear dependencies, and it is unclear on how to analyze them effectively. Extending this framework to HMMs and general settings is subject of future research. Analysis under non-stationary inputs is another challenging open problem.

We have assumed the realizable setting where samples are generated from a RNN. For general sequences, it is unclear on how to analyze the approximation bounds. While a solid theoretical framework exists for function approximation in the i.i.d. setting [21], the question of approximation bounds by a RNN with a fixed number of neurons is not satisfactorily resolved [12].

Acknowledgment

The authors thank Majid Janzamin for detailed discussions on sample complexity and constructive comments on the draft. This work was done during the time H. Sedghi was a visiting researcher at University of California, Irvine and was supported by NSF Career award FG15890. A. Anandkumar is supported in part by Microsoft Faculty Fellowship, NSF Career award CCF-1254106, ONR award N00014-14-1-0665, ARO YIP award W911NF-13-1-0084, and AFOSR YIP award FA9550-15-1-0221.

References

[1] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[2] C. Manning and H. Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[3] Asa Ben-Hur and Douglas Brutlag. Sequence motifs: highly predictive features of protein function. In *Feature Extraction*, pages 625–645. Springer, 2006.

[4] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[5] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

[6] Anima Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15:2773–2832, 2014.

[7] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Score Function Features for Discriminative Learning: Matrix and Tensor Frameworks. *arXiv preprint arXiv:1412.2863*, Dec. 2014.

[8] M. Janzamin, H. Sedghi, and A. Anandkumar. Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods. *Preprint available on arXiv:1506.08473*, June 2015.

[9] Bharath Sriperumbudur, Kenji Fukumizu, Revant Kumar, Arthur Gretton, and Aapo Hyvärinen. Density estimation in infinite dimensional exponential families. *arXiv preprint arXiv:1312.3516*, 2013.

[10] Maya Kallas, Paul Honeine, Cédric Richard, Clovis Francis, and Hassan Amoud. Kernel-based autoregressive modeling with a pre-image technique. In *Statistical Signal Processing Workshop (SSP), 2011 IEEE*, pages 281–284. IEEE, 2011.

[11] Yining Wang, Hsiao-Yu Tung, Alexander Smola, and Anima Anandkumar. Fast and guaranteed tensor decomposition via sketching. In *Proc. of NIPS*, 2015.

[12] Barbara Hammer. On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1):107–123, 2000.

[13] D. Chen and C. Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[14] Qizhe Xie, Kai Sun, Su Zhu, Lu Chen, and Kai Yu. Recurrent polynomial network for dialogue state tracking with mismatched semantic parsers. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 295, 2015.

[15] Leonid Aryeh Kontorovich, Kavita Ramanan, et al. Concentration inequalities for dependent random variables via the martingale method. *The Annals of Probability*, 36(6):2126–2158, 2008.

[16] Aryeh Kontorovich and Roi Weiss. Uniform chernoff and dvoretzky-kiefer-wolfowitz-type inequalities for markov chains and related processes. *Journal of Applied Probability*, 51(4):1100–1113, 2014.

[17] J. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012.

[18] Qingqing Huang, Rong Ge, Sham Kakade, and Munther Dahleh. Minimal realization problem for hidden markov models. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pages 4–11. IEEE, 2014.

[19] David Balduzzi and Muhammad Ghifari. Strongly-typed recurrent neural networks. *arXiv preprint arXiv:1602.02218*, 2016.

[20] H. Rue and L. Held. *Gaussian Markov random fields: theory and applications*. CRC Press, 2005.

[21] Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14: 115–133, 1994.

[22] George Konidaris and Finale Doshi-Velez. Hidden parameter markov decision processes: An emerging paradigm for modeling families of related tasks. In *2014 AAAI Fall Symposium Series*, 2014.

[23] Kamyar Azizzadenesheli, Alessandro Lazaric, and Anima Anandkumar. Reinforcement learning of pomdp's using spectral methods. *arXiv preprint arXiv:1602.07764*, 2016.

[24] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. In *Journal of Machine Learning Research*, pages 695–709, 2005.

[25] Kevin Swersky, David Buchman, Nando D Freitas, Benjamin M Marlin, et al. On autoencoders and score matching for energy based models. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1201–1208, 2011.

[26] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution. *arXiv preprint arXiv:1211.4246*, 2012.

[27] Hanie Sedghi and Anima Anandkumar. Provable methods for training neural networks with sparse connectivity. *NIPS workshop on Deep Learning and Representation Learning*, Dec. 2014.

[28] D. Spielman, H. Wang, and J. Wright. Exact recovery of sparsely-used dictionaries. In *Conference on Learning Theory*, 2012.

[29] L. Song, A. Anandkumar, B. Dai, and B. Xie. Nonparametric estimation of multi-view latent variable models. *arXiv preprint arXiv:1311.3287*, Nov. 2013.

[30] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *J. of Machine Learning Research*, 15:2773–2832, 2014.

[31] Anima Anandkumar, Rong Ge, and Majid Janzamin. Sample Complexity Analysis for Learning Overcomplete Latent Variable Models through Tensor Methods. *arXiv preprint arXiv:1408.0553*, Aug. 2014.

[32] Anima Anandkumar, Rong Ge, and Majid Janzamin. Guaranteed Non-Orthogonal Tensor Decomposition via Alternating Rank-1 Updates. *arXiv preprint arXiv:1402.5180*, Feb. 2014.

[33] Anima Anandkumar, Yi-kai Liu, Daniel J Hsu, Dean P Foster, and Sham M Kakade. A spectral algorithm for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 917–925, 2012.

A Notation

For completeness, all the notation required in the paper and Appendices are gathered here as well.

Let $[n] := \{1, 2, \ldots, n\}$, and $\|u\|$ denote the ℓ_2 or Euclidean norm of vector u, and $\langle u, v\rangle$ denote the inner product of vectors u and v. For sequence of n vectors $z_1, \ldots, z_n$, we use the notation $z[n]$ to denote the whole sequence. For vector v, v^{*m} refers to elementwise m^{th} power of v. For matrix $C \in \mathbb{R}^{d\times k}$, the j-th column is referred by C_j or c_j, $j \in [k]$, the j^{th} row is referred by $C^{(j)}$ or $c^{(j)}$, $j \in [d]$ and $\|C\|$ denotes the spectral norm of matrix C. Throughout this paper, $\nabla_x^{(m)}$ denotes the m^{th} order derivative operator w.r.t. variable x.

