TIME ADAPTIVE RECURRENT NEURAL NETWORK

Anonymous authors
Paper under double-blind review

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 (Dennis et al., 2019) 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 (Cho et al., 2014; Hochreiter, 1991; Arjovsky et al., 2016; Kusupati et al., 2018; Chang et al., 2019) 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 (Rosenblatt, 1962; Funahashi & Nakamura, 1993) that draw inspiration from ordinary differential equations (ODEs). (Chang et al., 2019) 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 (Khalil, 2002):

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

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 (Rosenblatt, 1962). 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, 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:

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

[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).

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 $\mathbf{s}_{m-1}, x_m$, we selectively adapt the amount of "pondering" that we need on each new input. Finally, we then 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).

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

$$\lambda(\mathbf{u}_m) \circ \dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m),\ \mathbf{z}(t_0) = \mathbf{s}_{m-1} \tag{3}$$

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 (Kusupati et al., 2018; Dennis et al., 2019) 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 Chen et al. (2018); Rubanova et al. (2019) 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 listed below.

- 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.
- Proposed 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 conduct extensive experiments on benchmark datasets, and show that we improve upon standard LSTM performance as well as other recent proposed works. We also demonstrate robustness to time-series distortions such as noise paddings.

Related Work. There is a rich literature on deep RNNs (Zilly et al., 2017; Bengio et al., 2013; Mujika et al., 2017), 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 vanilla RNN trainability through mitigation of vanishing and exploding gradients. First, there are works that propose improving state-transitions, based on unitary transition matrices (Arjovsky et al., 2016; Jing et al., 2017; Zhang et al., 2018; Mhammedi et al., 2016; Kerg et al., 2019; Lezcano-Casado & Martínez-Rubio, 2019), residual connections (Jaeger et al., 2007; Bengio et al., 2013; Kusupati et al., 2018) or gating (Hochreiter, 1991; Cho et al., 2014). 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. (Chang et al., 2019) and (Talathi & Vartak, 2015) 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. (Kag et al., 2020) propose Incremental-RNNs, a

[3](Chen et al., 2018; Rubanova et al., 2019), 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.

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 supress 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 (Graves, 2016; Chung et al., 2016; Yu et al., 2017; Jernite et al., 2017; Hansen et al., 2019)). 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 (Campos et al., 2018) is closely related to our viewpoint. Like us, (Campos et al., 2018) 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 (Campos et al., 2018) points out, our work can also be viewed as a temporal version of hard attention mechanisms for selecting image regions. These works (see (Campos et al., 2018)) that deal with visually-based sequential tasks, have high model-complexity, and are difficult to train on long input sequences. Others (Vaswani et al., 2017) leverage attention to bypass RNNs. In contrast, we offer an approach that is light-weight and improves RNN trainability on long-sequences.

2 Learning Time Adaptive Recurrent Neural Network (TARNN)

In this section we further present our objective, ODE discretization and further 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, $(y_1^{(i)}, y_2^{(i)}, \ldots, y_T^{(i)})$. The predictions $(\hat{y}_1^{(i)}, \hat{y}_2^{(i)}, \ldots, \hat{y}_T^{(i)})$ for each input $\mathbf{u}^{(i)}$ can be computed from the $D-$dimensional hidden states $(\mathbf{s}_1^{(i)}, \mathbf{s}_2^{(i)}, \ldots, \mathbf{s}_T^{(i)})$ 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.

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

Time-constants. We re-write the ODE in terms of $\boldsymbol{\beta}(\cdot)$, the inverse of $\boldsymbol{\lambda}(\cdot)$, since it is convenient for describing our discretization steps. We parameterize $\boldsymbol{\beta}(\mathbf{u}_m) = \sigma(\mathbf{U}_s\mathbf{s}_{m-1} + \mathbf{W}_x x_m)$, 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. 1 with the change of variables for time-constants, resulting in the ODE:

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

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:

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

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 $\mathbf{A}$ and consider two cases: (i) Decoupled with $\mathbf{A}$ as negative identity; (ii) mildly coupled with upper triangular identity blocks, i.e., $[\mathbf{A}]_{ii} = -1,\ [\mathbf{A}]_{i(i+D/2)} = 1,\ i \in [D/2]$, and else zero, to enhance state interaction. All the other model parameters $(\mathbf{B}, \mathbf{U}, \mathbf{W}, \mathbf{w}, b, \mathbf{U}_s, \mathbf{W}_x)$ parameterizing TARNN 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(Chen et al., 2018) or advanced root-finding methods such as the Broyden's method (Broyden, 1965). 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:

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

As shown in the Sec. 2.1, 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 : $\big(\mathbf{A}, \mathbf{U}, \mathbf{W}, \mathbf{U}_s, \mathbf{W}_s, \mathbf{B}\big)$
Initialize hidden state $\mathbf{s}_0 = 0$
for $m = 1$ **to** T **do**
  $\boldsymbol{\beta} = \sigma(\mathbf{U}_s \mathbf{s}_{m-1} + \mathbf{W}_x x_m)$
  $F(\cdot) = \boldsymbol{\beta} \odot (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))$
  $\mathbf{z}(t_1) = \text{ODESolve}(\ \mathbf{s}_{m-1}, x_m, F(\cdot), t_0, t_1\ )$
  $\mathbf{s}_m = \mathbf{z}(t_1)$
end for

2.1 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, or block triangular with identity blocks, 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. 1 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.

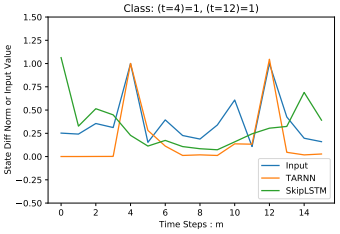
Theorem 1 shows that there is a configuration with lossless propagation. Thus, if it is necessary, the training algorithm will find a solution, that results in lossless propagation, even without imposing parameter constraints stated in the theorem. However, Theorem 1 suggests a natural regularizer, with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

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

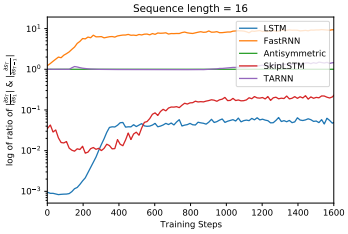
An interesting case is when B^2 row-wise sparse. In this case, states corresponding to zero rows operate as standard RNN (no linear term). We can ensure identity gradient holds in this case with block-wise parametric constraints, leading to more structured regularization penalty.

