Deep Incremental RNN for Learning Sequential Data: A Lyapunov Stable Dynamical System

Ziming Zhang[#], Guojun Wu[#], Yanhua Li[#], Yun Yue[#], Xun Zhou[†]
[#]Worcester Polytechnic Institute
[†]University of Iowa
Email: {zzhang15, gwu, yli15, yyue}@wpi.edu, xun-zhou@uiowa.edu

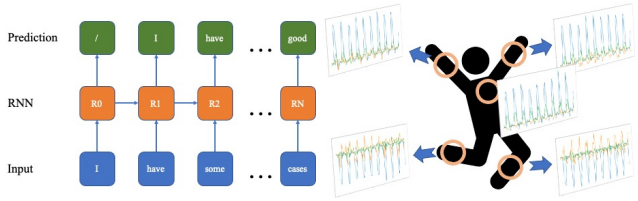
2021 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-2398-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDM51629.2021.00108

***Abstract*—With the recent advances in mobile sensing technologies, large amounts of sequential data are collected, such as vehicle GPS records, stock prices, sensor data from air quality detectors. Recurrent neural networks (RNNs) have been studied extensively to learn complex patterns for sequential data, with applicatons in natural language processing for sentence prediction/completion, human activity recognition for predicting or classifying human activities. However, there are many practical issues when training RNNs, e.g., vanishing and exploding gradients often occur due to the repeatability of network weights, etc. In this paper, we study the training stability in deep recurrent neural networks (RNNs), and propose a novel network, namely, deep incremental RNN (DIRNN). In contrast to the literature, we prove that DIRNN is essentially a Lyapunov stable dynamical system where there is no vanishing or exploding gradient in training. To demonstrate the applicability in practice, we also propose a novel implementation, namely TinyRNN, that sparsifies the transition matrices in DIRNN using weighted random permutations to reduce the model sizes. We evaluate our approach on seven benchmark datasets, and achieve state-of-the-art results. Demo code is provided in the supplementary file.**

***Index Terms*—Machine Learning, Recurrent Neural Network**

I. Introduction

Thanks to the fast development of mobile sensing technologies, large amounts of sequential data are being collected rapidly, such as vehicle GPS records, stock prices, sensor data from air quality detectors, etc. In these sequential datasets, the points in a sequence are dependent on the other points in the dataset. In practice, there are many applications in mining these sequential data. For example, *Natural Language Processing (NLP)* [1] aims to predict or guess the next word for us (Fig 1(a)), and *human activity recognition (HAR)* [2; 3] predicts or classifies human activities from sequential data collected from IoT wearable devices, such as accelerometers and gyroscopes (Fig 1(b)). Recurrent neural networks (RNNs) have achieved significant success in tackling these applications, i.e., learning complex patterns for sequential input data. At each time step t, an RNN stores the previous hidden state vector, $\mathbf{h}_{t-1} \in \mathbb{R}^D$, and upon receiving the current input vector, $\mathbf{x}_t \in \mathbb{R}^d$, linearly transforms the tuple $(\mathbf{h}_{t-1}, \mathbf{x}_t)$ and passes it through a non-linearity to update the state vectors over T time steps. Subsequently, RNNs output the predictions as a function of the hidden states. The model parameters (*i.e.,* state/input/prediction parameters) are learned by minimizing an empirical loss.



(a) Language modeling. (b) Human activity prediction.

Fig. 1: Illustration of various sequential tasks. For (a)language modeling task, we use sentence as input of recurrent models to predict the next word. As for (b)human activity prediction, the figure shows sensor data from IoT wearable devices on body, arms and legs respectively when the user jumps.

In the literature, there are significant amount of works on developing RNNs such as, just to name a few, long short-term memory (LSTM) [4], gated recurrent unit (GRU) [5], UGRNN [6], FastGRNN [2], unitary RNNs [7; 8; 9; 10; 11], deep RNNs [12; 13; 14], linear RNNs [15; 16; 17], residual/skip RNNs [18; 19; 20; 21; 2], ordinary differential equation (ODE) based RNNs [22; 23; 24; 2; 25; 26; 27].

Continuous-Time RNN (CTRNN). [28] introduced continuous-time RNN to mimic activation propagation in neural circuitry, which is modeled as follows:

$$\beta\dot{\mathbf{g}}_t = -\alpha\mathbf{g}_t + \phi(\mathbf{U}\mathbf{g}_t + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \tag{1}$$

where at the t-th time step, $\mathbf{x}_t \in \mathbb{R}^d$ denotes the input signal, $\mathbf{g}_t \in \mathbb{R}^D$ denotes the hidden state vector, $\dot{\mathbf{g}}_t$ denotes the change rate of the vector $\mathbf{g}_t$, ϕ denotes the activation function parametrized by $\mathbf{U} \in \mathbb{R}^{D\times D}, \mathbf{W} \in \mathbb{R}^{D\times d}, \mathbf{b} \in \mathbb{R}^D$, and $\alpha, \beta \in \mathbb{R}^+$ denote some predefined constants.

Incremental RNN (iRNN). Inspired by CTRNN, [27] introduced an incremental RNN whose hidden-state transition function is defined as follows:

$$\begin{aligned} &\mathbf{z}_t = \mathbf{g}_t + \mathbf{h}_{t-1}, \\ &\beta\dot{\mathbf{g}}_t = -\alpha\mathbf{z}_t + \phi(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \mathbf{g}_t(0) = \mathbf{0}, \mathbf{h}_t = \mathbf{g}_t^*, \end{aligned} \tag{2}$$

The first two authors contributed equally in this work.
Guojun Wu and Yanhua Li were supported in part by NSF grants IIS-1942680 (CAREER), CNS1952085, CMMI-1831140, and DGE-2021871. Ziming Zhang and Yun Yue were supported in part by NSF grant CCF-2006738. Xun Zhou was funded partially by Safety Research using Simulation University Transportation Center (SAFER-SIM). SAFER-SIM is funded by a grant from the U.S. Department of Transportation's University Transportation Centers Program (69A3551747131). However, the U.S. Government assumes no liability for the contents or use thereof.

978-1-6654-2398-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDM51629.2021.00108

where $\mathbf{g}_t(0)$ denotes the initial value for $\mathbf{g}_t$, $\mathbf{g}_t^*$ denotes an equilibrium point of the ODE, if any, and $\alpha, \beta \in \mathbb{R}^+$ are learnable through iRNN training. Then by using Euler's method to discretize the ODE in Eq. 2, we can further rewrite the equation as

$$\begin{aligned} &\mathbf{z}_t(k) = \mathbf{g}_t(k) + \mathbf{h}_{t-1}, \\ &\mathbf{g}_t(k+1) = \mathbf{g}_t(k) + \eta_t^k(\phi(\mathbf{U}\mathbf{z}_t(k) + \mathbf{W}\mathbf{x}_t + \mathbf{b}) - \alpha\mathbf{z}_t(k)), \\ &\mathbf{g}_t(0) = \mathbf{0}, \mathbf{h}_t = \mathbf{g}_t(K), k \in [K-1], \end{aligned} \tag{3}$$

where $\alpha, \eta_t^k \in \mathbb{R}^+$ are some learnable parameters. [27] proved that under mild conditions $\mathbf{g}_t(k)$ converges linearly and $\lim_{K\to\infty} \mathbf{g}_t(K) = \mathbf{g}_t^*$, if any equilibrium exists. Note that Eq. 3 is used for implementation.

