Time Adaptive Recurrent Neural Network

Anil Kag, Venkatesh Saligrama
Department of Electrical and Computer Engineering, Boston University
{anilkag, srv}@bu.edu

Abstract

We propose a learning method that, dynamically modifies the time-constants of the continuous-time counterpart of a vanilla RNN. The time-constants are modified based on the current observation and hidden state. Our proposal overcomes the issues of RNN trainability, by mitigating exploding and vanishing gradient phenomena based on placing novel constraints on the parameter space, and by suppressing noise in inputs based on pondering over informative inputs to strengthen their contribution in the hidden state. As a result, our method is computationally efficient overcoming overheads of many existing methods that also attempt to improve RNN training. Our RNNs, despite being simpler and having light memory footprint, shows competitive performance against standard LSTMs and baseline RNN models on many benchmark datasets including those that require long-term memory.

1. Introduction

We focus on trainability of vanilla Recurrent Neural Networks[1] (RNN). Improving vanilla RNN performance is important since they are deployed in a number of IoT applications [15] due to their light memory footprint. A fundamental challenge is that, during training, the gradient of loss back-propagated in time could suffer from exponential decay/explosion resulting in poor generalization for processes exhibiting long-term dependencies (LTD).

There has been a long-line of work such as [12, 21, 2, 31, 10] that propose matrix designs, gating and novel architectures, to mitigate gradient explosion/decay, and improve handling of state-transition. Different from these are works, which go back to [45, 18] that draw inspiration from ordinary differential equations (ODEs). [10] leverages stability theory of ODEs, to identify new transition matrices, and proposes discretization of ODEs, to improve trainability.

While we also draw upon ODEs to propose solutions to improve vanilla RNN trainability, our proposal differs from existing works in fundamental ways. To build intuition, first consider the ODE, with $\lambda \in \mathbb{R}^+$, $\mathbf{U} \in \mathbb{R}^{D\times D}$, $\mathbf{W} \in \mathbb{R}^{D\times d}$, and $\mathbf{A} \in \mathbb{R}^{D\times D}$ Hurwitz stable [28]:

$$\lambda \dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{x}_m) \tag{1}$$

where, $\phi(\cdot)$ is the conventional non-linear RNN activation function such as a ReLU; This particular form, serving as an analogue[2] of vanilla RNNs, is quite old [45]. In each round, m, we start from an initial state, $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$, which corresponds to the current hidden state, and input, $\mathbf{x}_m$, and evolve the ODE for a unit period of time. Subsequently, the hidden state is updated by setting $\mathbf{s}_m = \mathbf{z}(t_0 + 1)$, and in this way, mapping inputs to the hidden state sequence.

What is new? We introduce two novel aspects within this context. First, we allow for λ to be time-varying, and in particular, a function of previous hidden state and input. Our reasoning is that λ serves as a time-constant, and inherently accounts for how long we evolve the ODE in response to the current input. To see this, let us write the ODE in integral form for a fixed λ:

$$\mathbf{s}_m \triangleq \mathbf{z}(t_0+1) = \exp\left(A\frac{1}{\lambda}\right)\mathbf{s}_{m-1} + \frac{1}{\lambda}\int_0^1 \exp\left(A\frac{1-t}{\lambda}\right)\phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{x}_m)dt \tag{2}$$

Then, with $\lambda \to \infty$, we deduce that, $\mathbf{z}(t_0 + 1) \to \mathbf{s}_{m-1}$. Namely, when time constant is large relative to integration time, we barely process the new input, remaining essentially at our previous solution. Alternatively, if $\lambda \to 0$, namely, when the integration time is large relative to the time-constant, we reach equilibrium, and in this process strengthen influence of the current input. Moreover, by letting the time-constant be a function, of $\mathbf{s}_{m-1}, \mathbf{x}_m$, we selectively adapt the amount of "pondering" that we need on each new input. Finally, we let $\lambda(\cdot)$ take values in $\mathbf{R}^D$, and thus allow for element-wise dependence for each hidden state, leading to selective updates of hidden state components. These ideas result in a time-adaptive RNN (TARNN).

[1] By vanilla RNNs we refer to networks that sequentially update their hidden state by means of a simple linear transformation of the previous state and current input, followed by non-linear activation.

[2] Vanilla RNNs and residual variants amount to a suitable Euler discretization (see Appendix).

Next, we augment the current input with the hidden state, and consider $\mathbf{u}_m = [\mathbf{x}_m, \mathbf{s}_{m-1}]^\top$ as a composite input in our ODE with initial condition, $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$:

$$\lambda(\mathbf{u}_m) \circ \dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m) \quad (3)$$

where $\circ$ represents the element-wise (Hadamard) product. To build intuition into our ODE choice, observe from the first term in Eq. 2 that for $\mathbf{A}$ stable, the contribution of the hidden state, $\mathbf{s}_{m-1}$ decays exponentially in time, and as such, the discrete transition process, $\mathbf{s}_1, \ldots, \mathbf{s}_T$ rapidly de-correlates. We can overcome this effect by a persistent presence of the hidden state in the ODE. We also add the linear term, $\mathbf{B}\mathbf{u}_m$, as it turns out to be important for improving partial gradient properties for hidden state sequence. As such our choice does not significantly increase model complexity of vanilla RNN.

Our proposed ODE is sufficiently rich admitting parameter settings that completely eliminate gradient decay and explosion, which is desirable for LTD tasks. In addition, our method is capable of enhancing contribution from informative inputs, while suppressing noisy segments through the pondering mechanism described above. This aspect is useful in IoT applications [31, 15] such as keyword detection and wearable sensing devices.

Discretization: For simplicity we discretize our ODEs with Euler discretization to realize vanilla RNNs. Methods that seek computational and memory efficiency in this context [11, 46] are entirely complementary to our method. Our novelty is in the design of state-transition with the goal of realizing desirable ODE solutions[3].