Tensor: A real m^{th} *order tensor* $T \in \bigotimes^m \mathbb{R}^d$ is a member of the outer product of Euclidean spaces $\mathbb{R}^d$. The different dimensions of the tensor are referred to as *modes*. For instance, for a matrix, the first mode refers to columns and the second mode refers to rows.

Tensor matricization: For a third order tensor $T \in \mathbb{R}^{d\times d\times d}$, the matricized version along first mode denoted by $M \in \mathbb{R}^{d\times d^2}$ is defined such that

$$T(i, j, l) = M(i, l + (j - 1)d), \quad i, j, l \in [d]. \tag{9}$$

Tensor Reshaping: $T_2 = \text{Reshape}(T_1, v_1, \ldots, v_l)$ means that T_2 is a tensor of order l that is made by reshaping tensor T_1 such that the first mode of T_2 includes modes of T_1 that are shown in v_1, the second mode of T_2 includes modes of T_1 that are shown in v_2 and so on. For example if T_1 is a tensor of order 5, $T_2 = \text{Reshape}(T_1, [1\ 2], 3, [4\ 5])$ is a third order tensor, where its first mode is made by concatenation of modes $1, 2$ of T_1 and so on.

Tensor rank: A 3rd order tensor $T \in \mathbb{R}^{d\times d\times d}$ is said to be rank-1 if it can be written in the form

$$T = w \cdot a \otimes b \otimes c \Leftrightarrow T(i, j, l) = w \cdot a(i) \cdot b(j) \cdot c(l), \tag{10}$$

where $\otimes$ represents the *outer product*, and $a, b, c \in \mathbb{R}^d$ are unit vectors. A tensor $T \in \mathbb{R}^{d\times d\times d}$ is said to have a CP (Candecomp/Parafac) *rank* k if it can be (minimally) written as the sum of k rank-1 tensors

$$T = \sum_{i\in[k]} w_i a_i \otimes b_i \otimes c_i, \quad w_i \in \mathbb{R},\ a_i, b_i, c_i \in \mathbb{R}^d. \tag{11}$$

Note that $v^{\otimes p} = v \otimes v \otimes v \cdots \otimes v$, where v is repeated p times.

Definition 2 (Row-wise Kronecker product) *For matrices $A, B \in \mathbb{R}^{d\times k}$, the* Row-wise Kronecker product $\in \mathbb{R}^{d\times k^2}$ *is defined below*

$$\begin{bmatrix} \frac{a^{(1)}}{a^{(2)}} \\ \hline \vdots \\ \hline \frac{}{a^{(k)}} \end{bmatrix} \odot \begin{bmatrix} \frac{b^{(1)}}{b^{(2)}} \\ \hline \vdots \\ \hline \frac{}{b^{(k)}} \end{bmatrix} = \begin{bmatrix} \frac{a^{(1)} \otimes b^{(1)}}{a^{(2)} \otimes b^{(2)}} \\ \hline \vdots \\ \hline \frac{}{a^{(k)} \otimes b^{(k)}} \end{bmatrix},$$

where $a^{(i)}, b^{(i)}$ are rows of A, B respectively. Note that our definition is different from usual definition of Khatri-Rao product which is a column-wise Kronecker product (is performed on columns of matrices).

Tensor as multilinear form: We view a tensor $T \in \mathbb{R}^{d\times d\times d}$ as a multilinear form. Consider matrices $M_l \in \mathbb{R}^{d\times d_l}, l \in \{1, 2, 3\}$. Then tensor $T(M_1, M_2, M_3) \in \mathbb{R}^{d_1} \otimes \mathbb{R}^{d_2} \otimes \mathbb{R}^{d_3}$ is defined as

$$T(M_1, M_2, M_3)_{i_1,i_2,i_3} := \sum_{j_1,j_2,j_3\in[d]} T_{j_1,j_2,j_3} \cdot M_1(j_1, i_1) \cdot M_2(j_2, i_2) \cdot M_3(j_3, i_3). \tag{12}$$

In particular, for vectors $u, v, w \in \mathbb{R}^d$, we have [2]

$$T(I, v, w) = \sum_{j,l \in [d]} v_j w_l T(:, j, l) \ \in \mathbb{R}^d, \tag{13}$$

which is a multilinear combination of the tensor mode-1 fibers. Similarly $T(u, v, w) \in \mathbb{R}$ is a multilinear combination of the tensor entries, and $T(I, I, w) \in \mathbb{R}^{d \times d}$ is a linear combination of the tensor slices.

Derivative: For function $g(x) : \mathbb{R}^d \to \mathbb{R}$ with vector input $x \in \mathbb{R}^d$, the m-th order derivative w.r.t. variable x is denoted by $\nabla_x^{(m)} g(x) \in \bigotimes^m \mathbb{R}^d$ (which is a m-th order tensor) such that

$$\left[\nabla_x^{(m)} g(x)\right]_{i_1,\ldots,i_m} := \frac{\partial g(x)}{\partial x_{i_1} \partial x_{i_2} \cdots \partial x_{i_m}}, \quad i_1, \ldots, i_m \in [d]. \tag{14}$$

When it is clear from the context, we drop the subscript x and write the derivative as $\nabla^{(m)} g(x)$.

Derivative of product of two functions We frequently use the following gradient rule.

Lemma 6 (Product rule for gradient [7]) *For tensor-valued functions $F(x) : \mathbb{R}^n \to \bigotimes^{p_1} \mathbb{R}^n, G(x) : \mathbb{R}^n \to \bigotimes^{p_2} \mathbb{R}^n$, we have*

$$\nabla_x (F(x) \otimes G(x)) = (\nabla_x F(x) \otimes G(x))^{\langle \pi \rangle} + F \otimes \nabla_x G(x),$$

where the notation $^{\langle \pi \rangle}$ denotes permutation of modes of the tensor for permutation vector $\pi = [1, 2, \ldots, p_1, p_1 + 2, p_1 + 3, \ldots, p_1 + p_2 + 1, p_1 + 1]$. This means that the $(p_1 + 1)^{th}$ mode is moved to the last mode.