Training Stability Challenge of RNNs. Vanishing and exploding gradients often occur in training RNNs, due to the repeatability of network weights in the chain rule when computing gradients, leading to instability in training with their magnitude either too small or too large. It has been proven that theoretically there is no vanishing/exploding gradient in the training of iRNN, leading to fast convergence empirically. In the literature, some other RNNs have analyzed and demonstrated their training stability as well, such as AntisymmetricRNN [24], LipschitzRNN [29], MomentumRNN [30], nnRNN [31], expRNN [32]. In particular, the stability analysis often comes with the eigenvalues of the Jacobian of the hidden state dynamics in order to study the problem of vanishing/exploding gradients. Recently, [33] proposed analyzing RNN training using attractors and smoothness as alternatives.

Lyapunov Stability in Dynamical Systems. RNNs are often considered as dynamical systems, and several derivatives are developed from this perspective, such as iRNN and AntisymmetricRNN. However, there are few works that borrow the theory of dynamical systems to analyze the RNN training stability (without strong assumptions on the Jacobian). Recently, [34] studied the Lyapunov spectra of chaotic recurrent neural networks. [35] proposed using Lyapunov exponents to understand the information propagation in RNNs, but unfortunately there is no discussion on how to introduce such nice Lyapunov stability into the development of RNNs. [36] proposed viewing neural networks from a dynamical systems perspective as pointwise affine maps. However, the theoretical results are adapted from dynamical system analysis and the assumptions for deep neural networks are too strong to be met in practice. [37] proposed an ODE based network implementation to guarantee stability as well as incorporating prior knowledge.

Deep (Stacked) RNNs. By stacking the conventional shallow RNNs such as LSTM, GRU, or iRNN, we can generate deep RNNs whose hidden states depend on the previous states along not only time steps but also network layers. In particular, in this paper we consider the deep RNN as illustrated in Fig. 2. It is well-known that deep RNNs require a considerable amount of work (such as learning rate and clipping) to ensure proper convergence. In other words, the training stability of deep RNNs can be achieved empirically with careful initialization, but its theoretical property still remains elusive.

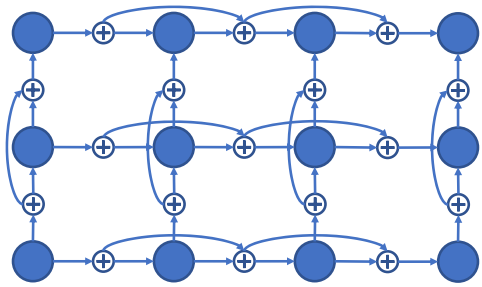


Fig. 2: Illustration of the network architecture in our DIRNN with $L = 3$ and $T = 4$. Here each blue node represents a neuron in the network that a hidden state vector is generated by an ODE.

Our Contributions. In this paper we study the training stability of deep RNNs from the perspective of Lyapunov stability, and propose a novel *deep incremental RNN* (DIRNN) that is a generalization of iRNN into the deep regime.

- *Theoretical contributions:* We prove that our DIRNN is essentially a Lyapunov stable dynamical system with a set of equilibrium points, each for a hidden state. We then prove the training stability of DIRNN where there exists no vanishing/exploding gradient. To the best of our knowledge, we are the *first* to provide such theoretical results on the training stability for deep RNNs.
- *Empirical contributions:* The model sizes of deep RNNs grow linearly with the increase of the network depth, which may limit the applications of DIRNN in reality. To address this problem, we propose a sparsified DIRNN, namely *TinyRNN*, that significantly reduces the number of parameters with marginal accuracy loss. We evaluate our approach on seven benchmark datasets and demonstrate state-of-the-art performance.

II. Related Work

RNN Architectures. LSTM [4] applies gate-controlled memory cells to mitigate the vanishing/exploding gradient issue in sequence-based tasks. Another widely-used variant of RNNs is GRU [5]. Both LSTM and GRU are developed through sophisticated recurrent units [38]. In particular, FastGRNN [2] feed-forwards state vectors to induce skip or residual connections, to serve as a middle ground between feed-forward and recurrent models, and to mitigate gradient decay. ShaRNN [39] was proposed to induce long-term dependencies and yet admit parallelization, where the first layer splits the input sequence and runs several independent RNNs, and the second layer consumes the output of the first layer using a second RNN thus capturing long term dependencies. iRNN [27] was proposed based on a novel ODE based formulation to facilitate the training of RNNs by achieving identity gradient between hidden layers and low-rank matrix decomposition. Unitary and orthogonal RNNs aim to preserve the norm of

hidden features by controlling the eigenvalues, explicitly or implicitly, that has been studied extensively in recent years [7; 8; 9; 10; 11; 29; 31; 32; 40; 41]. For instance, [32] proposed expRNN by performing the first-order optimization with orthogonal and unitary constraints based on a parametrization stemming from Lie group theory through the exponential map.

Deep RNNs. RNNs are inherently deep in time. Inspired by this property, researchers are seeking to develop new networks to investigate the benefits of depth in space of RNN architectures. For instance, [42] combined multiple recurrent levels on the basis of bi-directional LSTM [43; 44] to improve RNN performance in speech recognition task. Another study in [45], with a deeper analysis of the different emergent time scales, also proposed a similar stacking architecture. [46] proposed IndRNN where neurons in the same layer are independent of each other and they are connected across layers. Other deep RNNs have been proposed in the literature as well [47; 48; 49; 50; 51; 52; 12; 53].

Lightweight RNNs. Recently, lightweight RNNs, *i.e.,* RNNs with small model sizes, have been attracting more and more attention due to the great potentials in both academic research and industrial products [27; 39; 54; 55; 56; 57; 2]. Such RNNs can be loaded on mid-end and more powerful high-end edge devices such as smartphones, and have been introduced into some real-world applications such as machinery fault diagnosis [54]. In particular, FastGRNN [58] employed gated techniques and fixed-point quantization to further compress the models learned by FastRNN. [59; 60] proposed hybrid matrix decomposition (HMD) and Kronecker product (KP) approaches, respectively, for RNN compression on edge devices. Both approaches belong to low-rank tensor decomposition. DirNet [61] was proposed based on an optimized fast dictionary learning algorithm to compress RNNs on mobile devices.

III. DIRNN: Deep Incremental RNN

Motivation. iRNN [27] introduced an interesting notion of "identity" gradient in backpropagation. That is, given two arbitrary hidden state vectors $\mathbf{h}_t, \mathbf{h}_{t-1}, (t \in [T])$, it holds that $\partial \mathbf{h}_t + \partial \mathbf{h}_{t-1} = \mathbf{0}$ where ∂ denotes the partial derivative operator. Letting τ denote the continuous time for measuring the change rate, we then can achieve the following ODE:

$$\dot{\mathbf{h}}_t(\tau) + \dot{\mathbf{h}}_{t-1}(\tau) \equiv \frac{\partial \mathbf{h}_t(\tau)}{\partial \tau} + \frac{\partial \mathbf{h}_{t-1}(\tau)}{\partial \tau} = \mathbf{0}, \forall t \in [T]. \quad (4)$$

Now if we take $\mathbf{h}_t, \mathbf{h}_{t-1}$ as variables, then such linear dynamical systems are Lyapunov stable (see Def. 1). This novel perspective motivates us to study the training stability for deep RNNs.