Contributions: The main contributions of this work are

- TARNN learns to modulate time constants of transition function, allowing for selectively pondering on informative inputs to strengthen their contribution, and ignoring noisy inputs. This modification along with designing suitable transition matrices yield lossless information propagation.
- TARNN improves trainability leading to better handling of LTD tasks with a lighter memory footprint, and as such our proposed method can be leveraged for IoT tasks.
- Our pseudo code is an RNN cell that is readily deployable in any deep learning library. We provide a simple implementation at https://github.com/anilkagak2/TARNN.
- We conduct extensive experiments on benchmark datasets, and show that we improve upon standard LSTM performance as well as other recently proposed works. We also demonstrate robustness to time-series distortions such as noise paddings.

[3] [11, 46], also propose recurrent models to handle non-uniform input sampling. While this is interesting, their proposals are unrelated to our goal of improving RNN trainability.

2. Related Work

There is a rich literature on deep RNNs [54, 6, 43], which incorporate deep non-linear transition functions for complex and richer representation, and is outside the scope of our work. Indeed, our work is complementary, and we seek to improve vanilla RNN trainability. More loosely related are a growing number of works that propose to improve RNN trainability through mitigation of vanishing and exploding gradients. First, there are works that propose improving state-transitions, based on unitary transition matrices [2, 25, 53, 42, 27, 34], residual connections [23, 6, 31] or gating [21, 12]. While these methods provide some evidence of mitigating gradient decay, in practice, and in theory, vanishing gradients are not eliminated (see Appendix). Different from these works, our method is more closely related to works that draw insights from ODEs.

ODE inspired RNNs. [10, 16] and [48] draw upon insights from linear system theory to guide transition matrix designs for the discrete-time RNN. Ideally, in the regime where non-linear activation is essentially linear, explosion/decay can be eliminated, but outside this regime we can expect gradient degradation. [26] propose Incremental-RNNs, a novel architecture, where like us they evolve the system until equilibrium, and show mitigation of vanishing/exploding gradients.

Different from these efforts, we are motivated by the observation that mitigating gradient degradation while important, is by no means sufficient (see Fig. 1). This is often the case in many IoT applications where the signal can be bursty and there are segments that can be purely noisy. We propose methods to suppress noisy segments in addition to improving gradient explosion/decay.

Conditional Computation and Attention. Our pondering perspective can be viewed as a form of conditional computation in time. Nevertheless, much of the conditional computation work is aimed at gradually scaling model capacity without suffering proportional increases in computational (inference) cost (see [19, 13, 52, 24, 20]). Different from these works, our focus is on improving RNN trainability by suppressing noisy observations, so that long-term dependencies can be handled by ignoring uninformative input segments. Within this context, only [9] is closely related to our viewpoint. Like us, [9] also proposes to skip input segment to improve RNN training, but unlike us, since their state-transition designs are conventional, they still suffer vanishing and exploding gradients on segments that are not skipped, and as a result suffer performance degradation on benchmark datasets. Also, as [9] points out, our work can also be viewed as a temporal version of hard attention mechanisms for selecting image regions. These works (see [9]) that deal with visually-based sequential tasks, have high model-complexity, and are difficult to train on long input sequences. Others [49] leverage attention to bypass RNNs. In contrast,

we offer an approach that is lightweight and improves RNN trainability on long-sequences.

There have been attempts at improving the computational cost of the sequential models by introducing lighter recurrent connections. [33, 4] replace the hidden-to-hidden interactions in the LSTMs with linear connections in the hope of parallelism. [7] performs similar linear interactions along with the increased receptive field from the inputs, i.e. instead of just using the current observation, it uses previous few inputs as well to compensate for the lost non-linear interaction between the hidden states. Similar to these works, [36, 35] introduce linear connection in vanilla RNNs and compensate the loss of performance by allowing various architectures on top of this light recurrent unit. These architectures include stacked encoders, residual, and dense connections between multiple layers. It should be noted that although being light, the loss of non-linear interaction does result in a significant setback and as a result these works have to rely on more than one RNN layer to gain anything reasonable in comparison to traditional variants. These multi-layered models will be prohibitive for IoT devices as inference time would be larger than vanilla RNNs. Besides, we can extend our work by allowing only linear connections and apply their orthogonal ideas for better parallelism and computational speed.

3. Learning Time Adaptive Recurrent Neural Network (TARNN)

In this section we further present our objective, ODE discretization and algorithmic details.

Notation. $\{(\mathbf{u}^{(i)}, \mathbf{y}^{(i)})\},\ i \in [N]$ denotes training data. Each $\mathbf{u}^{(i)}$ is a $T-$length $d-$dimensional sequential input. For classification problems, $\mathbf{y}^{(i)}$ is a terminal label $y_T^{(i)}$, taking values in a discrete set of C classes. For language modeling tasks, we let the true label be a process, $(y_1^{(i)}, \ldots, y_T^{(i)})$. The predictions $(\hat{y}_1^{(i)}, \ldots, \hat{y}_T^{(i)})$ for each input $\mathbf{u}^{(i)}$ can be computed from the $D-$dimensional hidden states $(\mathbf{s}_1^{(i)}, \ldots, \mathbf{s}_T^{(i)})$ obtained by solving the ODE Eq. 3. When clear from the context we omit superscripts. Unless stated otherwise, $\sigma(\cdot)$ denotes the sigmoid activation; $\phi(\cdot)$ refers to any non-linear activation such as a ReLU. We collect all model parameters in θ.

Empirical Risk Minimization. Let $\ell(\hat{y}, y)$ be the function measuring loss incurred for predicting value $\hat{y}$ on the true value y. Our objective is to minimize the regularized empirical loss, through back-propagation in any deep learning framework. We specify the regularizer $\Omega(\theta)$ later.

$$L(\{\mathbf{u}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N) = \frac{1}{N}\frac{1}{T}\sum_{i=1}^{N}\sum_{m=1}^{T}\ell(\hat{y}_m^i, y_m^i) + \Omega(\theta) \tag{4}$$

Time-constants. We re-write the ODE Eq. 3 in terms of $\boldsymbol{\beta}(\cdot)$, the inverse of $\boldsymbol{\lambda}(\cdot)$, since it is convenient for describing our discretization steps. We parameterize $\boldsymbol{\beta}(\mathbf{u}_m) = \sigma(\mathbf{U}_s\mathbf{s}_{m-1} + \mathbf{W}_x\mathbf{x}_m)$, where $\mathbf{U}_s \in \mathbb{R}^{D\times D}, \mathbf{W}_x \in \mathbb{R}^{D\times d}$ are parameters to be learnt. For a component j where $\boldsymbol{\beta}_j \approx 1$, then $(\dot{\mathbf{z}}(t))_j \approx (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))_j$, and the system responds to the input $\mathbf{u}_m$ and reaches equilibrium. On the other hand, when $\boldsymbol{\beta}_j \approx 0$, then $(\dot{\mathbf{z}}(t))_j \approx 0$, and the corresponding state is frozen, with the input at time m completely skipped. In this paper we limit ourselves to a binary behavior, i.e. whether to ponder over the input observation for a long time or not ponder at all. For this reason, it suffices to limit the range in $[0, 1]$ with sigmoid activation. This also avoids numerical instabilities with unbounded non-linearities.