B Proof of Theorems 4, 5

B.1 Proof Theorem 4

The underlying idea behind the proof comes from Theorem 1. By Theorem 1 we have that

$$\mathbb{E}\left[y_t \otimes \mathcal{S}_2(x_t)\right] = \nabla_{x_t}^2 \mathbb{E}\left[y_t | x_t\right].$$

In order to show the derivative form more easily, let us look at derivative of each entry $i \in [d_y]$ of the vector y_t.

$$\begin{aligned}
\mathbb{E}\left[(y_t)_i | x_t\right] &= \langle (A_2)_{(i)}, (A_1 x_t + U h_{t-1})^{*2} \rangle = \sum_{j \in d_h} (A_2)_{ji} \left(\langle A_1^{(j)}, x_t \rangle + \langle U^{(j)}, h_{t-1} \rangle \right)^2, \\
\nabla_{x_t}^2 \mathbb{E}[(y_t)_i | x_t] &= \nabla_{x_t}^2 \left(\sum_{j \in d_h} (A_2)_{ji} \left(\langle A_1^{(j)}, x_t \rangle + \langle U^{(j)}, h_{t-1} \rangle \right)^2 \right) \\
&= 2 \sum_{j \in d_h} (A_2)_{ji} \nabla_{x_t} \left(\langle A_1^{(j)}, x_t \rangle + \langle U^{(j)}, h_{t-1} \rangle \right) A_1^{(j)} \\
&= 2 \sum_{j \in d_h} (A_2)_{ji} A_1^{(j)} \otimes A_1^{(j)}, \\
\nabla_{x_t}^2 \mathbb{E}\left[y_t | x_t\right] &= 2 \sum_{j \in d_h} A_2^{(j)} \otimes A_1^{(j)} \otimes A_1^{(j)},
\end{aligned}$$

and hence the form follows. By Theorem 1 we have that

$$\mathbb{E}\left[y_t \otimes \mathcal{S}_4(x_{t-1})\right] = \nabla_{x_{t-1}}^4 \mathbb{E}\left[y_t | x_{t-1}\right]$$

[2] Compare with the matrix case where for $M \in \mathbb{R}^{d \times d}$, we have $M(I, u) = Mu := \sum_{j \in [d]} u_j M(:, j) \in \mathbb{R}^d$.

In order to show the derivative form more easily, let us look at each entry $i \in [d_y]$ of the vector y_t.

$$(h_t)_k = \left(\langle A_1^{(k)}, x_t \rangle + \sum_{l \in [d_h]} U_{kl} \left(\langle A_1^{(l)}, x_{t-1} \rangle + \langle U^{(l)}, h_{t-2} \rangle \right)^2 \right)^2$$

$$\mathbb{E}[(y_t)_i | x_{t-1}] = \sum_{k \in [d_h]} (A_2)_{ki} \left(\langle A_1^{(k)}, x_t \rangle + \sum_{l \in [d_h]} U_{kl} \left(\langle A_1^{(l)}, x_{t-1} \rangle + \langle U^{(l)}, h_{t-2} \rangle \right)^2 \right)^2 .$$

The form follows directly using the derivative rule as in Lemma 6.

$$T = \nabla^4_{x_{t-1}} \mathbb{E}[y_t | x_{t-1}] = 2 \sum_{j \in d_h} A_2^{(j)} \otimes \sum_{k \in d_h} U_{jk} A_1^{(k)} \otimes A_1^{(k)} \otimes \sum_{m \in d_h} U_{jm} A_1^{(m)} \otimes A_1^{(m)},$$

Now when we reshape the above tensor T to Reshape($\nabla^4_{x_{t-1}} \mathbb{E}[y_t | x_{t-1}]$, [1 2], [3 4]) we have the form $\sum_{j \in d_h} A_2^{(j)} \otimes [U(A_1 \odot A_1)]^{(j)} \otimes [U(A_1 \odot A_1)]^{(j)}$.

B.2 Proof of Theorem 5

$$\mathbb{E}[y_t | h_t, z_t] = A_2^\top \begin{bmatrix} h_t \\ z_t. \end{bmatrix}$$

Hence,

$$\begin{aligned} T = \nabla^2_{x_t} \mathbb{E}[y_t | h_t, z_t] &= A_2^\top \begin{bmatrix} \nabla^2_{x_t} h_t \\ \nabla^2_{x_t} z_t. \end{bmatrix} \\ &= A_2^\top \begin{bmatrix} \sum_{i \in d_h} e_i \otimes (A_1)^{(i)} \otimes (A_1)^{(i)} \\ \sum_{i \in d_h} e_i \otimes (B_1)^{(i)} \otimes (B_1)^{(i)}. \end{bmatrix} \end{aligned}$$

The second Equation is direct result of Lemma 4. Therefore, if we decompose the above tensor, the first mode yields the matrix A_2. Next we remove the effect of A_2 by multiplying its inverse to the first mode of the moment tensor T. By the above Equations, we readily see that

$$T((A_2)^{-1}, I, I) = \begin{bmatrix} \sum_{i \in d_h} e_i \otimes (A_1)^{(i)} \otimes (A_1)^{(i)} \\ \sum_{i \in d_h} e_i \otimes (B_1)^{(i)} \otimes (B_1)^{(i)}. \end{bmatrix}$$

This means $T((A_2)^{-1}, I, I) = \sum_{i \in d_h} e_i \otimes c_i \otimes c_i$, where $c_i = \begin{bmatrix} (A_1)^{(i)} \\ (B_1)^{(i)} \end{bmatrix}$. Hence, Algorithm 2 correctly recovers A_2, A_1, B_1. Recovery of U, V directly follows Lemma 7.

C GLOREE for General Polynomial Activation Functions

C.1 GLOREE for IO-RNN with polynomial activation functions

Here, we consider an RNN with polynomial activation function of order $l \geq 2$, i.e.,

$$\mathbb{E}[y_t | h_t] = A_2^\top h_t, \quad h_t = \text{poly}_l(A_1 x_t + U h_{t-1}). \tag{15}$$

We have the following properties.

Theorem 7 (Learning parameters of RNN for general polynomial activation function) *The following is true:*

$$\mathbb{E}\left[y_t \otimes \mathcal{S}_2(x[n], t)\right] = \sum_{i \in d_h} \mu_i A_2^{(i)} \otimes A_1^{(i)} \otimes A_1^{(i)}. \tag{16}$$

Hence, we can recover A_1, A_2 via tensor decomposition assuming that they are full row rank.
In order to learn U we form the tensor $\mathbb{E}\left[y_t \otimes \mathcal{S}_{l^2}(x_{t-1})\right]$. Then we have

$$\mathbb{E}\left[y_t \otimes \textit{Reshape}(\mathcal{S}_{l^2}(x[n], t-1), 1, [1 \ \dots \ l], \dots, [l^2 - l + 1 \ \dots \ l^2]\right] = \sum_{i \in d_h} A_2^{(i)} \otimes \left[\left[U(A_1^{\odot l})\right]^{(i)}\right]^{\otimes l}. \tag{17}$$

Hence, we can recover $U(A_1^{\odot l})$ via tensor decomposition under full row rank assumption. Since A_1 is previously recovered, U can be recovered. Thus, Algorithm 3(GLOREE) consistently recovers the parameters of IO-RNN with polynomial activations.