Formulation. In order to prove the Lyapunov stability, we intentionally propose the following ODE based hidden-state transition function for our DIRNN with L hidden layers:

$$\begin{aligned} \mathbf{z}_{l,t} &= \mathbf{g}_{l,t} + \sum_{m=1}^{l-1} \gamma_{m,t} \mathbf{h}_{m,t} + \sum_{n=1}^{t-1} \rho_{l,n} \mathbf{h}_{l,n}, \quad (5) \\ \beta_l \dot{\mathbf{g}}_{l,t} &= -\alpha_l \mathbf{z}_{l,t} + \phi(\mathbf{U}_l \mathbf{z}_{l,t} + \mathbf{W}_l \mathbf{x}_t + \mathbf{b}_l), \\ \mathbf{g}_{l,t}(0) &= \mathbf{0}, \mathbf{h}_{l,t} = \mathbf{g}^*_{l,t}, \forall l \in [L], \forall t \in [T], \end{aligned}$$

where $\mathbf{g}_{l,t}(0)$ denotes the initial value for $\mathbf{g}_{l,t}$, $\mathbf{g}^*_{l,t}$ denotes an equilibrium point of the ODE, if any, and $\alpha_l, \beta_l \in \mathbb{R}^+, \gamma_{m,t}, \rho_{l,n} \in \mathbb{R}$ are learnable through DIRNN training. Compared with Eq. 2, we can see that the key difference between DIRNN and iRNN lies in the auxiliary variable $\mathbf{z}_{l,t}$ where DIRNN involves additional stacked hidden state vectors with suitable weights. This formulation corresponds to the network architecture in Fig. 2.

A. Stability Analysis

Definition 1 (Lyapunov Stability [62])**.** *Consider a nonlinear dynamical system $\dot{x} = f(x(\tau))$, $x(0) = x_0$ with the system state vector $x(\tau) \in \mathcal{D} \subseteq \mathbb{R}^n$ and a continuous function $f : \mathcal{D} \to \mathbb{R}^n$. Suppose that f has an equilibrium at x^* so that $f(x^*) = 0$, then this equilibrium is said to be* Lyapunov stable*, if for every $\epsilon > 0$, there exists a $\delta > 0$ such that, if $\|x(0) - x^*\| < \delta$, then for every $\tau \geq 0$ we have $\|x(\tau) - x^*\| < \epsilon$.*

In other words, if the solutions that start near an equilibrium point x^* stay near x^* forever, then x^* is Lyapunov stable.

Proposition 1 ([24])**.** *The solution of an ODE is (Lyapunov) stable if $\max_{i=1,\cdots,n} Re(\lambda_i(\mathbf{J}(t))) \leq 0, \forall t \geq 0$, where $\mathbf{J}(t) \in \mathbb{R}^{n \times n}$ denotes the Jacobian matrix of function f, $\lambda_i(\cdot)$ denotes the i-th eigenvalue, and $Re(\cdot)$ denotes the real part of a complex number.*

Lemma 1. *Consider the change rates of hidden states $\mathbf{h}$ over continuous time τ. Then DIRNN satisfies*

$$\dot{\mathbf{z}}_{l,t} = \dot{\mathbf{h}}_{l,t} + \sum_{m=1}^{l-1} \gamma_{m,t} \dot{\mathbf{h}}_{m,t} + \sum_{n=1}^{t-1} \rho_{l,n} \dot{\mathbf{h}}_{l,n} = \mathbf{0}, \forall l, \forall t. \quad (6)$$

Proof. By plugging the equilibrium point into the ODE, we have $-\alpha_l \mathbf{z} + \phi(\mathbf{U}_l \mathbf{z} + \mathbf{W}_l \mathbf{x}_t + \mathbf{b}_l) = \mathbf{0}$. Now by taking the derivative *w.r.t.* τ, we have $\alpha_l \dot{\mathbf{z}}_{l,t} = \left[\nabla \phi \cdot \mathbf{U}_l^T\right] \dot{\mathbf{z}}_{l,t}$ that holds for all possible $\mathbf{x}_t$, leading to $\dot{\mathbf{z}}_{l,t} = \mathbf{0}$. Here ∇ denotes the gradient operator and $(\cdot)^T$ denotes the matrix transpose operator. We now complete our proof. □

Theorem 1 (Stability of Hidden States)**.** *We define $\mathbf{s} = \sum_{l,t} \mathbf{z}_{l,t} = \sum_{l,t} \theta_{l,t} \mathbf{h}_{l,t}$ where $\theta_{l,t} \in \mathbb{R}, \forall l, \forall t$ denotes the combination weights. Then it holds in DIRNN that*

$$\dot{\mathbf{s}}(\tau) = \mathbf{0}, \forall \tau \geq 0, \quad (7)$$

leading to a Lyapunov stable dynamical system.

Proof. We can achieve Eq. 7 based on Eq. 6 in Lemma 1. This is equivalent to $\dot{\mathbf{s}} = \mathbf{0} \cdot \mathbf{s}$ whose Jacobian is $\mathbf{0}$ where all the eigenvalues are zeros. Then based on Prop. 1, we can complete our proof. □

Theorem 2 (Training Stability)**.** *Assuming $\theta_{l,t} \neq 0, \forall l, \forall t$ in Thm. 1, then it holds in DIRNN that $\left\| \frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}} \right\|_F$ is constant in time, leading to no vanishing or exploding gradient. Here $\|\cdot\|_F$ denotes the Frobenius norm of a matrix, respectively.*

Proof. Base on Thm. 1, we have $\frac{\partial}{\partial \tau}\left(\frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}}\right) = \frac{\partial \dot{\mathbf{s}}(\tau)}{\partial \mathbf{h}_{l,t}} = \mathbf{0}, \forall l, \forall t$. Then based on Def. 1, the magnitude of $\frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}}$ along any direction is constant in time, and thus $\left\| \frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}} \right\|_F$ is constant in time. If there were vanishing gradients over all the hidden states (*i.e.,* slow update), the F-norm would be arbitrarily small. If there were exploding gradients over at least one hidden state (*i.e.,* fluctuation), the F-norm would be arbitrarily big. We now complete our proof by the contradictions of both cases to the constant F-norm. □

B. Implementation

Similar to iRNN in Eq. 3, we apply Euler's method to compute the equilibrium point for each hidden state, *i.e.,*

$$\mathbf{z}_{l,t}(k) = \mathbf{g}_{l,t}(k) + \sum_{m=1}^{l-1} \gamma_{m,t} \mathbf{h}_{m,t} + \sum_{n=1}^{t-1} \rho_{l,n} \mathbf{h}_{l,n}, \tag{8}$$

$$\begin{aligned} &\mathbf{g}_{l,t}(k+1) \\ &= \mathbf{g}_{l,t}(k) + \eta_{l,t}^k(\phi(\mathbf{U}_l \mathbf{z}_{l,t}(k) + \mathbf{W}_l \mathbf{x}_t + \mathbf{b}_l) - \alpha_l \mathbf{z}_{l,t}(k)), \\ &\mathbf{g}_{l,t}(0) = \mathbf{0}, \mathbf{h}_{l,t} = \mathbf{g}_{l,t}(K), k \in [K-1], \forall l \in [L], \forall t \in [T]. \end{aligned}$$