Setting up the ODE. To obtain a discrete implementation, first, we update the ODE Eq. 3 with the change of variables for time-constants, resulting in the ODE:

$$\begin{aligned}\dot{\mathbf{z}}(t) =& \boldsymbol{\beta} \odot (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m)) \\ \triangleq& F(\mathbf{z}(t), \mathbf{u}_m);\ \ \mathbf{z}(t_0) = \mathbf{s}_{m-1}\end{aligned} \tag{5}$$

where, $\odot$ represents the Hadamard product. Next, we instantiate the specific parameterization for transition matrices. Finally, an ODE solver is invoked, over a time-horizon $[t_0, t_1]$ to update the state:

$$\mathbf{s}_m = \mathbf{z}(t_1);\ \ \mathbf{z}(t_1) = \text{ODESolve}(\mathbf{s}_{m-1}, \mathbf{u}_m, F(\cdot), t_0, t_1)$$

We predict the output $\hat{y}_m = \sigma(\mathbf{w}^\top \mathbf{s}_m + b)$ using a sigmoid activation on top of a linear layer parameterized as $(\mathbf{w}, b)$. Since, we need $\mathbf{A}$ to be Hurwitz-stable, and we impose equilibrium, when a component is active, we a priori fix $\mathbf{A}$ as negative identity. Other TARNN model parameters $(\mathbf{B}, \mathbf{U}, \mathbf{W}, \mathbf{w}, b, \mathbf{U}_s, \mathbf{W}_x)$ are learnt during training by minimizing the empirical loss in Eq. 4.

The ODE solver. A number of methods exists to numerically solve the ODE of Eq. 5 including black-box solvers such as Neural ODEs[11] or advanced root-finding methods such as the Broyden's method [8]. While these methods could be further employed to improve computational efficiency, for exposition we limit ourselves to Euler-recursion with $K = 3$ steps, since computational efficiency as such is not the focus of our paper. We let η denote the step-size, with $\mathbf{z}_m^k$ denoting the recursion steps:

$$\begin{aligned}\mathbf{z}_m^k =& \begin{cases} \mathbf{s}_{m-1} & \text{if}\ \ k = 1 \\ \mathbf{z}_m^{k-1} + \eta(F(\mathbf{z}_m^{k-1}, \mathbf{u}_m)) & \text{if}\ 1 < k < K \end{cases} \\ \mathbf{s}_m =& \ \mathbf{z}_m^K\end{aligned} \tag{6}$$

As shown in the Sec. 3.1, for suitable choice of the activation function, $\phi(\cdot)$, (includes popular activations such as ReLU, tanh, sigmoid, etc.), these recursions in the limit, for $(\boldsymbol{\beta})_j > 0$, $\mathbf{z}_m^* = \lim_{k\to\infty}\mathbf{z}_m^k$ is an equilibrium solution to the ODE of Eq. 5. We provide the pseudo code in Algorithm 1, which generates the hidden states for a sequential input $\{x_m\}_{m=1}^T$.

Algorithm 1 TARNN hidden states computation

Input : Sequence $\{x_m\}_{m=1}^{T}$
Model : $(\mathbf{A}, \mathbf{U}, \mathbf{W}, \mathbf{U}_s, \mathbf{W}_s, \mathbf{B})$
Initialize hidden state $\mathbf{s}_0 = 0$
for $m = 1$ **to** T **do**
  $\boldsymbol{\beta} = \sigma(\mathbf{U}_s \mathbf{s}_{m-1} + \mathbf{W}_x \mathbf{x}_m)$
  $F(\cdot) = \boldsymbol{\beta} \odot (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))$
  $\mathbf{z}(t_1) = \text{ODESolve}(\, \mathbf{s}_{m-1}, \mathbf{x}_m, F(\cdot), t_0, t_1\,)$
  $\mathbf{s}_m = \mathbf{z}(t_1)$
end for

3.1. Analysis

In this section, we show that our setup benefits from several properties, and as a result, our proposed method leads to a theoretically sound approach for an adaptive recurrent system that is capable of focusing attention on informative inputs and rejecting uninformative inputs. The first few propositions establish properties of TARNN with the proposed parameterization. We then describe a result to assert that our adaptively recurrent system preserves information by showing that the partial gradients of hidden states have unit norm.

The following proposition shows that equilibrium points for the ODE of Eq. 5 exist and are unique. Although, we a priori fix $\mathbf{A}$ to be negative identity, we present a more general result for the sake of completion. We impose the following conditions, (i) there is a $\eta_0 > 0$ such that for all $\eta \in [0, \eta_0]$, there is some $\alpha \in (0, 1]$ such that $\sigma_{\max}(I + \eta A) \leq 1 - \alpha\eta$. (ii) $\lambda_{\max}(\mathbf{A} + \mathbf{A}^\top) < -1$. It is easily verified that these conditions are satisfied in a number of cases including $\mathbf{A}$ -identity, $\mathbf{A}$ block triangular with negative identity blocks.