3 Experiments

Toy Example. For a sneak preview of our results, we illustrate the importance of both time-constants and gradient 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 the low-dimension, the (terminal) state cannot replicate the entire trace, requiring generalization. On the one hand, techniques



(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 (Campos et al., 2018) and proposed TARNN . Only ours responds to informative locations. Fig. (b) plots the norm of partials of hidden states. Only (Chang et al., 2019) 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.

that mitigate gradient explosion/decay like Antisymmetric (Chang et al., 2019), 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 also gets amplified in latent state updates.

On the other hand, SkipLSTM (Campos et al., 2018), which is capable of pondering at informative inputs and skipping uninformative inputs, is also ineffective. SkipLSTM (Campos et al., 2018) 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.

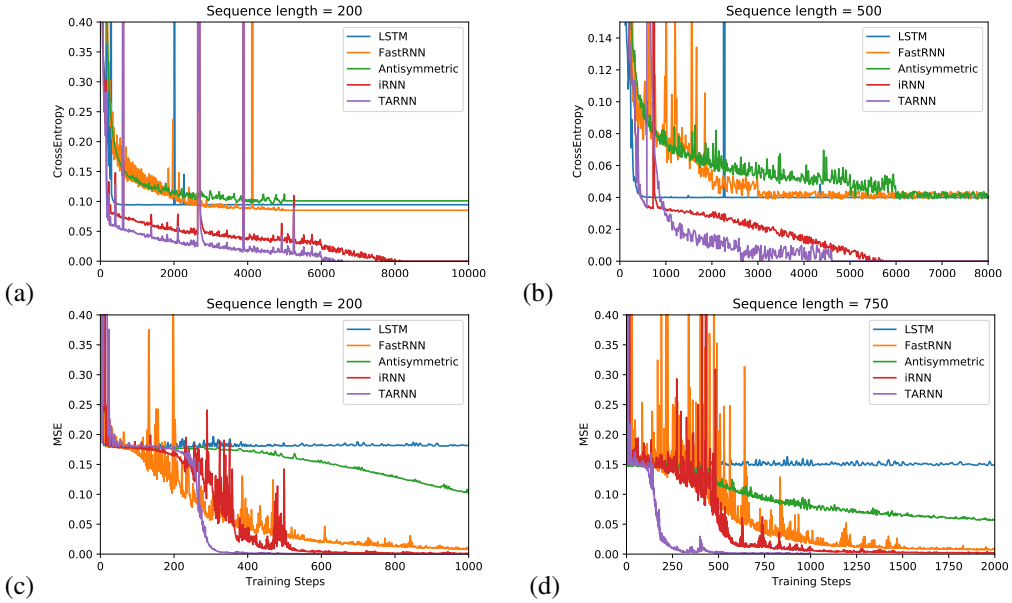


Figure 2: Following (Arjovsky et al., 2016) we display average Cross Entropy for the Copy Task (Sequence Length (baseline memoryless strategy)): (a) 200 (0.09) (b) 500 (0.039). Mean Squared Error for the Add Task, baseline performance is 0.167 (Sequence Length) : (c) 200 (d) 750.

3.1 Experimental Setup and Baselines

Datasets. Our experiments have been conducted on publicly available datasets with pre-processing and feature extraction details given in the appendix. We set aside 10% of training data for validation to tune hyper-parameters. Finally, models are trained on full train set, and reported performance achieved on the publicly available test set.

Code. We implemented the TARNN pseudo code shown in Algorithm 1 with tensorflow API. For most competing methods apart from (Chang et al., 2019), which we implemented, code is publicly available. We record the wall clock time for reporting both train and inference time. Experiments were run on an Nvidia GTX 1080 GPU on a machine with Intel Xeon 2.60 GHz CPU with 20 cores.

Baselines: We benchmark [4] TARNN against LSTM (Hochreiter & Schmidhuber, 1997), AntisymmetricRNN (Chang et al., 2019), FastRNN (Kusupati et al., 2018), iRNN (Kag et al., 2020). For baselines with gated/ungated variants, we report results for the best of the two. We also tried to incorporate SkipRNNs (Campos et al., 2018) 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 tables. Furthermore, adaptive computation time (ACT) (Graves, 2016) is not tabulated as we found that performance of SkipLSTM is significantly better. This has also been observed in (Fojo et al., 2018), who shows repeat-RNNs, a variant of iRNN outperforms ACT. Unitary RNN is excluded mainly due to their significant training costs. Additionally, except for the copy task, which in particular, Unitary RNN benefits by leveraging the modReLU function (Vorontsov et al., 2017), we find it to be dominated by our reported competitors. We include recent Unitary/Orthogonal RNNs (Kerg et al., 2019; Lezcano-Casado & Martínez-Rubio, 2019), which are computationally cheaper than earlier proposals. Unfortunately, we do not report their train time as their code is written in PyTorch.

Hyper Parameters We used grid search on validation wherever possible to set the hyper-parameters of each algorithm, or according to the settings published in (Kusupati et al., 2018; Arjovsky et al., 2016). We used ReLU as the non-linearity and Adam ((Kingma & Ba, 2015)) as the optimizer

[4]For fair comparison, methods that leverage dataset specific heuristics are omitted.

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.

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

for all the experiments. We provide the final hyper-parameters along with the grid values for our experiments in the appendix A.4. Our inference time is comparable to FastRNNs and iRNNs, in contrast LSTMs take 4x longer for inference (see Appendix A.9).

3.2 Synthetic and Real-World Tasks

We list four types of datasets, all of which in some way require effective gradient propagation and focus on informative time-segments: (1) Standard Benchmark tasks (Add & Copy tasks) that illustrate that TARNN can rapidly learn long-term dependence (LTD); (2) Benchmark vision tasks (pixel MNIST, perm-MNIST) that may not require long-term, but nevertheless, demonstrates that TARNN achieves SOTA for short term dependencies but with less resources. (3) Noise Padded Vision tasks (Noisy MNIST, Noisy CIFAR), where a large noise time segment separates information segments and the terminal state, and so the learner must extract informative parts while rejecting the noisy parts; (4) Sequence-sequence prediction task (PTB language modeling) that are different from terminal prediction and have short-term dependency.