Theorem 3. *Suppose that $\phi(\cdot)$ is a 1-Lipshitz function in the norm induced by $\|\cdot\|$ (*i.e., *the ℓ_2 norm), $\|\mathbf{U}_l\| < \alpha_l$, and $\eta_{l,t}^k \leq \frac{1}{\alpha_l - \|\mathbf{U}_l\|}, \forall k$, then it follows that DIRNN in Eq. 8 converges to the equilibrium point asymptotically,* i.e., *$\mathbf{h}_{l,t} = \lim_{K \to \infty} \mathbf{g}_{l,t}(K) = \mathbf{g}_{l,t}^*$. Moreover, if $\eta_{l,t}^k = \eta_{l,t}, \forall k$ holds, then the convergence is linear with the rate of $1 - \eta_{l,t}(\alpha_l - \|\mathbf{U}_l\|)$.*

Proof. Letting $T(\mathbf{g}) = \mathbf{g} + \eta_{l,t}^k(\phi(\mathbf{U}_l \mathbf{z}_{l,t}(\mathbf{g}) + \mathbf{W}_l \mathbf{x}_t + \mathbf{b}_l) - \alpha_l \mathbf{z}_{l,t}(\mathbf{g}))$, similar to iRNN, it follows that $T(\cdot)$ is a contraction: $\|T(\mathbf{g}) - T(\mathbf{g}')\| \leq [1 - \eta_{l,t}^k(\alpha_l - \|\mathbf{U}_l\|)]\|\mathbf{g} - \mathbf{g}'\| < \|\mathbf{g} - \mathbf{g}'\|$. Therefore, the Euler's method in Eq. 8 leads to asymptotic convergence, with linear rate if $\eta_{l,t}^k$ are the same over k. We then complete the proof. □

Learning Objective. Given training data $(x, y) \sim \mathcal{X} \times \mathcal{Y}$ where $x = \{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$ is a data sample and y is its label, we aim to minimize the following loss:

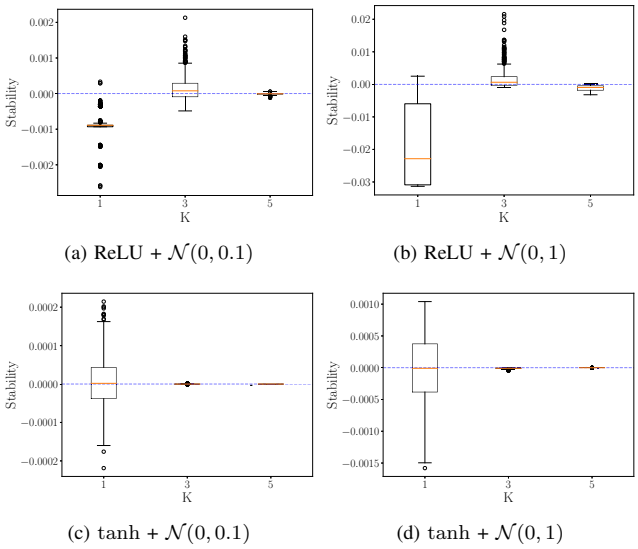
$$\min_{\alpha_l, \gamma_{l,t}, \rho_{l,t}, \eta_{l,t}^k, \omega, \mathbf{U}_l, \mathbf{W}_l, \mathbf{b}_l} \sum_{(x,y) \sim \mathcal{X} \times \mathcal{Y}} \ell\Big(y, \mathbf{z}_{L,T}; \omega\Big), \tag{9}$$

where $\ell : \mathcal{Y} \times \mathbb{R}^D \times \Omega \to \mathbb{R}$ denotes the loss function parameterized by ω such as cross-entropy.

C. Discussion

We validate our analysis on the *Adding task* that is widely used for RNN evaluation. We strictly follow [7] to generate the dataset using the public code[1]. There are two sequences with length $T = 50$. The first sequence is sampled uniformly at random $\mathcal{U}[0, 1]$. The second sequence is filled with 0 except for two entries of 1. The two entries of 1 are located uniformly at random position i_1, i_2 in the first half and second half of the sequence. The prediction value is the sum of the first sequence between $[i_1, i_2]$. The ground-truth mean squared error is 0.167 when a model simply guesses 1 as the output regardless of the input sequence. We use the value 0.167 as the baseline.

[1]https://github.com/rand0musername/urnn



(a) ReLU + $\mathcal{N}(0, 0.1)$ (b) ReLU + $\mathcal{N}(0, 1)$

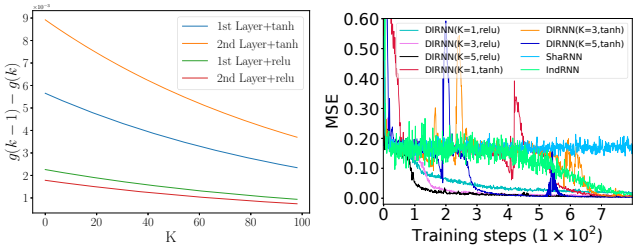
(c) tanh + $\mathcal{N}(0, 0.1)$ (d) tanh + $\mathcal{N}(0, 1)$

Fig. 3: Training stability validation for Thm 2.

Stability Validation in Thm. 2 with Random Gaussian Weights for DIRNN. Note that $\mathbf{s}$ is a variable over continuous time, which is difficult to measure directly. Instead, we utilize the sampling approach to simulate it. Specifically, first we randomly select a sample from the dataset (denoted as $\tau = 0$) and add Gaussian noise (sampled from either $\mathcal{N}(0, 0.1)$ or $\mathcal{N}(0, 1)$) to each time step of the sample to generate a new sample at the time τ, leading to 5,000 new samples in total. Then we feed these samples together with the original one to DIRNN with random Gaussian weights for $L = 3$ as demonstration. Such weights satisfy the conditions in Thm. 3. The activation functions ϕ are either ReLU or $\tanh$, and the hidden state vectors are well distributed. Finally we compute $\left[\|\frac{\partial \dot{\mathbf{s}}(\tau)}{\partial \mathbf{h}_{l,t}}\| / \|\frac{\partial \dot{\mathbf{s}}(0)}{\partial \mathbf{h}_{l,t}}\| - 1\right]_{l=3,t=1}$ to measure the stability, and illustrate the results in Fig. 3. Though the errors in computing the equilibrium points are propagated through the network, we can still observe that:

- A larger K leads to better stability, regardless of the noise, indicating that our analysis should hold for the equilibrium points, *i.e.,* $K \to +\infty$;
- With small amounts of noise, all K's can perform similarly, on average, to preserve the stability;
- With large amounts of noise, $K = 1$ seems to fail due to the poor estimation of the equilibrium points.

Such observations can be made across different network depth L, sequence length T, network weights, and activation functions that satisfy the conditions in Thm. 3. Please refer to our supplementary file for more results.



(a) Equilibrium (b) Training loss

Fig. 4: Illustration of convergence on the adding task.