Remark on form of the cross-moment tensor: The cross-moment tensor in Equation (17), is a tensor of order $l+1$, where modes $2, \dots, l+1$ are similar, i.e., they all correspond to rows of the matrix $U(A_1^{\odot l}) = U(A_1 \odot A_1 \dots \odot A_1)$, where A_1 has gone through row-wise Kronecker product l times [3]. This is a direct extension of the form in Theorem 4 from $l = 2$ to any $l \geq 2$.

Remark on coefficients μ_i: For the cross-moment tensor in (7), the coefficients μ_i are the expected values of derivatives of activation function. More concretely, if activation is a polynomial of degree l, we have that $\mu_i = \mathbb{E}\left[\mathrm{poly}_{l-2}\left(\langle A_1^{(i)}, x_t\rangle + \langle U^{(i)}, h_{t-1}\rangle\right)\right]$, where poly_{l-2} denotes a polynomial of degree $l-2$. Similarly, the coefficients of the tensor decomposition in (17) correspond to expectations over derivatives of (recursive) activation functions. We assume that these coefficients are non-zero in order to recover the weight matrices.

Remark on tensor decomposition via sketching: Consider line 10 in Algorithm 3 and line 7 in Algorithm 4. Here we are decomposing a tensor of order $l+1$. In order to perform this with efficient computational complexity, we can use tensor sketching proposed by Wang et al. [11]. They do not form the moment tensor explicitly and directly compute tensor sketches from data. This avoids the exponential blowup in computation, i.e., it reduces the computational complexity from m^{l+1} to $(m + m \log m)n$, where m is the sketch length and n denotes the number of samples. As expected, there is a trade off between the sketch length and the error in recovering the tensor components. For details, see [11].

Proof: By Theorem 1, we have that

$$\begin{aligned}
&\mathbb{E}\left[y_t \otimes \mathcal{S}_2(x[n], t)\right] = \nabla_{x_t}^2 \mathbb{E}\left[y_t | x_t\right], \\
&\mathbb{E}\left[y_t \otimes \text{Reshape}(\mathcal{S}_{l^2}(x[n], t-1), [1 \ \dots \ l], \dots, [l^2 - l + 1 \ \dots \ l^2])\right] \\
&= \mathbb{E}\left[\text{Reshape}(\nabla_{x_t}^{l^2} y_t, 1, [2 \ \dots \ l+1], \dots, [l^2 \ \dots \ l^2 + 1])\right].
\end{aligned}$$

The form follows directly using the derivative rule as in Lemma 6.

□

Thus, we provide an efficient framework for recovering all the weight matrices of an input-output recurrent neural network using tensor decomposition methods.

C.2 GLOREE-B for BRNN with general polynomial activation function

In Algorithm 4, we show the complete algorithm for training BRNN when the activation functions are polynomials of order l. The analysis directly follows from analysis of BRNNs with quadratic activation functions and the extension to $l \geq 2$ is similar to extension of IO-RNNs with quadratic activation functions to general polynomials of order $l \geq 2$.

[3] Since row-wise Kronecker product (as defined in notations) does not change the number of rows, this matrix multiplication is valid.

D Sample complexity analysis

We now analyze the sample complexity for GLOREE. We first start with the concentration bound for the moment tensor and then use analysis of tensor decomposition to show that our method has a sample complexity that is a polynomial of the model parameters.

We first derive concentration bounds for the empirical moment tensors. This is a bit more involved since we have a non-i.i.d. sequence. Moreover, since we assume a polynomial activation function, the hidden state h_t can grow in an unbounded manner. To avoid this, without loss of generality, we assume $\|x_i\|_2 < 1, \ \forall i \in [n]$ (with high probability). We also need the three additional assumptions mentioned below.

Concentration bounds for functions over Markov chains: Our input sequence $x[n]$ is a geometrically ergodic Markov chain. We can think of the empirical moment $\hat{\mathbb{E}}[y_t \otimes \mathcal{S}_m(x[n], t)]$ as functions over the samples $x[n]$ of the Markov chain. Note that this assumes $h[n]$ and $y[n]$ as deterministic functions of $x[n]$, and our analysis can be extended when there is additional randomness. Kontorovich and Weiss [16] provide the result for scalar functions and this is an extension of that result to matrix-valued functions.

We now recap concentration bounds for general functions on Markov chains. For any ergodic Markov chain with the stationary distribution ω, denote $f_{1\to t}(x_t|x_1)$ as state distribution given initial state x_1. The inverse mixing time is defined as follows

$$\rho_{\text{mix}}(t) = \sup_{x_1} \|f_{1\to t}(x_t|x_1) - \omega\|.$$

Kontorovich and Weiss [16] show that

$$\rho_{\text{mix}}(t) \leq G\theta^{t-1},$$

where $1 \leq G < \infty$ is geometric ergodicity and $0 \leq \theta < 1$ is the contraction coefficient of the Markov chain.

In the IO-RNN (and BRNN) model, the output is a nonlinear function of the input. Hence, the next step is to deal with this non-linearity. Kontorovich and Weiss [16] analyze the mixing of a (scalar) nonlinear function through its Lipschitz property. In order to analyze how the empirical moment tensor concentrates, we define the Lipschitz constant for matrix valued functions.

Definition 3 (Lipschitz constant for a matrix-valued function of a sequence) *A matrix-valued function $\Phi : \mathbb{R}^{nd_x} \to \mathbb{R}^{d_1 \times d_2}$ is c-Lipschitz with respect to the spectral norm if*

$$\sup_{x[n], \tilde{x}[n]} \frac{\|\Phi(x[n]) - \Phi(\tilde{x}[n])\|}{\|x[n] - \tilde{x}[n]\|_2} \leq c,$$

where $x[n], \tilde{x}[n]$ are any two possible sequences of observations. Here $\|\cdot\|$ denotes the spectral norm and $\mathbb{R}^{nd_x}$ is the state space for a sequence of n observations $x[n]$.

Concentration of empirical moments of IO-RNN and BRNN: In order to ensure that the empirical moment tensor has a bounded Lipschitz constant, we need the following assumptions.