(A) Benchmark LTD Tasks (Addition & Copy Memory) TARNN *rapidly learns and solves the long term dependencies*. These tasks (Hochreiter & Schmidhuber, 1997) have long been used as benchmarks in the literature to evaluate LTD (Hori et al., 2017; Zhang et al., 2018; Arjovsky et al., 2016; Martens & Sutskever, 2011). We follow the setup described in (Arjovsky et al., 2016) to create the adding and copying tasks. See appendix for description.

Figure 2 shows the average performance of various methods on these tasks. For the copying task we observe that TARNN converges rapidly to the naive baseline and is the only method except iRNN to achieve zero average cross entropy. For the addition task, all three FastRNN, iRNN and TARNN solves the addition task but FastRNN takes twice the number of iterations to reach desired 0 MSE. In both the tasks, TARNN performance is much more stable across number of online training samples. In contrast, other methods either takes a lot of samples to match TARNN 's performance or exhibit high variance in the evaluation metric. This shows that TARNN *converges faster than the baselines* (to the desired error). We omitted reporting unitary RNN variants (see appendix for explanation).

(B) Vision Tasks (Pixel & Permute MNIST). *TARNN exhibits* $1.5x$ *training speedup against the strongest baselines achieving higher accuracy thus demonstrating rapid convergence and generalization.* We report performance on the sequential vision tasks: (a) classification of MNIST images on a pixel-by-pixel sequence; (b) a fixed random permuted MNIST sequence (Lecun et al., 1998). These tasks typically do not fall in the LTD categories (Chang et al., 2019), but are useful to demonstrate faster training, which is attributed to better gradients.

TARNN outperforms all the baselines in these tasks. For the pixel-MNIST task, (Kusupati et al., 2018) reports that it takes significantly longer time for existing (LSTMs, Unitary, Gated, Spectral)

RNNs to converge to reasonable performance. In contrast, FastRNN trains at least $2x$ faster than LSTMs. Table 1 shows a $9x$ speedup relative to LSTMs, and $1.5x$ speedup in comparison to iRNN[5].

(C) Noise padding (Noisy-MNIST, Noisy-CIFAR) *TARNN is noise resilient, bypassing noisy time segments.* Additionally, as in (Chang et al., 2019; Kag et al., 2020), we *induce* LTD by padding CIFAR-10 and MNIST with noise exactly replicating their setup, resulting in Noisy-CIFAR and Noisy-MNIST. Intuitively we expect our model to be resilient to such perturbations and to be able to focus on the informative segments in the signal. We attribute TARNN 's superior performance to the fact that it is capable of suppressing noise, and focusing on signal part of the sequence along with well behaved gradient propagation. After processing the input signal, the state $\mathbf{s}_m$ will cease to update once it starts to encounter noise. Thus information from signal component is better preserved.

Results for Noisy-MNIST and Noisy-CIFAR are shown in Table 1. Note that almost all timesteps contain noise in these datasets. LSTMs perform poorly on these tasks due to vanishing gradients as observed earlier (Chang et al., 2019; Kag et al., 2020), while SkipLSTM performs similar to LSTM on this task, also due poor gradients. TARNN outperforms the baselines w.r.t accuracy on CIFAR-10, while on MNIST the gains are smaller, as it's a relatively easier task.

(D) Language Modelling. *TARNN adapts well, and achieves signicant improvements over competing methods on short-term dependency tasks.* We benchmark TARNN against sequence to sequence language modelling on Penn Tree Bank (PTB) dataset. We follow (Kusupati et al., 2018; Zhang et al., 2018) to setup our PTB experiments. We only pursue one layer language modelling, but with more difficult sequence length (300). This setting corresponds to the small configuration used by (McAuley & Leskovec, 2013) which consists of only one layer, with only difference in the sequence length. We do not report expRNN and nnRNN results are they perform poorly in comparison to LSTM (Kerg et al., 2019). Table 2 reports all the evaluation metrics for the PTB Language modelling task with 1 layer as setup by (Kusupati et al., 2018). It can be clearly seen that TARNN outperforms the baselines by roughly ≈ 10 point difference in the test perplexity for similar model complexity while it achieves ≈ 20 points for slightly larger model.

Table 2: PTB Language Modeling: 1 Layer (standard small config except the sequence length is 300 as per (Kusupati et al., 2018) as opposed to 30 in the conventional PTB).

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	262K

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, and 2 TARNN is consistently among the models with lowest number of model parameters. TARNN enjoys faster convergence rate as evident from the addition and copying tasks (Figure 2) and convergence plot for toy example (see appendix). Thus improving the training time. It should also be noted that TARNN has similar inference time as vanilla RNNs. TARNN 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. All of these leads us to conclude that TARNN improves vanilla RNN training. Due to the lightweight footprint TARNN is suitable for IoT tasks. We tabulate results for IoT datasets where TARNN outperforms baselines (see Appendix 5).

4 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.

[5]LSTMs have achieved roughly 98.9 with dataset specific heuristics (Cooijmans et al., 2016).

REFERENCES

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, pp. 216–223, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-35394-9. doi: 10.1007/978-3-642-35395-6_30. URL http://dx.doi.org/10.1007/978-3-642-35395-6_30.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

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

C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Journal of Mathematics and Computation*, 1965. doi: https://doi.org/10.1090/S0025-5718-1965-0198670-6.

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. URL https://openreview.net/forum?id=HkwVAXyCW.

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. URL https://openreview.net/forum?id=ryxepo0cFX.

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

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)*, pp. 1724–1734, 2014. doi: 10.3115/v1/D14-1179. URL http://www.aclweb.org/anthology/D14-1179.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016. URL http://arxiv.org/abs/1609.01704.

Tim Cooijmans, Nicolas Ballas, César Laurent, Ça xglar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

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*, pp. 12916–12926. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9451-shallow-rnn-accurate-time-series-classification-on-resource-constrained-devic
pdf.

Daniel Fojo, Víctor Campos, and Xavier Giró i Nieto. Comparing fixed and adaptive computation time for recurrent neural networks, 2018. URL https://openreview.net/forum?id=SkZq3vyDf.

Kenichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801 – 806, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80125-X. URL http://www.sciencedirect.com/science/article/pii/S089360800580125X.

Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. Frage: frequency-agnostic word representation. In *Advances in Neural Information Processing Systems*, pp. 1334–1345, 2018.

Alex Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016. URL http://arxiv.org/abs/1603.08983.

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. URL https://openreview.net/forum?id=B1xf9jAqFQ.

Josef Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 1991. URL http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf.

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

Chiori Hori, Takaaki Hori, Teng-Yok Lee, Ziming Zhang, Bret Harsham, John R Hershey, Tim K Marks, and Kazuhiko Sumi. Attention-based multimodal fusion for video description. In *ICCV*, pp. 4203–4212, 2017.

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. doi: 10.1016/j.neunet.2007.04.016.

Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1LVSrcge.

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*, pp. 1733–1741, 2017.

Anil Kag, Ziming Zhang, and Venkatesh Saligrama. Incremental {rnn}: A dynamical view. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylpqA4FwS.

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 (eds.), *Advances in Neural Information Processing Systems 32*, pp. 13613–13623. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9513-non-normal-recurrent-neural-network-nnrnn-learning-long-time-dependencies-whi
pdf.

H.K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002. ISBN 9780130673893. URL https://books.google.com/books?id=t_d1QgAACAAJ.

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

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.

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

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)*, pp. 3794–3803, 2019.

James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, 2011.

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, pp. 165–172, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2409-0. doi: 10.1145/2507157.2507163. URL http://doi.acm.org/10.1145/2507157.2507163.

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. URL http://arxiv.org/abs/1612.00188.

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

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

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 (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4785–4795. 2017.

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

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019. URL http://arxiv.org/abs/1907.03907.

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

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.

Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *ICML*, pp. 3570–3578, 2017. URL http://proceedings.mlr.press/v70/vorontsov17a.html.

Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv e-prints*, art. arXiv:1804.03209, April 2018.

Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4880–4888. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6327-full-capacity-unitary-recurrent-neural-networks.pdf.

Adams Wei Yu, Hongrae Lee, and Quoc V. Le. Learning to skim text. *CoRR*, abs/1704.06877, 2017. URL http://arxiv.org/abs/1704.06877.

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

Jiong Zhang, Qi Lei, and Inderjit S Dhillon. Stabilizing gradients for deep neural networks via efficient SVD parameterization. *arXiv preprint arXiv:1803.09327*, 2018.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *ICML*, pp. 4189–4198. JMLR. org, 2017.

A Appendix

A.1 Proofs

Proposition 3. *Consider the ODE in Eq. 5 and assumptions on* $\mathbf{A}$ *described above. Suppose we have* $\|U\| < \alpha/2$, *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.*

To prove the proposition, we must find a solution to the non-linear equation $\mathbf{Az} + \mathbf{Bu}_m + \phi(\mathbf{Uz} + \mathbf{Wu}_m) = 0$ and show that it is unique. We do this by constructing a fixed-point iterate, and show that the iteration is contractive.

To this end, define $\Gamma(\mathbf{z}) = \mathbf{z}+\eta(\mathbf{Az}+\mathbf{Bu}_m+\phi(\mathbf{Uz}+\mathbf{Wu}_m))$, and note that for any two $\mathbf{z}, \mathbf{z}' \in \mathbb{R}^D$, we find,

$$\begin{aligned}
\|\Gamma(\mathbf{z}) - \Gamma(\mathbf{z}')\| &\leq \|(I + \eta\mathbf{A})(\mathbf{z} - \mathbf{z}')\| + \eta\|\phi(\mathbf{Uz} + \mathbf{Wu}_m)) - \phi(\mathbf{Uz}' + \mathbf{Wu}_m))\| \\
&\leq \|(I + \eta\mathbf{A})(\mathbf{z} - \mathbf{z}')\| + \eta\|\mathbf{U}(\mathbf{z} - \mathbf{z}')\| \\
&\leq \sigma_{\max}(I + \eta\mathbf{A})\|(\mathbf{z} - \mathbf{z}')\| + \eta\|\mathbf{U}(\mathbf{z} - \mathbf{z}')\| \\
&\leq \sigma_{\max}(I + \eta\mathbf{A})\|\mathbf{z} - \mathbf{z}'\| + \eta\|\mathbf{U}\|\|\mathbf{z} - \mathbf{z}'\| \\
\implies \|\Gamma(\mathbf{z}) - \Gamma(\mathbf{z}')\| &\leq \Big(\sigma_{\max}(I + \eta\mathbf{A}) + \eta\|\mathbf{U}\|\Big)\|\mathbf{z} - \mathbf{z}'\| = \gamma\|\mathbf{z} - \mathbf{z}'\| \qquad (7)
\end{aligned}$$

If the constant $\gamma < 1$, then the above inequality proves that Γ is a contraction. The result then follows by invoking the Banach fixed point theorem (contraction-mapping theorem). All that remains to show is that $\gamma = \sigma_{\max}(I + \eta\mathbf{A}) + \eta\|\mathbf{U}\| < 1$. From assumptions we have, $\sigma_{\max}(I + \eta\mathbf{A}) \leq 1 - \alpha\eta$ and $\|\mathbf{U}\| < \alpha,$ where $\alpha > 0$; $\implies \gamma < 1 - \alpha\eta + \alpha\eta \leq 1$.