Validation of Convergence to Equilibrium in Thm. 3. In general, it will be challenging to learn a deep model that can meet all the conditions in Thm. 3, except for *homogeneous* functions over $\mathbf{U}_l, \mathbf{W}_l, \mathbf{b}_l, \gamma_{l,t}, \rho_{l,t}$. To see this, we can rewrite the conditions as $\max_k\{\alpha_l - 1/\eta_{l,t}^k\} \leq \|\mathbf{U}_l\| < \alpha_l$. Since empirically it often holds that $0 \leq \alpha_l \leq 1$ and $0 \leq \eta_{l,t}^k \leq 1$, the lower bound here is often less than 0, while $\|\mathbf{U}_l\|$ is often bigger than 0. Therefore, very often the lower bound holds. To guarantee the upper bound, we can simply divide $\mathbf{U}_l, \mathbf{W}_l, \mathbf{b}_l, \gamma_{l,t}, \rho_{l,t}$ by a sufficiently large constant. Such rescaling will not affect the lower bound.

Note that the conditions in Thm. 3 are sufficient. Therefore, in practice even the function in Eq. 8 is not homogeneous, the update for ODE may still converge. We illustrate such convergence behavior in Fig. 4(a) with either ReLU (leading to a homogeneous function in Eq. 8) or $\tanh$ as our activation functions. The network architecture is the same as the one for stability validation. Then we use Eq. 8 to compute the distances between the k and $k-1$ iterations in ODE. The monotonically decreasing curves over different layers demonstrate the convergence for both activation functions.

Training Convergence on Adding Task. To demonstrate the effect of our stability analysis on training networks, we illustrate the convergence of several RNNs on the adding task in Fig. 4(b), where ShaRNN and indRNN are another two deep RNNs. As we see, our DIRNN can converge with either ReLU or $\tanh$ activations, and ReLU leads to faster convergence. The setting of ReLU plus $K = 5$ converges the fastest in terms of the training steps. Note that neither ShaRNN nor indRNN has the convergence guarantee theoretically, and thus some tricks such as gradient clipping are often used in training. In contrast, there is no need at all to use such training tricks in DIRNN to avoid vanishing/exploding gradients.

IV. TinyRNN: A Sparsified DIRNN

A critical problem in DIRNN is that the number of parameters grows linearly with the number of layers L, leading to larger model sizes and longer running time *w.r.t.* other RNNs that may limit its applicability. To address this issue, we further propose a novel implementation of DIRNN, namely TinyRNN, where all the transition matrices are sparsified. Note that our stability analysis holds for any transition matrix which guarantees that TinyRNN can be trained well.

Inspired by ShuffleNet [63] and ShufflingRNN [64], we propose reparametrizing the transition matrices in Eq. 8 using weighted permutation matrices. That is,

$$\mathbf{U}_l = \mathbf{A}_l\mathbf{B}_l, \mathbf{W}_l = \mathbf{C}_l\mathbf{D}_l, \forall l \in [L], \tag{10}$$

where $\mathbf{A}_l, \mathbf{C}_l \in \mathbb{R}^{D\times D}$ denote two diagonal weighting matrices and $\mathbf{B}_l \in \mathbb{R}^{D\times D}, \mathbf{D}_l \in \mathbb{R}^{D\times d}$ denote two permutation matrices, respectively.

Our proposed weighted permutation matrices essentially favor the learning of orthogonality in the transition matrices for training RNNs [32]. To see this, let us take the calculation of $\mathbf{Uz}$ (no subscripts for simplicity) for example, $\mathbf{Uz} = \mathbf{ABz} = [a_i\mathbf{B}(i)\mathbf{z}]_{i\in[D]}$, where a_i denotes the i-th entry along the diagonal, $\mathbf{B}(i)$ denotes the i-th row in the matrix, and $[\cdot]$ denotes the vector concatenation operator. In our permutation matrices $\mathbf{B}_l, \mathbf{D}_l, \forall l$, the rows of each matrix are predefined as orthogonally with each other as possible. Such permutation matrices define new coordinate systems where the diagonal matrices are learned for proper scaling factors.

Predefined Random Permutation Matrices. Note that learning the permutation matrices along with the TinyRNN training is very challenging, because the permutation imposes a strong constraint in the matrix structure that is difficult to be satisfied during learning. To address this problem, we propose initializing both permutation matrices randomly ($\mathbf{B}_l$ without repetition and $\mathbf{D}_l$ with repetition if $d < D$, otherwise without repetition) and fixing them during training. We only learn the weighting matrices accordingly.

Number of Model Parameters. To further reduce the model size, we simply set $\mathbf{b}_l = \mathbf{0}, \forall l$, because we only observe marginal accuracy improvement with such parameters. Therefore, the learnable parameters in our TinyRNN are $\mathbf{A}_l, \mathbf{C}_l, \alpha_l, \gamma_{l,t} = \gamma_l, \rho_{l,t} = \rho_l, \eta_{l,t}^k = \eta_l$, leading to the total number of parameters as $(2D+4)L$. We also learn a linear classifier ω on top of hidden state $\mathbf{z}_{L,T}$ consisting of $D|\mathcal{Y}|$ parameters, where $|\mathcal{Y}|$ denotes the cardinality of the label set. Together, we have $(2D+4)L + D|\mathcal{Y}|$ parameters.

Fix-Point Quantization. To further reduce the model size, the fix-point quantization techniques can be integrated into TinyRNN as well, either during or after the training. For simplicity to demonstrate this capability, in our experiments we apply the quantizer in [58] as a post-processing step to compress our models.

V. Experiments

Datasets. We test our approach on different tasks that are widely used in the literature of RNN evaluation. All the statistics of the benchmark datasets are listed in Table I.

- *Benchmark vision task:* We conduct the experiments on two datasets, *i.e.,* Pixel-MNIST and Perm-MNIST. Specifically, Pixel-MNIST refers to pixel-by-pixel sequences of images in MNIST where each 28×28 image is flattened into a 784 time sequence vector, while a random permutation to the Pixel-MNIST is applied to generate a harder time sequence dataset as Perm-MNIST.

TABLE I: Dataset statistics and default hyperparameters in DIRNN and TinyRNN. Please refer to our ablation study for more details.

Dataset	Statistics					DIRNN			TinyRNN		
	#Train	#Test	#Time	#Feat.	#Cls.	D	L	K	D	L	K
Pixel-MNIST	60k	10k	784	1	10	128	3	3	128	3	2
Perm-MNIST	60k	10k	784	1	10	128	5	3	128	3	2
Noisy-MNIST	60k	10k	1000	28	10	128	4	3	128	10	1
Noisy-CIFAR	50k	10k	1000	96	10	256	4	3	256	10	1
HAR-2	7.3k	2.9k	128	9	2	128	5	3	128	5	3
DSA-19	4.6k	4.6k	125	45	19	128	4	3	128	6	3
PTB	929k	82k	300	300	10k	300	5	3	300	5	1

- *Noise padded vision task:* For this task, we utilize Noisy-MNIST and Noisy-CIFAR. For Noisy-MNIST, each row of an MNIST image with dimension 28 is fed into the model at every time step. The first 28 time steps of input contain the original 28 rows of MNIST. The remaining 972 time steps are filled with random noise, leading to $T = 1,000$ time steps in total. Noisy-CIFAR is created in the same way with input dimension $32 * 3 = 96$ due to the RGB channels.
- *Human activity recognition task:* We test our method on HAR-2 [2] and DSA-19 [3]. HAR-2 was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. DSA-19 was collected from a resource constrained IoT wearable device with 5 Xsens MTx sensors having accelerometers, gyroscopes and magnetometers on the torso and four limbs.
- *Language modeling task:* We test our method on Penn Treebank (PTB) dataset [65]. 300 length word sequences were used for word level language modeling task using Penn Treebank (PTB) corpus. The vocabulary consisted of $10,000$ words and the size of trainable word embeddings was kept the same as hidden units. This is the setup used in [58]. Note that, for the language modeling task, just the model size of the various RNN architectures has been reported.