Proposition 1. *Consider the ODE in Eq. 5 and assumptions on $\mathbf{A}$ described above. Suppose we have $\|U\| < \alpha$, and $\phi(\cdot)$ is 1-Lipshitz function, it follows that, for any given, $\boldsymbol{\beta}$, $\mathbf{u}_m$, an equilibrium point exists and is unique.*

Remark. Note that, we impose conditions on $\mathbf{U}$ to derive our result. In experiments we do not impose this condition, since for our choices for $\mathbf{A}$, $\alpha \approx 1$, and as such, initializing $\mathbf{U}$ to a Gaussian zero-mean, unit covariance often takes care of this requirement during training, since we generally operate with a small learning rate.

Proof Sketch. To show this we must find a solution to the non-linear equation $\mathbf{A}\mathbf{z} + \mathbf{B}\mathbf{u}_m - \phi(\mathbf{U}\mathbf{z} + \mathbf{W}\mathbf{u}_m) = 0$ and show that it is unique. We do this by constructing a fixed-point iterate, and show that the iteration is contractive. The result then follows by invoking the Banach fixed point theorem (contraction-mapping theorem). The proof is presented in the appendix 3.

Proposition 2. *With the setup in Proposition 1, and regardless of $\boldsymbol{\beta}$, the equilibrium point is globally asymptotically stable, and the discrete Euler recursion converges to the equilibrium solution at a linear rate.*

We discuss the main idea and present the proof in the appendix. Let $\mathbf{z}^*$ be the equilibrium solution. We consider the Lyapunov function $V(\mathbf{z}(t)) = \|\mathbf{z}(t) - \mathbf{z}^*\|^2$ and show that it is monotonically decreasing along the ODE system trajectories. Observe that, as per our setup, components where $(\boldsymbol{\beta})_j = 0$ does not pose a problem, because those states remain frozen, and serve as an additional exogenous input in our ODE.

Lossless Information Propagation. Our goal is to show that there exist parameter constraints in Eq. 5 that can result in identity partial gradients of the hidden states. This will in turn inform our regularization objective, $\Omega(\theta)$ later. With the constraint in place, for arbitrary values, m, $n \in \mathbb{Z}^+$, we will show that, $\frac{\partial \mathbf{s}_n(j)}{\partial \mathbf{s}_m(j)} = 1$. For ease of analysis we replace binary-valued $\boldsymbol{\beta}$ with a continuous function and let the output be a ReLU non-linearity. Partition $\mathbf{W} = [\mathbf{W}^1, \mathbf{W}^2]$, $\mathbf{B} = [\mathbf{B}^1, \mathbf{B}^2]$, where $\mathbf{W}^2, \mathbf{B}^2 \in \mathbf{R}^{D \times D}$ are associated with the hidden state components. To realize identity gradients for a specific component i we need to constrain the parameter space. While there are many possibilities, we consider following constraints, because they lead to concrete regularization objectives, and generalize the specific $\mathbf{A}$ matrices we have in mind (identity, and upper-triangular). We constrain $\|\mathbf{U}\| < 1 \leq \|A\|$, and consider the following case: $\mathbf{A} \pm \mathbf{B}^2 = 0$, $\mathbf{U} \pm \mathbf{W}^2 = 0$.

Theorem 1. *Under the above setup, as $K \to \infty$ in Eq. 6, for any $m, n \in \mathbb{Z}^+$, $|\partial \mathbf{s}_n(i) / \partial \mathbf{s}_m(i)| \to 1$.*

Proof Sketch (see Appendix for proof). Note that, when $\beta_j = 0$, the jth component $\mathbf{s}_m(j) = \mathbf{s}_{m-1}(j)$ and the result follows trivially. Suppose now the jth component $(\beta)_j > 0$, we will show that, $\partial \mathbf{s}_m(j) / \partial \mathbf{s}_{m-1}(j) = 1$, which then establishes the result through chain rule.

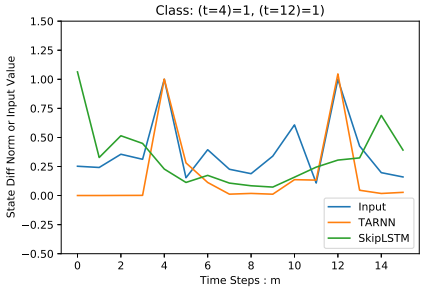
Theorem 1 shows that there is a configuration with lossless propagation. Thus, if it is necessary, the training algorithm will find a solution, that results in lossless propagation, even without imposing parameter constraints stated in the theorem. However, Theorem 1 suggests a natural regularizer, with γ_1 and γ_2 serving as hyperparameters. As a case in point, we could encourage parameters to subscribe to constraints of theorem if we consider the following regularizer for Eq. 4:

$$\Omega(\theta) \triangleq \Omega([\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{W}]) = \gamma_1 \|\mathbf{A} + \mathbf{B}_2\|_2^2 + \gamma_2 \|\mathbf{U} + \mathbf{W}_2\|_2^2$$

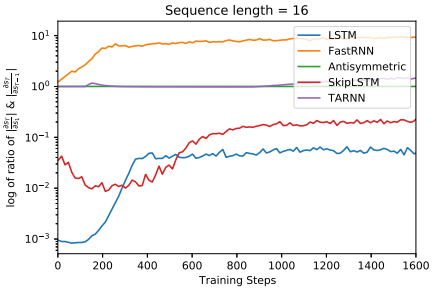
An interesting case is when B^2 row-wise sparse. In this case, states corresponding to zero rows operate as standard RNN (no linear term). We can ensure identity gradient holds in this case with block-wise parametric constraints, leading to more structured regularization penalty.