Proposition 4. *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.*

Let $\mathbf{z}^*$ be the equilibrium solution, i.e. $\mathbf{Az}^* + \mathbf{Bu}_m + \phi(\mathbf{Uz}^* + \mathbf{Wu}_m) = 0$. 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. Consequently, we can assume without loss of generality that $(\boldsymbol{\beta})_j = 1$ for all $j \in [D]$. The gradient of the Lyapunov function along the ODE system trajectories can be written as

$$\begin{aligned}
\frac{dV(z(t))}{dt} &= (\dot{\mathbf{z}}(t))^\top(\mathbf{z}(t) - \mathbf{z}^*) + (\mathbf{z}(t) - \mathbf{z}^*)^\top\dot{\mathbf{z}}(t) \\
&=(\mathbf{Az}(t) + \mathbf{Bu}_m + \phi(\mathbf{Uz}(t) + \mathbf{Wu}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*) \\
&+(\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{Az}(t) + \mathbf{Bu}_m + \phi(\mathbf{Uz}(t) + \mathbf{Wu}_m)) \\
&=(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \mathbf{B}(\mathbf{u}_m - \mathbf{u}_m) + \phi(\mathbf{Uz}(t) + \mathbf{W}x_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*) \\
&\quad + (\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \mathbf{B}(\mathbf{u}_m - \mathbf{u}_m) + \phi(\mathbf{Uz}(t) + \mathbf{W}x_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m)) \\
&=(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \phi(\mathbf{Uz}(t) + \mathbf{W}x_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*) \\
&\quad + (\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A}(\mathbf{z}(t) - \mathbf{z}^*) + \phi(\mathbf{Uz}(t) + \mathbf{W}x_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m)) \\
&=(\mathbf{z}(t) - \mathbf{z}^*)^\top(\mathbf{A} + \mathbf{A}^\top)(\mathbf{z}(t) - \mathbf{z}^*) + 2(\phi(\mathbf{Uz}(t) + \mathbf{Wu}_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*)
\end{aligned}$$

We now invoke Cauchy-Schwartz inequality to bound the second term, namely,

$$\begin{aligned}
|(\phi(\mathbf{Uz}(t) + \mathbf{Wu}_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m))^\top(\mathbf{z}(t) - \mathbf{z}^*)| &\leq \|(\phi(\mathbf{Uz}(t) + \mathbf{Wu}_m) - \phi(\mathbf{Uz}^* + \mathbf{Wu}_m))\|\|\mathbf{z}(t) - \mathbf{z}^*\| \\
&\leq \|\mathbf{U}\|\|\mathbf{z}(t) - \mathbf{z}^*\|\|\mathbf{z}(t) - \mathbf{z}^*\| < \|\mathbf{z}(t) - \mathbf{z}^*\|^2
\end{aligned}$$

where in the last inequality we used the fact that $\phi(\cdot)$ is 1-Lipshitz and $\|\mathbf{U}\| < \alpha \leq 1$. As a result, we have,

$$\frac{dV(z(t))}{dt} < (\lambda_{\max}(\mathbf{A} + \mathbf{A}^T) + 1)\|\mathbf{z} - \mathbf{z}^*\|^2 \leq 0$$

where the last inequality follows because, we have $(\lambda_{\max}(\mathbf{A} + \mathbf{A}^T) \leq -1)$. This shows that the ODE is globally asymptotically stable and converges to a unique equilibrium point. To show a linear rate of convergence we note that K-fold iterations of the Euler method (see Prop 1), $\mathbf{z}^k = \Gamma(\mathbf{z}^{k-1}) = \mathbf{z}^{k-1} + \eta(\mathbf{Az}^{k-1} + \mathbf{Bu}_m + \phi(\mathbf{Uz}^{k-1} + \mathbf{Wu}_m))$, results in,

$$\|\mathbf{z}^K - \mathbf{z}^*\| \leq \gamma^K\|\mathbf{z}^0 - \mathbf{z}^*\|$$

which follows directly from the fact that $\mathbf{z}^K = \Gamma(\mathbf{z}^{K-1})$, $\mathbf{z}^* = \Gamma(\mathbf{z}^*)$, and Γ is a contraction as obtained by Eq. 7. This establishes the linear-rate of convergence.

Table 3: Dataset Statistics

Dataset	Avg. Activity Time	Input Time	Sequence Ratio	#Train	#Fts	#Steps	#Test
Noisy-MNIST	28	1000	7/250	60,000	28	1000	10,000
Noisy-CIFAR	32	1000	4/125	60,000	96	1000	10,000
Pixel-MNIST				60,000	1	784	10,000
Permuted-MNIST				60,000	1	784	10,000
PTB				929,589	300	300	82,430

Proof of Theorem 1
Note that, when $\beta_i = 0$, $\mathbf{s}_m(i) = \mathbf{s}_{m-1}(i)$. On the other hand when $\beta_i > 0$, the system is in equilibrium, and for those components, j, we have

$$(\dot{\mathbf{z}}(t))_j = (F(\mathbf{z}(t), \mathbf{u}_m))_j = 0, \text{ where } F(\mathbf{z}(t), \mathbf{u}_m) = \beta(\mathbf{u}_m) \circ (\mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}\mathbf{u}_m))$$

Now $(F(\mathbf{s}_m, \mathbf{u}_m))_k = 0$ regardless of β_k. This is because if $\beta_k(\mathbf{u}_m) > 0$ we reach equilibrium, and $\dot{\mathbf{z}}(t) = 0$, and on the other have if $\beta_k = 0$ then $(F(\mathbf{s}_m, \mathbf{u}_m))_k = 0$ in any case. With this in mind, define $D = \text{diag}[\mathbf{1}_{\beta_j(\mathbf{u}_m) > 0}]$. We then write the vector $\mathbf{s}_m = D\mathbf{s}_m + (I - D)\mathbf{s}_{m-1}$. Let $J_{m,m-1}$ denote the Jacobian of $\mathbf{s}_m$ with respect to s_{m-1}. Taking derivatives we get,

$$\begin{aligned} 0 = \nabla F(\mathbf{s}_m, \mathbf{u}_m) =& \beta(\mathbf{u}_m) \circ (\mathbf{A}(DJ_{m,m-1} + (I - D)) + \mathbf{B}_2) \\ &+ \beta(\mathbf{u}_m) \circ (\nabla\phi(\mathbf{U}(DJ_{m,m-1} + (I - D)) + \mathbf{W}_2)) \\ &+ D\nabla\sigma(\mathbf{U}_s\mathbf{s}_{m-1} + \mathbf{W}_x x_m)(\mathbf{A}\mathbf{s}_m + \mathbf{B}\mathbf{u}_m + \phi(\mathbf{U}\mathbf{s}_m + \mathbf{W}\mathbf{u}_m)) \end{aligned}$$

First, note that the third term is always zero, due to the fact we noted earlier, namely, if a component is active, then the corresponding state reaches equilibrium, and there is nothing to do if the component is otherwise inactive. Now noting that $\mathbf{A} = \mathbf{B}_2$ and $\mathbf{U} = \mathbf{W}_2$, we get,

$$\nabla F(\mathbf{s}_m, \mathbf{u}_m) = \beta(\mathbf{u}_m) \circ (\mathbf{A}D)(J_{m,m-1} - I) + \nabla\phi(\cdot)\mathbf{U}D(J_{m,m-1} - I)$$

Collecting the common terms, we have,

$$\nabla F(\mathbf{s}_m, \mathbf{u}_m) = \beta(\mathbf{u}_m) \circ (\mathbf{A} - \nabla\phi U)D(J_{m,m-1} - I)$$

Now for the case in hand, $\|\nabla\phi U\| < 1$, and since $\|\mathbf{A}\| \geq 1$, the middle term is non-zero. This implies that for all the active components, $(J_{m,m-1})_{kk} = 1$.