Assumptions:

- Without loss of generality, we assume that the input sequence is bounded by 1 with high probability, $\|x_i\| < 1 \ \forall i \in [n]$
- Assume that $\|A_1\| + \|U\| \leq 1$.
- If the activation function is a polynomial of order l, we need $\|U\| \leq 1/l$.
- $\max_{i\in[n]} \|\mathcal{S}_2(x[n], i)\|$, $\max_{i\in[n]} \{\|\mathcal{S}_3(x[n], i) - \mathcal{S}_2(x[n], i) \otimes \mathcal{S}_1(x[n], i)\|\}$ are bounded.

Under the above assumptions, we have that

Lemma 8 (Lipschitz property for the Empirical Moment Tensor) *For the IO-RNN discussed in* (15)*, if the above assumptions hold, the matricized tensor Mat* $\left(\hat{\mathbb{E}}[y_i \otimes \mathcal{S}_2(x[n], i)]\right)$ *is a function of the input sequence with Lipschitz constant*

$$c \leq \frac{1}{n} \frac{\|A_2\|\|A_1\|}{1 - l\|U\|} \max_{i \in [n]} \|\mathcal{S}_2(x[n], i)\| + \|A_2\|\| \max_{i \in [n]} \{\|\mathcal{S}_3(x[n], i) - \mathcal{S}_2(x[n], i) \otimes \mathcal{S}_1(x[n], i)\|\}. \tag{18}$$

Remark: For a Gaussian AR-1 process, form of the score function is shown in (5). Since $|\phi| \leq 1$, we have that

$$\max_{i \in [n]} \|\mathcal{S}_2(x[n], i)\| \leq (1 + |\phi|)^4 + (1 + |\phi|)^2 \leq 20,$$
$$\max_{i \in [n]} \{\|\mathcal{S}_3(x[n], i) - \mathcal{S}_2(x[n], i) \otimes \mathcal{S}_1(x[n], i)\|\} \leq (1 + |\phi|)^4 \leq 16.$$

As we see, the third assumption is met and the Lipschitz constant is

$$c \leq \frac{20}{n} \frac{\|A_2\|\|A_1\|}{1 - l\|U\|} + 16\|A_2\|.$$

D.1 Proof of lemma 8: Lipschitz property for the Empirical Moment Tensor

In order to prove lemma 8, we first need to show that the matricized cross-moment tensor is a Lipschitz function of the input sequence and find the Lipschitz constant.

We first show that the output function is a Lipschitz function of the input sequence and find the Lipschitz constant. In order to prove this, we need the above assumptions to ensure a bounded hidden state and a bounded output sequence. Then, we have that

Lemma 9 (Lipschitz property for the Output of IO-RNN) *For the IO-RNN discussed in* (15)*, if the above assumptions hold, then the output is a Lipschitz function of the input sequence with Lipschitz constant* $\frac{1}{n} \frac{\|A_2\|\|A_1\|}{1 - l\|U\|}$ *w.r.t.* ℓ_2 *metric.*

Proof: This follows directly from the definition. In order to find the Lipschitz constant, we need to sum over all possible changes in the input sequence [15]. Therefore, we bound derivative of the function w.r.t. each input entry and then take the average the results to provide an upper bound on the Lipschitz constant. With the above assumptions, it is straightforward to show that

$$\|\nabla_{x_i} y_t\| \leq l^{t-i+1} \|A_2\|\|A_1\|\|U\|^{t-i}.$$

Taking the average of this geometric series for $t \in [n]$ and large sample sequence, we get $\frac{1}{n} \frac{\|A_2\|\|A_1\|}{1 - l\|U\|}$ as the Lipschitz constant. □

Next we want to find the Lipschitz constant for the matricized tensor $T = \mathbb{E}[y_i \otimes \mathcal{S}_2(x[n], i)]$ which is a function of the input sequence. Considering the rule for derivative of product of two functions as in Lemma 6 we have that

$$\nabla_{x_i} y_i \otimes \mathcal{S}_2(x[n], i) = \mathcal{S}_2(x[n], i) \otimes \nabla_{x_i} y_i + y_i \otimes \nabla_{x_i} \mathcal{S}_2(x[n], i)$$

Note that by definition of score function in (3), we have that

$$\nabla_{x_i} \mathcal{S}_2(x[n], i) = \mathcal{S}_3(x[n], i) - \mathcal{S}_2(x[n], i) \otimes \mathcal{S}_1(x[n], i).$$

Hence,

$$\|\nabla_{x_i} y_i \otimes \mathcal{S}_2(x[n], i)\| = \|\mathcal{S}_2(x[n], i) \otimes \nabla_{x_i} y_i + y_i \otimes \nabla_{x_i} \mathcal{S}_2(x[n], i)\|$$
$$\leq \max_{i \in [n]} \|\mathcal{S}_2(x[n], i)\|\|A_2\| + \|\nabla_{x_i} y_i\| \max_{i \in [n]} \{\|\mathcal{S}_3(x[n], i) - \mathcal{S}_2(x[n], i) \otimes \mathcal{S}_1(x[n], i)\|\}$$

and assuming that $\max_{i\in[n]} \|\mathcal{S}_2(x[n],i)\|$, $\max_{i\in[n]} \{\|\mathcal{S}_3(x[n],i) - \mathcal{S}_2(x[n],i)\otimes\mathcal{S}_1(x[n],i)\|\}$ are bounded values, it is straightforward to show that Mat $(\mathbb{E}[y_i \otimes \mathcal{S}_2(x[n],i)])$ is Lipschitz with Lipschitz constant

$$c = \frac{1}{n}\frac{\|A_2\|\|A_1\|}{1-l\|U\|}\|\max_{i\in[n]}\mathcal{S}_2(x[n],i)\| + \|A_2\|\|\max_{i\in[n]}\nabla_{x_i}\mathcal{S}_2(x[n],i)\|$$

Now that we have proved the Lipschitz property for the cross-moment tensor, we can prove the concentration bound for the IO-RNN.

Theorem 10 (Concentration bound for RNN) *For the IO-RNN discussed in* (15)*, let $z[n]$ be the sequence of matricized empirical moment tensors Mat $\left(\hat{\mathbb{E}}[y_i \otimes \mathcal{S}_2(x[n],i)]\right)$ for $i \in [n]$. Then,*

$$\|z - \mathbb{E}(z)\| \leq G\frac{1+\frac{1}{\sqrt{8}cn^{1.5}}}{1-\theta}\sqrt{8c^2 n\log\left(\frac{d_y+d_x^2}{\delta}\right)},$$

with probability at least $1-\delta$, where $\mathbb{E}(z)$ is expectation over samples of Markov chain when the initial distribution is the stationary distribution and c is specified in Equation (18)*.*

Proof of Theorem 10

In order to get the complete concentration bound in Theorem 10, we need Lemma 8 in addition to the following Theorem.