Baseline Algorithms.. We compare our results with baselines including vanilla RNN, LSTM [4], AntisymmetricRNN [24] [2], FastRNN [58][3], FastGRNN [58][3], IndRNN [46][4], ShaRNN [39][3], iRNN [27][5], LipschitzRNN [29][6] and MomentumRNN [30][7]. As a demonstration for comparison on model size, we employ the pruning algorithm from [66][8] as a post-processing step for TinyRNN to further compress the learned models. Note that using the public code we have verified the results of each competitor on the datasets that were reported in the references. For simplicity and consistency we cite the numbers from the references, if exist, otherwise, we report our reproduced results with the tuned best hyperparameters.

Training & Testing Protocols. In our implementation, we utilize ReLU as our activation functions, as we observed that this activation works better than others such as $\tanh$ in terms of both accuracy and convergence. This observation is consistent with the state-of-the-art methods such as iRNN and FastRNN. In both DIRNN and TinyRNN, we set $\gamma_{l,t} = \gamma_l, \rho_{l,t} = \rho_l, \eta^k_{l,t} = \eta_l$ over different time steps and iterations in the ODE solver. Following FastRNN and iRNN for a fair comparison, we replicate the same benchmark training/testing split with 20% of training data for validation to tune hyperparameters. Then we retrain the models using best tuned hyperparameters using the full training set and test them on the test set. We report our results over three trials with randomization wherever needed. All the experiments were run on an Nvidia Tesla P40 GPU with CUDA10 on a machine with Intel Xeon@2.60GHz CPU with 20 cores.

Hyperparameters. We use the grid search to fine-tune the hyperparameters of each baseline as well as ours on the validation datasets whenever is necessary. Table I lists all the hyperparameters for DIRNN and TinyRNN architectures on different datasets, where we follow [58; 27; 24] to set the hidden feature dimension D, and fine-tune the numbers of layers in depth L, and update times in iRNN K, respectively. The batch size of 128 seems to work well across all the datasets for all the methods. Adam [67] is used as the optimizer for all the methods. The learning rate is always initialized to 10^{-3} with linear scheduling of weight decay.

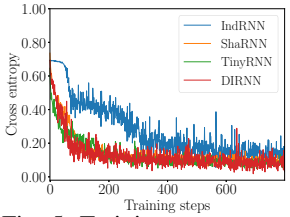


Fig. 5: Training curve comparison on HAR-2.

A. Ablation Study on HAR-2

We first show the training convergence in Fig. 5 that has similar behavior to Fig. 4(b) for the adding task. This demonstrates that the training stability can be generalized on different datasets. Then we illustrate the performance of our method under different (D, L, K)-hyperparameter settings in Fig. 6, where we can see that:

- The deep architectures contribute the most to the performance of both architectures;
- $D = 128$ is already sufficient for good accuracy with a small model size;
- Among all settings, we can observe a performance drop from $L = 5$ to $L = 6$;
- It seems that small K's can already produce good accuracy that leads to good computational efficiency as well.

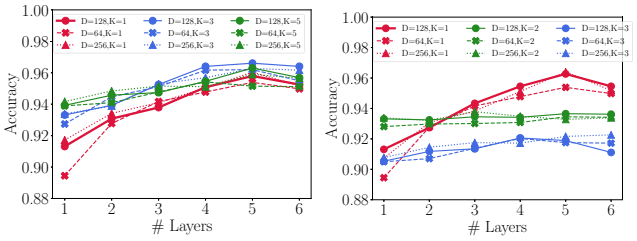
B. State-of-the-art Performance Comparison

Benchmark Vision Task. Fig. 7(a-b) illustrate our performance comparison. The y-axis is the test accuracy of different models and the x-axis is the number of parameters in the log scale. As we see, DIRNN achieves the best on both datasets, indicating the advantages of deep RNN structures. Compared with it, the results of TinyRNN are lower by $1\% \sim 2\%$ with a tiny fraction of parameters. This compact design in TinyRNN can provide us the capability of processing larger data orrunning multiple models in parallel in real-world applications.

[2]https://github.com/KurochkinAlexey/AntisymmetricRNN
[3]https://github.com/microsoft/EdgeML
[4]https://github.com/Sunnydreamrain/IndRNN_pytorch
[5]https://github.com/dtake1336/ERNN-for-speech-enhancement
[6]https://github.com/erichson/LipschitzRNN
[7]https://github.com/minhtannguyen/MomentumRNN
[8]https://github.com/mightydeveloper/Deep-Compression-PyTorch

TABLE II: Result comparison of quantization. Our results are over three trials with networks being trained from the scratch.

Algorithms	HAR-2		DSA-19		PTB	
	Acc.(%)	Model (KB)	Acc.(%)	Model (KB)	Perplexity	Model (KB)
RNN	91.31	29	71.68	20	144.71	129
LSTM	93.65	74	84.84	526	117.41	2052
FastRNN	94.50	29	84.14	97	127.76	513
FastGRNN	95.59	3	83.73	3.25	116.11	39
Antisymmetric	93.15	29	85.37	32	116.87	45
iRNN	96.30	18	88.11	19	115.71	288
ShaRNN	95.40	29	87.36	40	116.14	130
IndRNN	95.35	139	86.93	93	116.02	653
LipschitzRNN	95.32	35	87.51	260	115.36	578
MomentumRNN	95.28	29	87.13	20	115.87	130
DIRNN	**96.53**±0.18	174	**88.38**±0.16	124	**115.35**±0.35	775
TinyRNN	96.17±0.24	8	86.04±0.17	12	115.81±0.45	45
TinyRNN+Q	95.73±0.32	**0.99**	85.65±0.18	**1.4**	116.09±0.43	**20**



(a) DIRNN (b) TinyRNN

Fig. 6: Ablation study on the HAR-2 dataset.

Noise Padded Vision Task. This task is specifically designed for evaluating the capability of RNNs to capture the long term dependency (LTD) among the data. Fig. 7(c-d) illustrate our comparison results. On Noisy-MNIST, DIRNN achieves the best accuracy, while on Noisy-CIFAR LipschitzRNN achieves the best and DIRNN is the second. TinyRNN achieves only ∼1% accuracy loss with ∼3x smaller architecture compared with LipschitzRNN. Surprisingly, ShaRNN works as poorly as vanilla RNNs on both datasets. We hypothesize that this comes from the network design choice where the two-layer architecture with data splitting actually breaks the LTD among the data, although this may not occur on other datasets.

Human Activity Recognition Task. We illustrate our comparison results in Fig. 7(e-f). Unsurprisingly, our DIRNN achieves the best on HAR-2 with an accuracy of 96.53%. And on DSA-19, the DIRNN architecture also achieves the best accuracy. Besides, our TinyRNN achieves the second best on HAR-2 with the smallest number of parameters. This is probably because the data is relatively simple with low-dimensional features and fewer classes where models with small complexity can handle.