4. Experiments

Toy Example. For a sneak preview of our results, we illustrate the importance of both time-constants and gradient



(a)



(b)

Algorithm	Accuracy
Random Guess	25
FastRNN	45
LSTM	45
Antisymmetric	37
SkipLSTM	60
TARNN (Ours)	**100**

(c)

Figure 1: Example illustrates importance of mitigating gradient explosion/decay as well as ignoring noisy observations. Table lists test performance of baselines focused on improving RNN training. Fig. (a) plots the noisy input, and sequential changes in hidden state norms for SkipLSTM[9] and proposed TARNN . Only ours responds to informative locations. Fig. (b) plots the norm of partials of hidden states. Only AntisymmetricRNN[10] and ours TARNN exhibit near identity gradients. However, only ours is effective as seen from the table. As such we infer TARNN (a) realizes near identity gradients for partials of hidden states, thus mitigating gradient explosion/decay, (b) zooms in on informative inputs and ignores noisy observations, and (c) By jointly ensuring (a) and (b), it improves RNN trainability, providing good generalization.

mitigation on a toy example. We construct a 16-length input sequence with 4 class labels. Information is placed in the form of binary $\{0, 1\}$ values at locations $4, 12$, corresponding to the four classes, and for all other locations we assign values from a uniform distribution in the unit interval. RNNs with a 2-dimensional state-space are trained on 50K time-traces. Due to low-dimension, the (terminal) state cannot replicate the entire trace, requiring generalization. On one hand, techniques that mitigate gradient explosion/decay like Antisymmetric [10], do so across all input locations, but fail to output meaningful results as seen from Figure 1(c). Thus focusing solely on vanishing/exploding gradients is not sufficient, since noise gets amplified in latent state updates. On the other hand, SkipLSTM [9], which is capable of pondering at informative inputs and skipping uninformative inputs, is also ineffective. SkipLSTM [9] suffers severe gradient degradation, leading to poor control over which locations to ponder. In contrast, TARNN exhibits near identity gradients, skips all but locations $4, 12$, and achieves 100% accuracy. Similar trend holds for larger state space (see Appendix).

4.1. Experimental Setup and Baselines

4.1.1 Datasets

We follow earlier works [10, 26, 31] in order to setup experiments. Datasets used in this work are publicly available, except NTU RGB+d[47] (skeleton modality is available for academic usage). We use 10% of the training data as validation set for tuning the hyper-parameters through grid search. The grid for each method is setup as per their experimental section. Finally, the entire training set is used to train the model. The performance is reported on the publicly available test set. We briefly describe each dataset here and refer the reader to [26](Appendix:A.4) for detailed description.

1. **Add & Copy tasks [22]** probe the LTD capabilities of any RNN architecture. Add task presents two input sequences, one draws points uniformly from $[0, 1]$, and second is a binary sequence with exactly two entries as 1. The target is the sum of entries in the first sequence with index same as the two 1s in the second sequence. Copy tasks contains sequences that embeds a random message drawn from an alphabet, followed by many 'blank' character and a delimiter. The goal is to remember the original signal across the timesteps.
2. **Pixel & Permute MNIST [32]** are sequential variants of the popular MNIST dataset, used for evaluating the generalization performance of RNNs. Pixel-MNIST is generated by flattening the 28×28 input image into a long 784 length sequence. Permute-MNIST is generated by applying a fixed permutation on the images generated by the Pixel-MNIST task.
3. **Noisy-MNIST & Noisy-CIFAR [10, 26]** evaluate the noise resiliency of RNNs. Popular MNIST/CIFAR images are converted into 1000 length sequences. Each row of the image is embedded at initial timesteps and the remaining ones contain noise.
4. **Penn Tree Bank (PTB)-300 [31, 53]** is a popular word level language modelling dataset. We use the small configuration proposed in [39] that uses one layer for modelling, but with harder sequences of length 300.
5. **Penn Tree Bank (PTB-w, PTB-c) [39]** are two popular datasets widely used for word level and character level language modelling. This uses the traditional sequence length of 70 and 150 for word and character level respectively. Current state-of-the-art results utilize more than one layer. We follow [51] in order to setup the word level task while we utilize [40] to setup the character level task. We use three layers for modelling as recommended.
6. **Skeleton based Action Recognition [47]** is performed on the NTU RGB-d dataset with 60 action classes. We follow [35, 36] in order to create the cross-subject (CS) and cross-view (CV) datasets. After eliminating the spurious entries, CS dataset contains 40,091 train and 16,487 test samples, while CV dataset contains $37,646$ train and 18,932 test