Theorem 11 *(Concentration bound for a matrix-valued function of a Markov chain] Consider a Markov chain with observation samples $x[n] = (x_1,\ldots,x_n) \in S^n$, geometric ergodicity G, contraction coefficient θ and an arbitrary initial distribution. For any c-Lipschitz matrix-valued function $\Phi(\cdot) : S^n \to \mathbb{R}^{d_1\times d_2}$, we have*

$$\|\Phi - \mathbb{E}[\Phi]\| \leq G\frac{1+\frac{1}{\sqrt{8}cn^{1.5}}}{1-\theta}\sqrt{8c^2 n\log\left(\frac{d_1+d_2}{\delta}\right)},$$

with probability at least $1-\delta$, where $\mathbb{E}(\Phi)$ is expectation over samples of Markov chain when the initial distribution is the stationary distribution.

Proof: The proof follows result of [15], [22], and Matrix Azuma theorem (which can be proved using the analysis of [17] for sum of random matrices). The upper bound can be decoupled into two parts, (1) $\|\Phi - \mathbb{E}[\Phi]\|$ where the expectation is over the same initial distribution as used for Φ and (2) the difference between $\mathbb{E}[\Phi]$ for the case where the initial distribution is the same initial distribution as used for Φ and the initial distribution being equal to the stationary distribution. It is direct from analysis of [16] that the latter is upper bounded by $\sum_i G\theta^{-(i-1)} \leq \frac{G}{1-\theta}$. The former can be bounded by Theorem 12 below and hence Theorem 11 follows. □

Theorem 12 (Matrix Azuma [23]) *Consider Hidden Markov Model with finite sequence of n samples S_i as observations given arbitrary initial states distribution and c-Lipschitz matrix valued function $\Phi : S_1^n \to \mathbb{R}^{d_1\times d_2}$, then*

$$\|\Phi - \mathbb{E}[\Phi]\| \leq \frac{1}{1-\theta}\sqrt{8c^2 n\log\left(\frac{d_1+d_2}{\delta}\right)},$$

with probability at least $1-\delta$. The $\mathbb{E}[\Phi]$ is given the same initial distribution of samples.

Proof: This proof is from [23] and is repeated here for completeness. Theorem 7.1 [17] provides the upper confidence bound for summation of matrix random variables. Consider a finite sequence of matrices $\Psi_i \in \mathbb{R}^{d_1\times d_2}$. The variance parameter σ^2 is the upper bound for $\sum_i \left[\Psi_i - \mathbb{E}_{i-1}[\Psi_i]\right], \forall i$ and we have that

$$\|\sum_i \left[\Psi_i - \mathbb{E}_{i-1}[\Psi_i]\right]\| \leq \sqrt{8\sigma^2\log\frac{d_1+d_2}{\delta}},$$

with probability at least $1-\delta$. For function Φ, we define the martingale difference of function Φ as the input random variable with arbitrary initial distribution over states.

$$\text{MD}_i(\Phi; S_1^i) = \mathbb{E}[\Phi|S_1^i] - \mathbb{E}[\Phi|S_1^{i-1}],$$

where S_i^j is the subset of samples from i-th position in sequence to j-th one. Hence, the summation over these set of random variables gives $\mathbb{E}[\Phi|S_1^n] - \mathbb{E}[\Phi] = \Phi(S_1^n) - \mathbb{E}[\Phi]$, $\mathbb{E}[\Phi]$ is the expectation with the same initial state distribution.

Then it remains to find σ which is the upper bound for $\|\text{MD}_i(\Phi; S_1^i)\|$ for all possible sequences. Define $\text{MD}_i(\Phi) = \max_{S_1^i} \text{MD}_i(\Phi; S_1^i)$. By [16], $\text{MD}_i(\Phi)$ is a c-Lipschitz function and is upper bounded by $G\theta(n-i)$.

□

Considering the analysis of tensor decomposition analysis in [8], Theorem 10 implies polynomial sample complexity for GLOREE. The sample complexity is a polynomial of $(d_x, d_y, d_h, G, \frac{1}{1-\theta}, \sigma_{\min}^{-1}(A_1), \sigma_{\min}^{-1}(A_2), \sigma_{\min}^{-1}(U))$. Detailed proof is similar to analysis in [8], [23] Note that with similar analysis we can prove polynomial sample complexity for GLOREE-B.

E Discussion

E.1 Score Function Estimation

According to [7], there are various efficient methods for estimating the score function. The framework of score matching is popular for parameter estimation in probabilistic models [24, 25], where the criterion is to fit parameters based on matching the data score function. Swersky et al. [25] analyze the score matching for latent energy-based models. In deep learning, the framework of auto-encoders attempts to find encoding and decoding functions which minimize the reconstruction error under added noise; the so-called Denoising Auto-Encoders (DAE). This is an unsupervised framework involving only unlabeled samples. Alain and Bengio [26] argue that the DAE approximately learns the first order score function of the input, as the noise variance goes to zero. Sriperumbudur et al. [9] propose non-parametric score matching methods that provides the non-parametric score function form for infinite dimensional exponential families with guaranteed convergence rates. Therefore, we can use any of these methods for estimating $\mathcal{S}_1(x[n], [n])$ and use the recursive form [7].

$$\mathcal{S}_m(x[n], [n]) = -\mathcal{S}_{m-1}(x[n], [n]) \otimes \nabla_{x[n]} \log p(x[n]) - \nabla_{x[n]} \mathcal{S}_{m-1}(x[n], [n])$$

to estimate higher order score functions.

E.2 Training IO-RNN and BRNN with scalar output

In the main text, we discussed training IO-RNNs and BRNNs with vector outputs. Here we expand the results to training IO-RNNs and BRNNs with scalar outputs. Note that in order to recover the parameters uniquely, we need the cross-moment to be a tensor of order at least 3. This is due to the fact that in general matrix decomposition does not provide unique decomposition for non-orthogonal components. In order to obtain a cross-moment tensor of order at least 3, since the output is scalar, we needs its derivative tensors of order at least 3. In order to have a non-vanishing gradient, the activation function needs to be a polynomial of order $l \geq 3$.