Model Size Comparison. Table II lists the test accuracy and model sizes (*i.e.,* the storage on the hard drive, including the linear classifiers) of different competitors. *TinyRNN+Q* indicates the method where the quantization method in FastGRNN is applied to further reduce the sizes of learned models by TinyRNN. As we see, DIRNN works the best in terms of accuracy, and TinyRNN+Q performs slightly worse but with smaller models by two orders of magnitude. This indicates that the proposed tiny architecture can achieve a good balance between model size and performance.

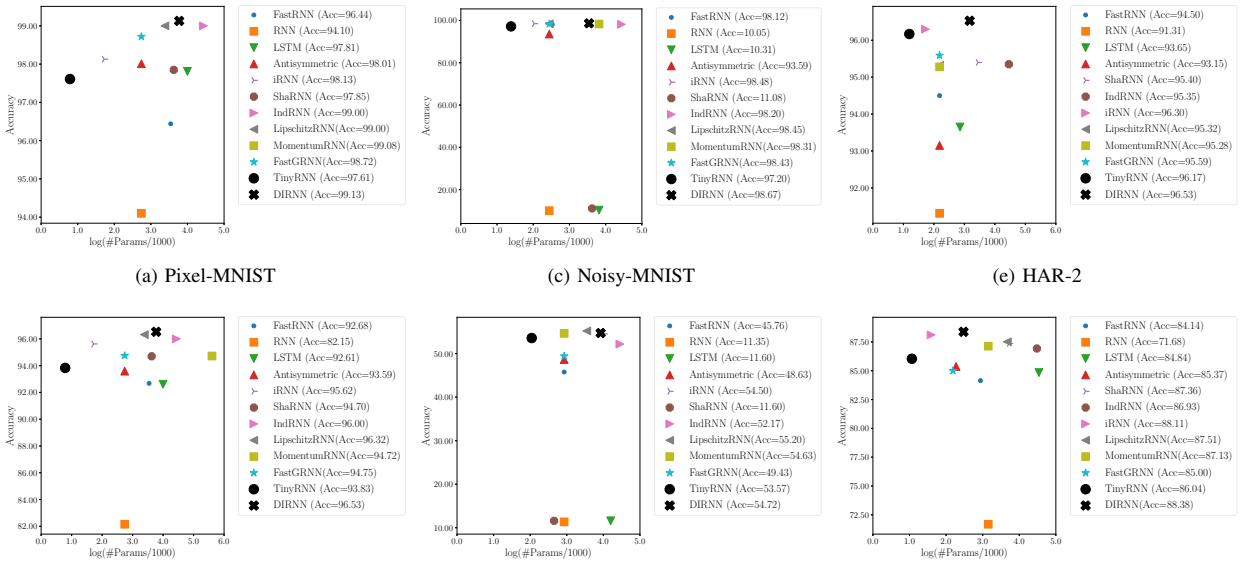
Training & Inference Time. Such numbers are highly dependent on the network architectures of DIRNN as well as the data. For instance, on HAR-2 and DSA-19, the training time is 103s and 100s per epoch and the inference time is 0.46ms and 0.35ms per data sequence on an Nvidia Tesla P40 GPU.

C. Case Study

We use real world cases from DSA-19 dataset to demonstrate the advantage of the proposed DiRNN model. The DSA-19 dataset contains 19 different human activities and some activities have similar patterns which sometimes makes it hard to distinguish those similar activities. For example, both jumping and playing basketball are categories of those 19 activities. However, jumping is a common activity when playing basketball. In Fig. 8, we show the acceleration sensor data of users' body movement when they perform playing basketball and jumping, respectively. From the Fig. 8 (b), we can observe periodic fluctuations of the acceleration in X Axis, which indicates the body ascending and descending in the air. However, when a user plays basketball, the jumping sequence becomes noisy, since the player usually completes more complex movements with the basketball such like dribbling or shooting. In our experiments, we notice that the traditional RNN model usually cannot tell the difference between jumping and playing basketball, we believe it is because that the vanilla RNN model only detects the jumping patterns in the basketball sequence and treat other irregular patterns as noise. However, with our proposed DiRNN model, we can successfully classify the jumping and playing basketball activities. This indicates that the proposed DiRNN model can better learn the sequential patterns in the series data.

VI. Conclusion

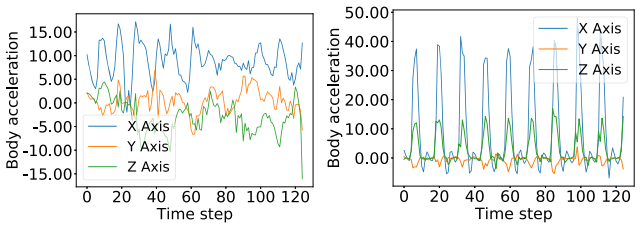
In this paper, we study the problem of training stability in deep RNNs. We propose a novel deep incremental RNN (DIRNN) that has skip connections along both dimensions of time steps as well as network depth. Inspired by recent works such as iRNN, we propose a novel ODE based formulation for



(a) Pixel-MNIST (c) Noisy-MNIST (e) HAR-2

(b) Perm-MNIST (d) Noisy-CIFAR (f) DSA-19

Fig. 7: Test accuracy comparison. To better view the differences between different RNN architectures, by following some recent papers such as FastGRNN and iRNN, here the numbers of parameters exclude those for linear classifiers that are identical for all the competitors.



(a) Playing basketball (b) Jumping

Fig. 8: Two kinds of activities in DSA-19 dataset.

DIRNN that can be solved efficiently using Euler's method. We prove that DIRNN is a Lyapunov stable dynamical system where there is no vanishing/exploding gradient in training, and thus leading to good training stability. To the best of our knowledge, we are the first in the literature to provide such theoretical results on the training stability for deep RNNs. To address the model complexity issue in DIRNN, we also propose a novel implementation, namely TinyRNN, where the transition matrices are sparsified using weighted random permutation matrices to reduce the number of parameters in the network. The learned models can be further compressed using other techniques such as network pruning. We evaluate both RNN models on five different tasks that involve seven benchmark datasets and ten baseline algorithms. Our DIRNN can achieve state-of-the-art accuracy and TinyRNN (with pruning) can achieve the best trade-off between accuracy and model size. Note that our theoretical results can be valuable to analyze the training stability of other networks such as ResNet [68]. In future work, we will explore more for deep learning from the perspective of dynamical systems.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," in *Advances in Neural Information Processing Systems*, 2018.

[3] K. Altun, B. Barshan, and O. Tunçel, "Comparative study on classifying human activities with miniature inertial and magnetic sensors," *Pattern Recognition*, vol. 43, no. 10, pp. 3605–3620, 2010.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[6] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and Trainability in Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1611.09913, Nov. 2016.

[7] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *International Conference on Machine Learning*, 2016, pp. 1120–1128.

[8] L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić, "Tunable efficient unitary neural networks (eunn) and their application to rnns," in *International Conference on Machine Learning*, 2017, pp. 1733–1741.

[9] J. Zhang, Q. Lei, and I. S. Dhillon, "Stabilizing gradients for deep neural networks via efficient svd parameterization," in *ICML*, 2018.

[10] Z. Mhammedi, A. D. Hellicar, A. Rahman, and J. Bailey, "Efficient orthogonal parametrisation of recurrent neural networks using householder reflections," *CoRR*, vol. abs/1612.00188, 2016. [Online]. Available: http://arxiv.org/abs/1612.00188

[11] J. Pennington, S. Schoenholz, and S. Ganguli, "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 4785–4795.