For the other case, the proof follows in an identical manner. Specifically, for the non-zero rows of $\mathbf{B}$ the proof is identical, and the claims hold for those associated state components. For the rows with zero rows since,

A.2 Dataset Details

Table 3 lists the statistics of all the datasets described below.

Penn Treebank: 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 the number of hidden units of architecture. This is the setup used in Kusupati et al. (2018); Zhang et al. (2018). 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 also means we require less number of parameters for the embedding representation.

Pixel-MNIST: Pixel-by-pixel version of the standard MNIST-10 dataset [6]. Each image in the dataset has image size 28×28. In this task, we provide the RNN, each pixel in a sequential manner resulting in the 784 length sequential input. The dataset was zero mean - unit variance normalized during training and prediction.

Permuted-MNIST: This is similar to Pixel-MNIST, except its made harder by shuffling the pixels with a fixed permutation. We keep the random seed as 42 to generate the permutation of 784 pixels.

[6] http://yann.lecun. com/exdb/mnist/

Noisy-MNIST: To introduce more long-range dependencies to the Pixel-MNIST task, we define a more challenging task called the Noisy-MNIST, inspired by the noise padded experiments in Chang et al. (2019). Instead of feeding in one pixel at one time, we input each row of a MNIST image at every time step. After the first 28 time steps, we input independent standard Gaussian noise for the remaining time steps. Since a MNIST image is of size 28 with 1 RGB channels, the input dimension is m = 28. The total number of time steps is set to T = 1000. In other words, only the first 28 time steps of input contain salient information, all remaining 972 time steps are merely random noise. For a model to correctly classify an input image, it has to remember the information from a long time ago. This task is conceptually more difficult than the pixel-by-pixel MNIST, although the total amount of signal in the input sequence is the same.

Noisy-CIFAR: This is exactly replica of the noise paded CIFAR task mentioned in Chang et al. (2019). Instead of feeding in one pixel at one time, we input each row of a CIFAR-10 image at every time step. After the first 32 time steps, we input independent standard Gaussian noise for the remaining time steps. Since a CIFAR-10 image is of size 32 with three RGB channels, the input dimension is m = 96. The total number of time steps is set to T = 1000. In other words, only the first 32 time steps of input contain salient information, all remaining 968 time steps are merely random noise. For a model to correctly classify an input image, it has to remember the information from a long time ago. This task is conceptually more difficult than the pixel-by-pixel CIFAR-10, although the total amount of signal in the input sequence is the same.

Addition Task: We closely follow the adding problem defined in Arjovsky et al. (2016); Hochreiter & Schmidhuber (1997) to explain the task at hand. Each input consists of two sequences of length T. The first sequence, which we denote x, consists of numbers sampled uniformly at random $\mathcal{U}[0, 1]$. The second sequence is an indicator sequence consisting of exactly two entries of 1 and remaining entries 0. The first 1 entry is located uniformly at random in the first half of the sequence, whilst the second 1 entry is located uniformly at random in the second half. The output is the sum of the two entries of the first sequence, corresponding to where the 1 entries are located in the second sequence. A naive strategy of predicting 1 as the output regardless of the input sequence gives an expected mean squared error of 0.167, the variance of the sum of two independent uniform distributions.

Copying Task: Following a similar setup to Arjovsky et al. (2016); Hochreiter & Schmidhuber (1997), we outline the copy memory task. Consider 10 categories, $\{a_i\}_{i=0}^{9}$. The input takes the form of a $T + 20$ length vector of categories, where we test over a range of values of T. The first 10 entries are sampled uniformly, independently and with replacement from $\{a_i\}_{i=0}^{7}$, and represent the sequence which will need to be remembered. The next $T - 1$ entries are set to a_8, which can be thought of as the 'blank' category. The next single entry is a_9, which represents a delimiter, which should indicate to the algorithm that it is now required to reproduce the initial 10 categories in the output. The remaining 10 entries are set to a_8. The required output sequence consists of $T + 10$ repeated entries of a_8, followed by the first 10 categories of the input sequence in exactly the same order. The goal is to minimize the average cross entropy of category predictions at each time step of the sequence. The task amounts to having to remember a categorical sequence of length 10, for T time steps.

A simple baseline can be established by considering an optimal strategy when no memory is available, which we deem the memoryless strategy. The memoryless strategy would be to predict a_8 for $T + 10$ entries and then predict each of the final 10 categories from the set $\{a_i\}_{i=0}^{7}$ i=0 independently and uniformly at random. The categorical cross entropy of this strategy is $\frac{10 \log(8)}{T+20}$

A.3 Baseline Justification

In our experiments section, we stated that some of the potential baselines were removed due to experimental conditions enforced in the setup. Here we clearly justify our choice. Mostly the reasoning is to avoid comparing complementary add-ons and compare the bare-bone cells.

- Cooijmans et al. (2016) is removed since its an add-on and can be applied to any method. Besides its pixel-mnist results involve dataset specific heuristics.
- Gong et al. (2018) is also an add-on and hence can be applied to any method.