Hence, our method can also be used for training IO-RNN and BRNN with scalar output if the activation function is a polynomial of order $l \geq 3$, i.e., Let y_t be the output of

$$\mathbb{E}[y_t|h_t] = \langle a_2, h_t \rangle, \quad h_t = \text{poly}_l(A_1 x_t + U h_{t-1}),$$

where $x_t \in \mathbb{R}^{d_x}, h_t \in \mathbb{R}^{d_h}, y_t \in \mathbb{R}$ and hence $A_1 \in \mathbb{R}^{d_h \times d_x}, U \in \mathbb{R}^{d_h \times d_h}$, $a_2 \in \mathbb{R}^{d_h}$. We can learn the parameters of the model using GLOREE with guarantees.

We have

Lemma 13 (Learning parameters of RNN for general activation function, scalar output)

$$\mathbb{E}\left[y_t \otimes \mathcal{S}_3(x_t)\right] = \sum_{i \in d_h} \mu_i a_{2i} A_1^{(i)} \otimes A_1^{(i)} \otimes A_1^{(i)},$$
$$\mu_i = \mathbb{E}\left[\left(\langle A_1^{(i)}, x_t\rangle + \langle U^i, h_{t-1}\rangle\right)^{*(l-3)}\right]$$

In order to learn U, we form the tensor $\hat{\mathbb{E}}\left[y_t \otimes \textit{Reshape}(\mathcal{S}_{l^2}(x_{t-1}), 1, [1 \ \dots \ l], \dots, [l^2 - l + 1 \ \dots \ l^2]\right]$. Then we have

$$\mathbb{E}\left[y_t \otimes \textit{Reshape}(\mathcal{S}_{l^2}(x_{t-1}), 1, [1 \ \dots \ l], \dots, [l^2 - l + 1 \ \dots \ l^2]\right] = \sum_{i \in d_h} (a_2)_i \otimes \left[\left[U(A_1^{\odot l})\right]^{(i)}\right]^{\otimes l},$$

where $\odot$ is the row-wise Kronecker product defined in 1. Hence, since we know A_1, we can recover U via tensor decomposition.

We can prove that we can learn the parameters of a BRNN with scalar output and polynomial activation functions of order $l \geq 3$ using the same trend as for Lemma 5.

E.3 Training Linear IO-RNN

In the paper we discussed the problem of training IO-RNNs with polynomial activation function of order $l \geq 2$. Here we propose a method for training IO-RNNs with linear activation functions. Although our proposed methods for two cases differ in nature, we include both of them for completeness and covering all cases.

Sedghi and Anandkumar [27] provide a method to train first layer of feed-forward neural networks using the first-order score function of the input. For a NN with vector output, their formed cross-moment is a matrix of the form $\mathbb{E}[y \otimes \mathcal{S}_1(x)] = BA_1$, where A_1 is the weight matrix for the first layer and B is a matrix that includes the rest of the derivative matrix. Then they argie that if A_1 is sparse, the problem of recovering A_1 is a sparse dictionary learning problem that can be solved efficiently using Sparse Dictionary Learning Algorithm [28].

Here we show that for IO-RNN with linear activation function, we can expand the result of Sedghi and Anandkumar [27] to the non-i.i.d. input sequence.

Let

$$y_t = A_2^\top h_t, \quad h_t = A_1 x_t + U h_{t-1},$$

where $x_t \in \mathbb{R}^{d_x}, h_t \in \mathbb{R}^{d_h}, y_t \in \mathbb{R}^{d_y}, A_2^\top \in \mathbb{R}^{d_h \times d_y}$ and hence $A_1 \in \mathbb{R}^{d_y \times d_x}, U \in \mathbb{R}^{d_y \times d_h}$.

Let $\tilde{y}[n] = [y_1, y_2, \dots, y_n], \tilde{x}[n] = [x_1, x_2, \dots, x_n]$. Similar to our earlier analysis, we have

$$\mathbb{E}[\tilde{y}[n] \otimes \mathcal{S}(\tilde{x}[n], [n]) = \nabla_{\tilde{x}[n]} \tilde{y}[n]$$

For our linear model the derivative has a Toeplitz form. Assuming that A_1 is sparse, we can use this structure and Sparse Dictionary Learning Algorithm [28] to recover the model parameters.

Below we write the cross-moment Topelitz form for $n = 4$ for simplicity.

$$\mathbb{E}[\tilde{y}[n] \otimes \mathcal{S}(\tilde{x}[n], [n]) = \begin{bmatrix} A_2^\top A_1 & 0 & 0 & 0 \\ A_2^\top U A_1 & A_2^\top A_1 & 0 & 0 \\ A_2^\top U^2 A_1 & A_2^\top U A_1 & A_2^\top A_1 & 0 \\ A_2^\top U^3 A_1 & A_2^\top U^2 A_1 & A_2^\top U A_1 & A_2^\top A_1 \end{bmatrix}$$

If we recover the Toeplitz structure, we have access to the following matrices: $A_2 A_1, A_2 U A_1, \dots, A_2 U^n A_1$. Next, we put these matrices in a new matrix C as below.

$$C = \begin{bmatrix} A_2 A_1 \\ A_2 U A_1 \\ \vdots \\ A_2 U^n A_1 \end{bmatrix}$$

It is easy to see that $C = BA_1$ for matrix B as shown below

$$B = \begin{bmatrix} A_2A_1 \\ A_2UA_1 \\ \vdots \\ A_2U^nA_1 \end{bmatrix}$$

Now assuming that A_1 is sparse and B is full column-rank, we can recover A_1 using Sparse Dictionary Learning Algorithm [28].

Let $U = V\Lambda V^\top$ be the singular-value decomposition of U, where $\Lambda = \mathrm{Diag}(\lambda_i)$. It is easy to show that, to ensure that B is full column-rank, we need the singular-values of U to satisfy $\lambda_i \sim \frac{1}{\sqrt{d_h}}$. Once we recover A_1, we can recover $A_2 = A_2A_1A_1^{-1}$ and $U = A_2^{-1}A_2UA_1A_1^{-1}$.

F Spectral Decomposition Algorithm

As part of GLOREE, we need a spectral/tensor method to decompose the cross-moment tensor to its rank-1 components. Refer to notation for definition of tensor rank and its rank-1 components. As depicted in notation, we are considering CP decomposition. Note that CP tensor decomposition for various scenarios is extensively analyzed in the literature [29], [30], [31], [32], [7], [8]. We follow the method in [8].