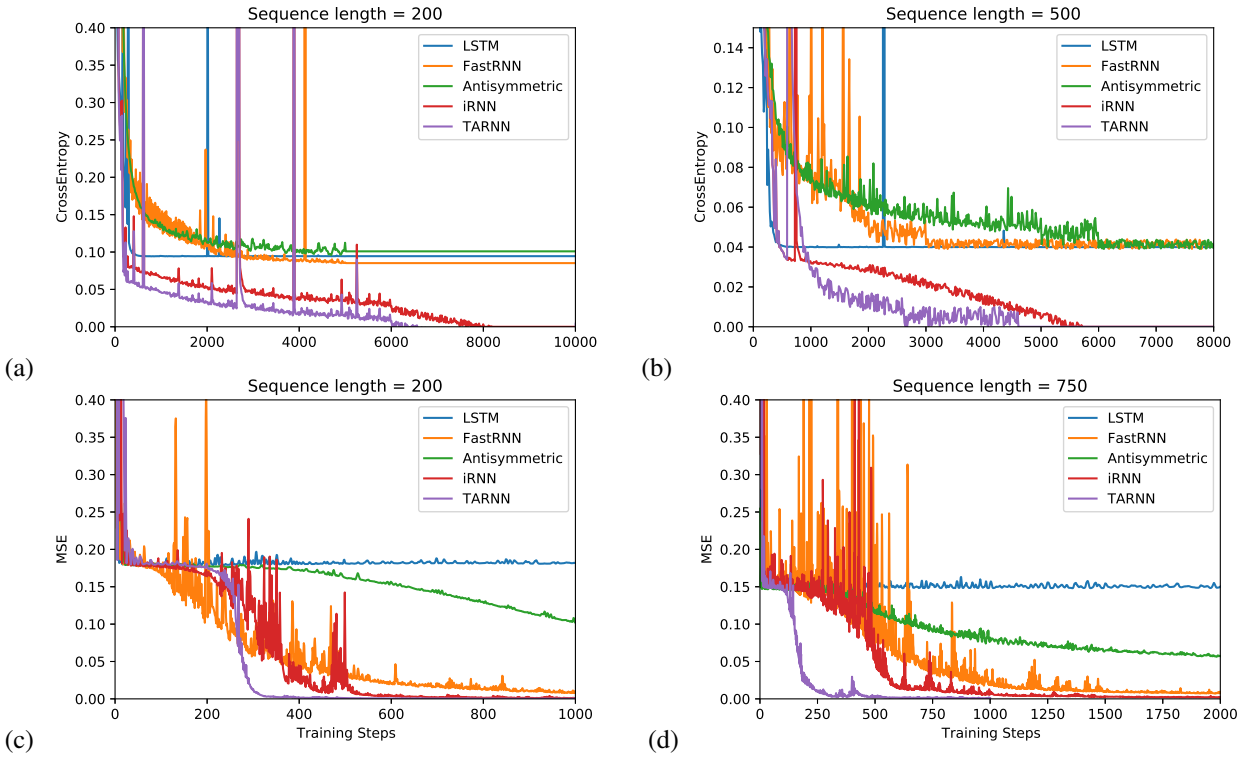


Figure 2: We evaluate TARNN on synthetic LTD tasks: Copy task with sequence lengths : (a) 200, (b) 500, and Add task with sequence lengths: (c) 200, (d) 750. Note that many methods perform similar to a simple fixed baselines described in [26](Appendix:A.4), while TARNN achieves significantly better solution in fewer training steps.

samples. In this dataset, 5% of the train data is used for hyper-parameter selection.

4.1.2 Baselines

We use various state-of-the-art methods for evaluating TARNN 's performance, including popular RNNs such as: gating methods(LSTM[22], FastGRNN[31]), ODE inspired(iRNN [26], AntisymmetricRNN [10]), conditional computation (SkipRNN [9]), as well as recent Unitary/Orthogonal (nnRNN[27] & expRNN[34]). For baselines with gated/ungated variants, we report results for the best of the two. We also tried to incorporate SkipRNNs [9] in our baselines, but for many of our tasks, its performance remained similar to the corresponding RNN variant. Hence, we do not list SkipRNNs on all our experiments. Furthermore, adaptive computation time (ACT) [19] is not tabulated as we found that performance of SkipLSTM is significantly better. This has also been observed in [17], who shows repeat-RNNs, a variant of iRNN outperforms ACT. Note that we do not report baselines [33, 7, 36] which trade-off non-linear hidden-to-hidden connections with linear connections, since these interactions are complementary to our method and can be incorporated in TARNN for computational advantages. Note that the datasets 5 and 6 (PTB-w, PTB-c and Action recognition) are computationally expensive and take days for a single run with standard baselines, hence we do not run baselines on these datasets and simply cite the current best known results. These datasets demonstrate that, (a) TARNN can outperform baselines with smaller models, and (b) since these datasets require stacked or other complex architectures, our experiments show that multi-layered TARNN can be trained with similar ease.

4.1.3 Code & Evaluation Metrics.

We implement TARNN in the tensorflow framework using the pseudo code. Most of the baselines are publicly available except Antisymmetric and Incremental RNNs which provide pseudo code for implementation. For all the methods, we report the accuracy, training time and the model parameters. Unfortunately, we do not report train times for nnRNN and expRNN as their code is written in PyTorch. We use Adam([29]) for minimizing the loss function in Eq. 4. We provide the final hyper-parameters along with the grid values for our experiments in the appendix A.2. Our inference time is comparable to FastRNNs and iRNNs, in contrast LSTMs take 4x longer for inference (see Appendix A.9).

4.2. Results and Discussion

Figure 2 shows the results on Copy and Add tasks. Table 1 reports the performance on Pixel-MNIST, Permute-MNIST,

Table 1: Results for Pixel MNIST, Permuted MNIST, Noise Padded CIFAR-10 and MNIST datasets. Since TARNN effectively focuses on informative segments, it achieves better performance with faster convergence. Note that we only keep baselines which report results with single RNN layer and no batch normalization (this excludes baselines such as [36], [14]).

Dataset	**Pixel-MNIST**				**Permute-MNIST**			
	Hidden Dimension	**Accuracy (%)**	**Train Time (hr)**	**#Params**	**Hidden Dimension**	**Accuracy (%)**	**Train Time (hr)**	**#Params**
FastRNN	128	97.71	16.10	18K	128	92.68	9.32	18K
LSTM	128	97.81	26.57	68K	128	92.61	19.31	68K
SkipLSTM	128	97.31	-	68K	128	93.72	23.31	68K
Antisymmetric	128	98.81	10.34	18K	128	93.59	4.75	18K
expRNN	128	97.35	-	34K	128	94.01	-	34K
nnRNN	128	97.81	-	51K	128	94.29	-	51K
iRNN	128	98.13	2.93	**4K**	128	95.62	2.41	**8K**
TARNN	32	98.43	**2.13**	10K	32	96.21	**1.71**	10K
TARNN	128	**98.93**	3.42	68K	128	**97.13**	2.96	68K
Dataset	**Noisy-MNIST**				**Noisy-CIFAR**			
FastRNN	128	98.12	8.93	11K	128	45.76	11.61	16K
(Skip)LSTM	128	10.21	19.43	82K	128	10.41	13.31	114K
Antisymmetric	128	97.76	5.21	10K	128	54.70	7.48	41K
expRNN	128	97.92	-	37K	128	48.97	-	47K
nnRNN	128	98.06	-	54K	128	49.28	-	63K
iRNN	128	98.48	2.14	**6K**	128	54.50	2.47	**12K**
TARNN	32	98.78	**1.31**	8K	32	57.42	2.01	14K
TARNN	128	**99.03**	1.71	78K	128	**59.06**	**1.05**	100K

Noisy-MNIST and Noisy-CIFAR datasets. These results show that TARNN outperforms various methods on many benchmark LTD tasks, which can be attributed to its near lossless gradient propagation between informative segments. Additionally, tables 2 and 3 report TARNN 's performance on various PTB datasets, and table 4 lists accuracies of all the methods on CS and CV variants of the Skeleton based Action recognition task. These experiments demonstrate that TARNN outperforms many baselines in learning short-term dependencies on language modelling tasks and terminal short term dependency task. Below we present TARNN 's useful properties backed by empirical evaluations.

Table 2: PTB Language Modeling: 1 Layer (standard small config except the sequence length is 300 as per [31] as opposed to 70 in the conventional PTB). TARNN achieves significantly better performance than the baselines on this task (even with half the hidden dimensions than the baselines). Note that embedding size is same as hidden dimension in these experiments, thus smaller hidden dimensions result in smaller embedding storage as well.