- Zilly et al. (2017); Pascanu et al. (2013); Mujika et al. (2017) denote deep transitioning methods. They are add-ons for any single recurrent block and hence can be applied to any recurrent cell.
- **Unitary RNN Variants.** Results for methods based on unitary transitions (such as Arjovsky et al. (2016); Wisdom et al. (2016); Vorontsov et al. (2017); Zhang et al. (2018)) are not reported in the main paper (when available reported in appendix) for the following reasons: (a) They are substantially more expensive, and requiring large model sizes; (b) Apart from the benchmark copy and add tasks, results tabulated by FastRNN and Antisymmetric authors (see Zhang et al. (2018); Chang et al. (2019)) show that they are well below SOTA; (c) TARNN dominates unitary-RNN variants on add-task; (d) On copy task, while unitary invariants are superior, Vorontsov et al. (2017) attributes it to modReLU or leaky ReLU activations. Leaky ReLUs allow for linear transitions, and copy task being a memory task benefits from it. With hard non-linear activation, unitary RNN variants can take up to 1000's of epochs for even 100-length sequences (Vorontsov et al. (2017)).

  We omitted reporting unitary RNN variants for Add and Copy task. On Add-task we point out that our performance is superior. In particular, for the longer $T = 750$ length, Arjovsky et al. (2016), points out that MSE does not reach zero, and uRNN is noisy. Others either (Wisdom et al., 2016) do not report add-task or report only for shorter lengths (Zhang et al., 2018)
- LSTM solves the addition task in (Arjovsky et al., 2016) after $10K$ steps while we only use $1K$ steps.

A.4 Implementation Details

We acquired the publicly available code for the baselines except Antisymmetric RNN (Chang et al., 2019) and Incremental RNN(Kag et al., 2020). We write the RNN cell implemntation for Antisymmetric RNN and Incremental RNNs from the pseudo code provided in their papers. Before running our grid search, we ensured that we were able to reproduce the publicly reported results. Following which we run our experiments for suggested hidden states as per the previous works for each dataset.

In order to avoid non-determinism in the experiments, we initialize both the numpy and tensorflow random library with the same seed number, 1234. Our parameter matrices are initialized with a random normal initializer with mean 0 and standard deviation 0.1 while our time-constant biases are initialized with -3.0 and remaining biases are initialized with 0.

We provide the pseudo code in Algorithm 1 to generate the hidden states of the TARNN . In order to implement this routine on a deep learning framework, we need to elaborate a bit more about the ODESolve function. We implement the Euler iterations described in the practical implementations in the method section. Following the recommendation from Kag et al. (2020) and the fact that many of these datasets are slowly time varying, we use the $K = 5$ in the Euler recursions to reach the equilibrium. Table 4 provides the number of hidden units used for different datasets.

Table 4: Various hyper-parameters to reproduce results

Dataset	Hidden Dimension (hr)	Learning Rate (hr)	L2 regularization	Init η	Epochs τ		Batch Size
Pixel-MNIST	128	$1e^{-2}$	$4.5e^{-6}$	0.08	30	5	128
Permuted-MNIST	128	$1e^{-2}$	$4.5e^{-6}$	0.0008	30	5	128
Noisy-MNIST	128	$1e^{-2}$	$4.5e^{-5}$	0.0008	30	5	512
Noisy-CIFAR	128	$1e^{-2}$	$4.5e^{-5}$	0.001	30	5	256
Addition Task	128	$1e^{-2}$	$1.0e^{-5}$	0.001	2	-	128
Copying Task	128	$1e^{-2}$	$1.0e^{-6}$	0.45	-	-	128
PTB	256	-	-	0.001	100	-	

Our experiments use hidden size as suggested by (Kusupati et al., 2018; Chang et al., 2019) i.e. 128. We point out that this is not the setting used by (Kerg et al., 2019; Lezcano-Casado & Martínez-Rubio, 2019) as their best results are achieved with much larger state space i.e. 512 state dimension, thus

requiring much larger models. Thus, in order to provide fair comparison we only allow state space as 128 dimensions.

In order to enable grid search on the baseline methods, we use the method specific hyper-parameter values suggested in the respective baselines. We allow the methods to pick the non-linearity from the set $\{$ ReLU, tanh, sigmoid $\}$. For Antisymmetric RNN, as per their recommendation we step size from the set $\{0.01, 0.1, 1\}$ and diffusion parameter $\gamma \in \{0.001.0.01, 0.1, 1.0\}$. For nnRNN and expRNN methods, we follow the hyper-parameter search grid as suggested in (Kerg et al., 2019).

We use grid search for tuning the hyper-parameters for the methods. We used the values $[4.5E-6, 4.5E-5, 4.5E-4, 1E-6, 1E-5, 1E-4]$ for L2 regularization. We searched over $[1e-2, 1e-3, 1e-4]$ as the base learning rates which are halved after each $\tau = [5, 10, 20]$ epochs have passed. We allowed the methods to train for $[30, 50, 100, 300]$ epochs. We use ReLU as the non-linearity for all of our experiments except in Copy and PTB tasks where we use tanh as the non-linearity (performs better than ReLU).

We point out that we set $\mathbf{A} = -I$ for all our experiments except Pixel-MNIST/Permute-MNIST tasks where we use $\mathbf{A}$ to be the blocked triangular identity matrix as mentioned in the analysis Section 2.1. This allows us to couple the linear part resulting in better performance on these tasks in comparison to the $\mathbf{A} = -I$ configuration.

Note that the settings used for PTB dataset corresponds to the small configuration with 300 as the sequence length. We piggy back on the configuration changes used in (Kusupati et al., 2018; Kag et al., 2020; Zhang et al., 2018) which describes the learning rate along with the learning rate schedule and the number of epochs all the methods are trained. Thus, we do not list these hyper-parameters in the table 4.

A.5 Unitary RNNs do not solve vanishing gradients.

(Lezcano-Casado & Martínez-Rubio, 2019; Kerg et al., 2019) and others propose to "cheaply" design orthonormal transition matrices (OTM), appealing to Arjovsky et al. (2016) for justification. Arjovsky et al. (2016) (Eq. 4) only shows an upper-bound with ReLU + OTM. This solves exploding gradients, but the more pernicious vanishing gradients remains (RELU+OTM is discussed in Pennington et al. (2017) [PSG17]). In Arjovsky et al. (2016)'s notation with D_k binary diagonal arising from ReLU activations, W unitary, we would need, $\|\partial C/\partial h_T(\prod_{s=t}^{T-1} D_s W^\top)\| \geq \|\partial C/\partial h_T\|$. This is generally not true due to matrix non-commutativity. E.g. for $t = T-2$, this is possible if $\|D_{T-1}W^\top D_{T-2}W^\top\| = \|D_{T-1}W^\top D_{T-2}\| \geq 1$. Unless, $D_{T-1} = D_{T-2}$ is identity, $D_{T-1}W D_{T-2}$ is a submatrix of W, and generically has norm less than one.