The only difference between our tensor decomposition setting and that of [8] is that they have a symmetric tensor (i.e., $\hat{T} = \sum_{i\in[r]} c_i \otimes c_i \otimes c_i$) whereas in GLOREE, we have two asymmetric tensor decomposition procedures in the form of $\hat{T} = \sum_{i\in[r]} b_i \otimes c_i \otimes c_i$. Therefore,

1 We first make a symmetric version of our tensor. For our specific case, this includes multiplying the first mode of the tensor with a matrix D, such that $\hat{T}(D, I, I) \simeq \sum_{i\in[r]} c_i \otimes c_i \otimes c_i$. We use the rule presented in [33] to form the symmetrization tensor. For example for $\hat{T} = \mathbb{E}[y_i \otimes \mathcal{S}_2(x[n], i)]$, we use $\hat{D} = [\mathbb{E}[y_i \otimes \mathcal{S}_1(x[n], i)]]^{-1}$.

2 Next, we run the tensor decomposition procedure as in [8] to recover estimates of $\hat{c}_i, i \in [r]$. The steps are shown in Figure 2. For more details, see [8].

3 The last step includes reversing the effect of symmetrization matrix D to recover estimate of $\hat{b}_i, i \in [r]$. For more discussion on symmetrization, see [33].

Our overall algorithm is shown in Algorithm 5.

Remark on tensor decomposition via sketching: Consider line 10 in Algorithm 3 and line 7 in Algorithm 4. Here we are decomposing a tensor of order $l + 1$. The tensor decomposition algorithm for third order tensor readily generalizes to higher order tensors. In order to perform this with efficient computational complexity, we can use tensor sketching proposed by Wang et al. [11]. They do not form the moment tensor explicitly and directly compute tensor sketches from data. This avoids the exponential blowup in computation, i.e., it reduces the computational complexity from m^{l+1} to $(m + m\log m)n$, where m is the sketch length and n denotes the number of samples. As expected, there is a trade off between the sketch length and the error in recovering the tensor components. For details, see [11].

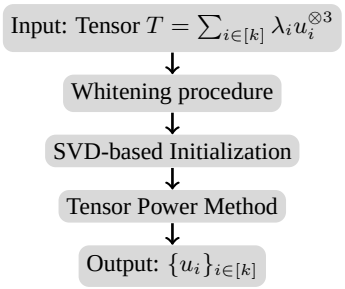


Figure 2: Overview of tensor decomposition algorithm for a symmetric third order tensor [8].

Algorithm 3 GLOREE (Guaranteed Learning Of Recurrent nEural nEtworks) for vector input

input (a) Labeled samples $\{(x_i, y_i) : i \in [n]\}$ from IO-RNN model in Figure 1(b), polynomial order l for activation function.

1: Compute 2nd-order score function $\mathcal{S}_2(x[n], i)$ of the input sequence as in Equation (4).
2: Compute $\widehat{T} := \widehat{\mathbb{E}}\left[y_i \otimes \mathcal{S}_2(x[n], i)\right]$.The empirical average is over a single sequence.
3: $\{\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\widehat{T})$; see Appendix F.
4: $\hat{A}_2 = \hat{R}_1, \hat{R}_1 = \left(\hat{R}_2 + \hat{R}_3\right)/2$.
5: Compute l^2th-order score function $\mathcal{S}_{l^2}(x[n], i)$ of the input sequence as in Equation (4).
6: Compute $\widehat{T} = \hat{\mathbb{E}}\left[y_t \otimes \text{Reshape}(\mathcal{S}_{l^2}(x[n], t-1), [1 \ \ldots \ l], \ldots, [l^2 - l + 1 \ \ldots \ l^2]\right]$.
7: $\{\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\widehat{T})$; using sketching [11].
8: $\tilde{R} = \left(\hat{R}_2 + \hat{R}_3\right)/2$.
9: $\hat{U} = \tilde{R}\left[\hat{A}_1 \odot \hat{A}_1\right]^{\dagger}$, row-wise Kronecker product $\odot$ is defined in Definition 2.
10: **return** $\hat{A}_1, \hat{A}_2, \hat{U}$.

Algorithm 4 GLOREE-B (Guaranteed Learning Of Recurrent nEural nEtworks-Bidirection case) for general activation function

input Labeled samples $\{(x_i, y_i) : i \in [n]\}$, polynomial order l_h for activation function in the forward direction, polynomial order l_z for activation function in the backward direction.

input 2nd-order score function $\mathcal{S}_2(x[n], [n])$ of the input x; see Equation (1) for the definition.

1: Compute $\widehat{T} := \hat{\mathbb{E}}[y_i \otimes \mathcal{S}_2(x[n], i)]$.
2: $\{(\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\widehat{T})$; see Appendix F.
3: $\hat{A}_2 = \hat{R}_1$.
4: Compute $\tilde{T} = \widehat{T}(((\hat{A}_2)^\top)^{-1}, I, I)$. For definition of multilinear form see Section A.
5: $\{(\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\tilde{T})$;
6: $\hat{C} = (\hat{R}_2 + \hat{R}_3)/2$.
7: $\hat{C} = \begin{bmatrix} \hat{A}_1 \\ \hat{B}_1 \end{bmatrix}$.
8: Compute l^2th-order score function $\mathcal{S}_{l^2}(x[n], t-1)$ of the input sequence as in Equation (4).
9: Compute $\widehat{T} = \hat{\mathbb{E}}\left[y_t \otimes \text{Reshape}(\mathcal{S}_{l^2}(x[n], t-1), 1, [1 \ \ldots \ l], \ldots, [l^2 - l + 1 \ \ldots \ l^2]\right]$.
10: $\{\hat{w}, \hat{R}_1, \hat{R}_2, \hat{R}_3\} =$ tensor decomposition$(\widehat{T})$; using sketching [11].
11: $\tilde{R} = \left(\hat{R}_2 + \hat{R}_3\right)/2$.
12: $\hat{U} = \tilde{R}\left[\hat{A}_1 \odot \hat{A}_1\right]^\dagger$, row-wise Kronecker product $\odot$ is defined in Definition 2.
13: Repeat lines (8)-(11) with $\mathcal{S}_4(x[n], t+1)$ instead of $\mathcal{S}_4(x[n], t-1)$ to recover $\hat{V}$.
14: **return** $\hat{A}_1, \hat{A}_2, \hat{B}_1, \hat{U}, \hat{V}$.

Algorithm 5 Tensor Decomposition Algorithm Setup

input Asymmetric tensor T, symmetrization matrix D.

1: Symmetrize the tensor : $T = T(D, I, I)$.
2: $(A_1)_j = \text{TensorDecomposition}(T)$ as in Figure 2. For details, see [8].
3: $A_2 = D^{-1}A_1$
4: **return** A_2, A_1, A_1.