Algorithm	Hidden Dimension	Test Perplexity	Train Time (min)	#Params
FastRNN	256	115.92	40.33	131K
LSTM	256	116.86	56.52	524K
SkipLSTM	256	114.23	63.52	524K
iRNN	256	113.38	**34.11**	**100K**
TARNN	128	**102.42**	40.23	114K
TARNN	256	**94.62**	53.16	524K

(A) Fast convergence. Figure 2 shows the convergence plots for various methods on the Add & Copy tasks. It should be observed that TARNN solves both of these tasks significantly faster than the baselines. Due to poor gradient propagation, LSTMs only achieve the performance of fixed strategies. While iRNN solves these two tasks, it requires more training steps to reach the desired target error. Note that we do not show Unitary RNNs on these tasks, as they take significantly longer number of training steps to solve the Addition task, and benefit from the modReLU activation on the copy tasks [26]. Similarly, TARNN trains significantly faster on LTD tasks presented in the Table 1 (at least $8\times$ faster than LSTMs and at least $1.3\times$ faster than the best).

(B) Better generalization. Table 1 shows that TARNN outperforms the baselines resulting in better accuracies on all the terminal prediction tasks. On Noisy-CIFAR dataset, TARNN achieves more than four points increase in accuracy, while on the 300-length PTB language modelling task, we get nearly 20 points better in perplexity than the best method.

(C) Noise resiliency. In order to evaluate TARNN's noise resilience, we conduct experiments on the Noisy-MNIST and Noisy-CIFAR datasets [10, 26] which introduces the informative segments in the first few timesteps and embeds every other segment with noise. These datasets requires both lossless gradient propagation along with the ability to suppress noisy segments and only focus on informative segments. Intuitively we expect to perform better on this task since TARNN selectively ponders on informative segments to strengthen their contribution and allows the state transition to achieve near lossless gradient propagation. Table 1 shows that TARNN achieves much better performance than iRNNs/AntisymmetricRNNs which in turn beat the remaining methods by significant margins.

(D) Adapts well on short-term dependency tasks. We benchmark TARNN on PTB-300 dataset. We do not report expRNN and nnRNN results as they perform poorly in comparison to LSTM [27]. Table 2 reports all the evaluation metrics for the PTB Language modelling task with 1 layer as setup by [31]. It can be clearly seen that TARNN outperforms the baselines by roughly $\approx$ 10 point difference

Table 3: Results for Penn Tree Bank Character and Word level language modelling tasks. These use shorter sequence length (typically 50-150) and use more than one RNN layer for modelling. For the PTB-w dataset, where ever applicable, all the baselines report the results with dynamiceval[30]. Our model uses 3 layer composition. It can be seen that we report reasonable performance with much smaller models than other methods. With comparable model sizes as the baselines we report higher performance.

Dataset	PTB-c			PTB-w		
	Hidden Dimension	BPC	#Params	Hidden Dimension	Perplexity	#Params
(GAM) RHN[54, 38]	600	**1.147**	16M	830	66.0	24M
Trellis-Net [3]	1000	1.158	13.4M	1000	54.19	34M
AWD-LSTM [41, 30]	1000	1.175	13.8M	1150	51.1	24M
Neural Architectural Search [55]	800	1.21	16.3M	800	62.4	54M
IndRNN [36]	2000	1.21	22M	2000	60.21	28M
Residual IndRNN [35]	2000	1.19	50.7M	2000	58.99	57M
Dense IndRNN [35]	2000	1.18	45.7M	2000	**50.97**	52M
TARNN	500	1.29	**7M**	500	60.90	**11M**
TARNN	1400	1.19	42M	1200	53.21	56M

in the test perplexity for similar model complexity while it achieves ≈ 20 points for a larger model. Likewise, TARNN adapts well to other short-term dependency tasks as observed by Table 3 and Table 4.

(E) Low model complexity. Table 1, 2 show TARNN performance with two different hidden state dimensions, namely one configuration with similar model size as iRNN and other one with similar model size as larger RNNs. With model complexity similar to iRNNs, which are much compact than the other baselines, we achieve better performance than iRNNs. With larger model complexity, we achieve much better performance on Permute-MNIST, Noisy-CIFAR and PTB datasets. The other tasks are relatively saturated as almost all the methods are near optimal. We point out that the number of parameters reported in the Table 2 only count the RNN parameters and omit the embeddings. We achieve 102 perplexity with lower hidden dimension, i.e. 128. This means we require less number of parameters for the embedding representation. Similarly, Table 3, 4 compare TARNN's performance on larger multi-layered RNN tasks, namely PTB-c, PTB-w, and Action recognition. It can be seen that TARNN achieves similar performance as known baselines with much smaller model.

Table 4: Results for NTU RGB-d dataset (Skeleton based action recognition). We do not use augmentation on top of the Skeleton data. We point out that TARNN achieves competitive performance with much lower complexity model. We also ran a dense variant of TARNN similar to IndRNN that results in better performance.

Dataset	NTU RGB-d		
	Accuracy CS (%)	Accuracy CV (%)	#Params
2-Layer LSTM [47]	60.09	67.29	>1M
2-Layer PLSTM [47]	62.93	70.27	>1M
Enhanced Visualization+CNN [37]	80.03	87.21	-
Pose Conditioned STA-LSTM [5]	77.10	84.50	-
6-Layer IndRNN [36]	81.80	87.97	2M
Dense IndRNN [35]	**84.88**	90.43	2.3M
3-Layer TARNN	80.52	87.54	**180K**
Dense TARNN	82.31	**90.86**	5.6M

RNN Trainability. *TARNN exhibits substantial improvement with respect to (a) size of memory footprint, (b) computational efficiency (faster convergence, training and inference times), and (c) generalization (test performance).* As evident from the Tables 1, 2, 3, and 4, TARNN is consistently among the models with lowest number of model parameters. It enjoys faster convergence rate as evident from the convergence plots for addition and copying tasks (Figure 2) and toy example (Appendix A.5). Thus improving the training time. It should also be noted that TARNN has similar inference time as vanilla RNNs. It also generalizes well as evident from the test accuracy on multiple synthetic and real-world tasks. This is attributed to the ability to achieve near identity gradients and effectively skipping uninformative input segments. This leads to the conclusion that TARNN improves vanilla RNN training. Due to the light footprint TARNN is suitable for IoT tasks. We tabulate results for IoT datasets where TARNN outperforms baselines (see Appendix 7).