A.6 Relationship to existing Recurrent architectures.

We will now briefly discuss other recurrent architectures in the literature to gain intuition into our framework. We will refer to the ODE Eq. 5

(a) Vanilla RNNs: Setting $\beta = 1$, $\mathbf{A} = -\mathbf{I}$, $\mathbf{B}^1 = \mathbf{0}; \mathbf{B}^2 = \mathbf{0}$, results in the ODE, $\dot{\mathbf{z}}(t) = -\mathbf{z}(t) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}x_m)$; $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$. Euler discretization of this ODE with only one step results in Vanilla RNNs.

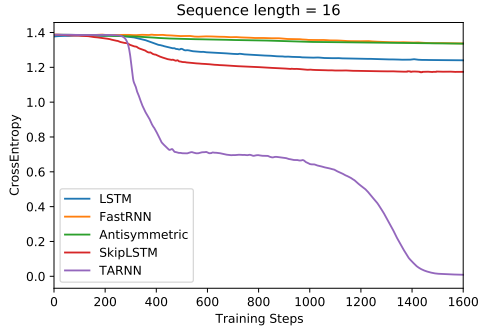
(b) Fast/Antisymmetric RNNs: Setting $\beta = 1$, $\mathbf{A} = \mathbf{0}$, $\mathbf{B}^1 = \mathbf{0}; \mathbf{B}^2 = \mathbf{0}$, results in the ODE, $\dot{\mathbf{z}}(t) = \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}x_m)$; $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$. Euler discretization of this ODE with only one step results in Kusupati et al. (2018); Chang et al. (2019).

(c) Incremental RNNs: Setting $\beta = 1$, $\mathbf{A} = -\mathbf{I}$, $\mathbf{B}^1 = \mathbf{0}; \mathbf{B}^2 = \mathbf{I}$, results in the ODE, $\dot{\mathbf{z}}(t) = -\mathbf{z}(t) + \phi(\mathbf{U}\mathbf{z}(t) + \mathbf{W}x_m)$; $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$. Since the initial state of the ODE, $\mathbf{z}(t_0) = \mathbf{s}_{m-1}$, we can write it into $\dot{\mathbf{z}}(t) = -(\mathbf{z}(t) - \mathbf{s}_{m-1}) + \phi(\mathbf{U}(\mathbf{z}(t) - \mathbf{s}_{m-1}) + \mathbf{W}x_m)$ with $\mathbf{z}(t_0) = 0$. This ODE is equivalent to Kag et al. (2020).

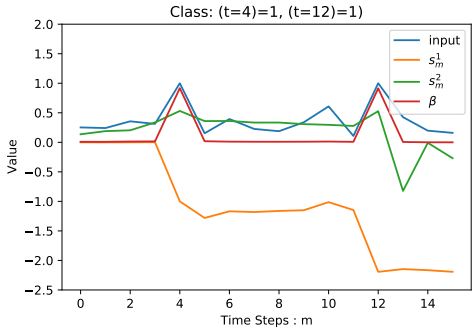
A.7 Additional plots for Toy Example.

We add additional figures for the toy example in order to describe the following properties: (a) TARNN achieves faster convergence than the baselines, (b) TARNN time constants activate at the

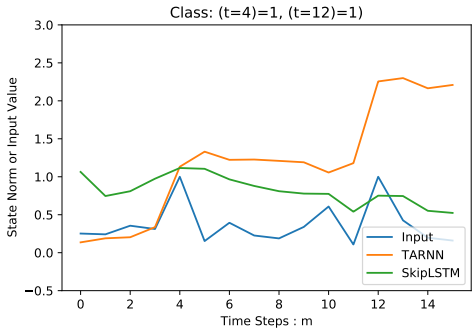
correct locations where the markers are placed and hence we get the hidden state transitions at these locations, and finally (c) we plot a the hidden state norms in order to demonstrate that SkipLSTM does focus at the input markers while TARNN ends up changing the hidden states at these locations.



(a)



(b)



(c)

Figure 3: Toy Example. (a) TARNN converges quickly to the 0.0 cross-entropy error. (b) shows time constant β along with the input, at locations $t = 4, 12$, both the input and time constants are in sync resulting in the state update while everywhere else the time constant does not allow the state to update (see s_m^1 state which captures the update or skip state part). (c) shows the norm of the hidden state for SkipLSTM and TARNN .

A.8 GOOGLE-30, HAR-2 DATASETS

In order to verify that our method works well for IoT tasks, we use popular datasets from previous works (Kusupati et al. (2018)). These datasets primarily focus on detecting activitiy embedded in a longer sequence. We pick two datasets namely: (a) HAR-2 Anguita et al. (2012), *i.e.* Human Activity Recognition from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone, and (b) Google-30 Warden (2018), *i.e.* detection of utterances of 30 commands plus background noise and

silence. For these tasks, light footprint of the model also becomes extremely important given that these models are deployed on resource constrained IoT devices.

Table 5 shows accuracy, model size, training time, inference time, and the number of parameters. TARNN beats the baselines in terms of test accuracy. TARNN has model size, inference time comparable to iRNN and hence well suited for IoT tasks.

Table 5: Results for Activity Recoginition (IoT) Datasets.

Data set	Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)	Test Time (ms)	#Params
HAR-2	FastRNN	94.50	29	0.063	**0.01**	7.5k
	LSTM	93.65	74	0.183	0.04	16k
	Antisymmetric	93.15	29	0.087	**0.01**	7.5k
	iRNN	96.30	18	**0.018**	0.03	4k
	TARNN	**96.59**	**17**	0.03	0.02	**3.7k**
Google-30	FastRNN	91.60	96	1.30	**0.01**	18k
	LSTM	90.31	219	2.63	0.05	41k
	Antisymmetric	90.91	64	0.54	**0.01**	12k
	iRNN	94.23	45	0.44	0.05	**8.5k**
	TARNN	**94.93**	**20**	**0.38**	0.01	9k

A.9 INFERENCE TIME

As the table 5 shows that the inference time for TARNN is similar to FastRNN and about at least one-half of the inference time for the LSTMs.