[12] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[13] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," in *ICML*. JMLR. org, 2017, pp. 4189–4198.

[14] A. Mujika, F. Meier, and A. Steger, "Fast-slow recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 5915–5924.

[15] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," *CoRR*, vol. abs/1611.01576, 2016. [Online]. Available: http://arxiv.org/abs/1611.01576

[16] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, "Simple recurrent units for highly parallelizable recurrence," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[17] D. Balduzzi and M. Ghifary, "Strongly-typed recurrent neural networks," *arXiv preprint arXiv:1602.02218*, 2016.

[18] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural networks : the official journal of the International Neural Network Society*, vol. 20, pp. 335–52, 05 2007.

[19] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628, 2013.

[20] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang, "Dilated recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 77–87.

[21] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," *arXiv preprint arXiv:1708.06834*, 2017.

[22] S. S. Talathi and A. Vartak, "Improving performance of recurrent neural network with relu nonlinearity," *arXiv preprint arXiv:1511.03771*, 2015.

[23] M. Y. Niu, L. Horesh, and I. Chuang, "Recurrent neural networks in the eye of differential equations," *arXiv preprint arXiv:1904.12933*, 2019.

[24] B. Chang, M. Chen, E. Haber, and E. H. Chi, "AntisymmetricRNN: A dynamical system view on recurrent neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryxepo0cFX

[25] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, 2018, pp. 6571–6583.

[26] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud, "Latent odes for irregularly-sampled time series," *CoRR*, vol. abs/1907.03907, 2019. [Online]. Available: http://arxiv.org/abs/1907.03907

[27] A. Kag, Z. Zhang, and V. Saligrama, "Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients?" in *International Conference on Learning Representations*, 2020.

[28] F. Rosenblatt, *Principles of neurodynamics.* Spartan Books, Washington, D.C., 1962.

[29] N. B. Erichson, O. Azencot, A. Queiruga, and M. W. Mahoney, "Lipschitz recurrent neural networks," *arXiv preprint arXiv:2006.12070*, 2020.

[30] T. M. Nguyen, R. G. Baraniuk, A. L. Bertozzi, S. J. Osher, and B. Wang, "Momentumrnn: Integrating momentum into recurrent neural networks," *arXiv preprint arXiv:2006.06919*, 2020.

[31] G. Kerg, K. Goyette, M. P. Touzel, G. Gidel, E. Vorontsov, Y. Bengio, and G. Lajoie, "Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 613–13 623.

[32] M. Lezcano-Casado and D. Martınez-Rubio, "Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group," in *International Conference on Machine Learning*, 2019, pp. 3794–3803.

[33] A. H. Ribeiro, K. Tiels, L. A. Aguirre, and T. Schön, "Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2370–2380.

[34] R. Engelken, F. Wolf, and L. Abbott, "Lyapunov spectra of chaotic recurrent neural networks," *arXiv preprint arXiv:2006.02427*, 2020.

[35] R. Vogt, M. P. Touzel, E. Shlizerman, and G. Lajoie, "On lyapunov exponents for rnns: Understanding information propagation using dynamical systems tools," *arXiv preprint arXiv:2006.14123*, 2020.

[36] J. Drgona, E. Skomski, S. Vasisht, A. Tuor, and D. Vrabie, "Spectral analysis and stability of deep neural dynamics," *arXiv preprint arXiv:2011.13492*, 2020.

[37] A. Tuor, J. Drgona, and D. Vrabie, "Constrained neural ordinary differential equations with stability guarantees," *arXiv preprint arXiv:2004.10883*, 2020.

[38] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[39] D. Dennis, D. A. E. Acar, V. Mandikal, V. S. Sadasivan, V. Saligrama, H. V. Simhadri, and P. Jain, "Shallow rnn: accurate time-series classification on resource constrained devices," in *Advances in Neural Information Processing Systems*, 2019, pp. 12 896–12 906.

[40] K. Helfrich, D. Willmott, and Q. Ye, "Orthogonal recurrent neural networks with scaled cayley transform," in *International Conference on Machine Learning*, 2018, pp. 1969–1978.

[41] K. D. Maduranga, K. E. Helfrich, and Q. Ye, "Complex unitary recurrent neural networks using scaled cayley transform," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4528–4535.

[42] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[43] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[44] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[45] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Advances in neural information processing systems*, 2013, pp. 190–198.

[46] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (indrnn): Building a longer and deeper rnn," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5457–5466.

[47] W.-Y. Chen, Y.-F. Liao, and S.-H. Chen, "Speech recognition with hierarchical recurrent neural networks," *Pattern Recognition*, vol. 28, no. 6, pp. 795–805, 1995.

[48] S. El Hihi and Y. Bengio, "Hierarchical recurrent neural networks for long-term dependencies," in *Advances in neural information processing systems*, 1996, pp. 493–499.

[49] S. Fernández, A. Graves, and J. Schmidhuber, "Sequence labelling in structured domains with hierarchical recurrent neural networks," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, 2007.

[50] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992.

[51] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[52] H. Jaeger, "Discovering multiscale dynamical features with hierarchical echo state networks," Jacobs University Bremen, Tech. Rep., 2007.

[53] P. H. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *31st International Conference on Machine Learning (ICML)*, 2014.

[54] W. Liu, P. Guo, and L. Ye, "A low-delay lightweight recurrent neural network (llrnn) for rotating machinery fault diagnosis," *Sensors*, vol. 19, no. 14, p. 3109, 2019.

[55] I. Korvigo, M. Holmatov, A. Zaikovskii, and M. Skoblov, "Putting hands to rest: efficient deep cnn-rnn architecture for chemical named entity recognition with no hand-crafted rules," *Journal of cheminformatics*, vol. 10, no. 1, pp. 1–10, 2018.

[56] D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu, "Delta networks for optimized recurrent network computation," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2584–2593.

[57] D. Roy, S. Srivastava, P. Jain, A. Kusupati, M. Varma, and A. Arora, "Lightweight deep rnns for radar classification," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2019, pp. 360–361.

[58] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network," in *Advances in Neural Information Processing Systems*, 2018, pp. 9017–9028.

[59] U. Thakker, J. Beu, D. Gope, G. Dasika, and M. Mattina, "Run-time efficient rnn compression for inference on edge devices," *arXiv preprint arXiv:1906.04886*, 2019.

[60] U. Thakker, J. Beu, D. Gope, C. Zhou, I. Fedorov, G. Dasika, and M. Mattina, "Compressing rnns for iot devices by 15-38x using kronecker products," *arXiv preprint arXiv:1906.02876*, 2019.

[61] J. Zhang, X. Wang, D. Li, and Y. Wang, "Dynamically hierarchy revolution: dirnet for compressing recurrent neural network on mobile devices," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 3089–3096.

[62] S. Sastry, "Lyapunov stability theory," in *Nonlinear Systems*. Springer, 1999, pp. 182–234.

[63] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[64] M. Rotman and L. Wolf, "Shuffling recurrent neural networks," *arXiv preprint arXiv:2007.07324*, 2020.

[65] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," *arXiv preprint arXiv:1707.05589*, 2017.

[66] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.