5. Conclusion

We proposed a time adaptive RNN method for learning complex patterns in sequential data. Our method, based on modifying the time-constants of an ODE-RNN, the continuous-counterpart of the vanilla RNN, learns to skip uninformative inputs, while focusing on informative input segments. Additionally, we develop parameter constraints, which leads to lossless information propagation from informative inputs, by mitigating gradient explosion or decay. A number of experiments on benchmark datasets validates our approach against competitors with similar complexity. Indeed, we realize competitive performance with a lighter memory footprint, faster training time, without suffering performance degradation or increased inference time.

Acknowledgement

This research was supported by National Science Foundation grants CCF-2007350 (VS), CCF-2022446(VS), CCF-1955981 (VS), the Data Science Faculty and Student Fellowship from the Rafik B. Hariri Institute, the Office of Naval Research Grant N0014-18-1-2257 and by a gift from the ARM corporation.

References

[1] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the 4th International Conference on Ambient Assisted Living and Home Care*, IWAAL'12, pages 216–223, Berlin, Heidelberg, 2012. Springer-Verlag. 15

[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016. 1, 2, 13

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *International Conference on Learning Representations*, 2019. 8

[4] David Balduzzi and Muhammad Ghifary. Strongly-typed recurrent neural networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1292–1300, 2016. 3

[5] Fabien Baradel, Christian Wolf, and Julien Mille. Pose-conditioned spatio-temporal attention for human action recognition, 2017. 8

[6] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628, 2013. 2

[7] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016. 3, 6

[8] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Journal of Mathematics and Computation*, 1965. 3

[9] Víctor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*, 2018. 2, 5, 6

[10] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019. 1, 2, 5, 6, 7, 12, 13

[11] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018. 2, 3

[12] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. 1, 2

[13] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016. 2

[14] Tim Cooijmans, Nicolas Ballas, César Laurent, Çaxglar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016. 7

[15] Don Dennis, Durmus Alp Emre Acar, Vikram Mandikal, Vinu Sankar Sadasivan, Venkatesh Saligrama, Harsha Vardhan Simhadri, and Prateek Jain. Shallow rnn: Accurate time-series classification on resource constrained devices. In *Advances in Neural Information Processing Systems 32*, pages 12916–12926. Curran Associates, Inc., 2019. 1, 2

[16] N. Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W. Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2021. 2

[17] Daniel Fojo, Víctor Campos, and Xavier Giró i Nieto. Comparing fixed and adaptive computation time for recurrent neural networks, 2018. 6

[18] Kenichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801 – 806, 1993. 1

[19] Alex Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016. 2, 6

[20] Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. Neural speed reading with structural-jump-LSTM. In *International Conference on Learning Representations*, 2019. 2

[21] Josef Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 1991. 1, 2

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 5, 6

[23] Herbert Jaeger, Mantas Lukosevicius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks : the official journal of the International Neural Network Society*, 20:335–52, 05 2007. 2

[24] Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. In *International Conference on Learning Representations*, 2017. 2

[25] Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*, pages 1733–1741, 2017. 2

[26] Anil Kag, Ziming Zhang, and Venkatesh Saligrama. Incremental {rnn}: A dynamical view. In *International Conference on Learning Representations*, 2020. 2, 5, 6, 7, 12, 13, 16

[27] Giancarlo Kerg, Kyle Goyette, Maximilian Puelma Touzel, Gauthier Gidel, Eugene Vorontsov, Yoshua Bengio, and Guillaume Lajoie. Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13613–13623. Curran Associates, Inc., 2019. 2, 6, 7, 12, 13

[28] H.K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002. 1

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICML*, 2015. 6

[30] Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. volume 80 of *Proceedings of Machine Learning Research*, pages 2766–2775, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 8

[31] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, 2018. 1, 2, 5, 6, 7, 12, 13, 15

[32] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. 5

[33] Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018. 3, 6

[34] Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning (ICML)*, pages 3794–3803, 2019. 2, 6, 12, 13

[35] Shuai Li, Wanqing Li, Chris Cook, Yanbo Gao, and Ce Zhu. Deep independently recurrent neural network (indrnn), 2019. 3, 5, 8

[36] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 3, 5, 6, 7, 8

[37] Mengyuan Liu, Hong Liu, and Chen Chen. Enhanced skeleton visualization for view invariant human action recognition. *Pattern Recognition*, 68:346 – 362, 2017. 8

[38] W. Luo and F. Yu. Recurrent highway networks with grouped auxiliary memory. *IEEE Access*, 7:182037–182049, 2019. 8

[39] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 165–172, New York, NY, USA, 2013. ACM. 5

[40] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182*, 2017. 5

[41] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018. 8

[42] Zakaria Mhammedi, Andrew D. Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. *CoRR*, abs/1612.00188, 2016. 2

[43] Asier Mujika, Florian Meier, and Angelika Steger. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 5915–5924, 2017. 2

[44] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4785–4795. 2017. 13

[45] F. Rosenblatt. *Principles of neurodynamics.* Spartan Books, Washington, D.C., 1962. 1

[46] Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019. 2

[47] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2016. 5, 8

[48] Sachin S Talathi and Aniket Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*, 2015. 2

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 2

[50] Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv e-prints*, page arXiv:1804.03209, Apr. 2018. 15

[51] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank rnn language model, 2018. 5

[52] Adams Wei Yu, Hongrae Lee, and Quoc V. Le. Learning to skim text. *CoRR*, abs/1704.06877, 2017. 2

[53] Jiong Zhang, Qi Lei, and Inderjit S. Dhillon. Stabilizing gradients for deep neural networks via efficient svd parameterization. In *ICML*, 2018. 2, 5, 13

[54] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *ICML*, pages 4189–4198. JMLR. org, 2017. 2, 8

[55] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017. 8