meta-RNN: Formulating Recurrent Neural Networks via Online Bilevel Optimization as Meta-Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper we consider the training of recurrent neural networks (RNNs) itself as a meta-learning problem, and propose a family of RNNs, namely meta-RNN, that can be formulated using online bilevel optimization. With the help of stochastic gradient descent (SGD), we manage to convert the meta-learning problem into an RNN where the feedforward and backpropagation solve the lower and upper-level optimization for learning hidden states and their hyperparameters, respectively. We prove that under mild conditions there is no vanishing or exploding gradient in training meta-RNN. Empirically we demonstrate our approach with superior performance on several benchmark datasets, with fewer parameters, less training data, and much faster convergence. **Code is attached in the supplementary file.**

1 Introduction

Training Stability in RNNs. Recurrent neural networks (RNNs) have achieved significant success in learning complex patterns for sequential input data. Due to the repeatability of network weights in the chain rule when computing gradients, vanishing or exploding gradients often occur in training RNNs, *i.e.,* the gradient magnitudes either too small or too large, leading to the well-known critical training stability issue [1]. To see this, suppose that we are optimizing the following RNN:

$$\min_{\omega,\theta} \mathbb{E}_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \ell(h_T, y; \omega), \text{ s.t. } h_t = f(h_{t-1}, \mathbf{x}_t; \theta), \forall \mathbf{x}_t \in x, \forall t \in [T], \tag{1}$$

where $(x, y) \in \mathcal{X} \times \mathcal{Y}$ denotes the training data with samples $x = \{\mathbf{x}_1, \cdots, \mathbf{x}_T\} \subseteq \mathbb{R}^d$ and label y, $h_t \in \mathbb{R}^D$ denotes the hidden state variable at the t-th time step with a predefined initial h_0, ℓ, f denote the loss and nonconvex transition functions parametrized by ω, θ, respectively, and $\mathbb{E}$ denotes the expectation operation. The gradient of ℓ *w.r.t.* θ per data in the RNN can be computed as follows:

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial h_T} \cdot \sum_{1\le t\le T} \left(\frac{\partial h_T}{\partial h_t} \frac{\partial h_t}{\partial \theta} \right), \text{ where } \frac{\partial h_T}{\partial h_t} = \prod_{t<k\le T} \frac{\partial h_k}{\partial h_{k-1}}. \tag{2}$$

The training challenge mainly comes from $\frac{\partial h_T}{\partial h_t}$, *i.e.,* $\|\frac{\partial h_T}{\partial h_t}\| \to 0$ leading to vanishing gradients and $\|\frac{\partial h_T}{\partial h_t}\| \to +\infty$ leading to exploding gradients, where $\|\cdot\|$ denotes the magnitude of a gradient.

Penalty/Lagrangian Methods. In fact, the learning problem in Eq. 1 is a constrained optimization problem with recursive equality constraints. To solve it, an intuitive idea is to relax it to an unconstrained optimization problem using, for instance, penalty methods [2] or (augmented) Lagrangian methods [3] that lift the constraints into the objective function. As demonstration, we list a relaxation of Eq. 1 using a penalty method as follows:

$$\min_{\omega,\theta,\{h_t\}} \mathbb{E}_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \left[\ell(h_T, y; \omega) + \sum_{t\in[T]} \lambda_t \left\| h_t - f(h_{t-1}, \mathbf{x}_t; \theta) \right\|^2 \right], \tag{3}$$

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

where $\lambda_t \geq 0, \forall t$ denotes a penalty coefficient. Now the gradient *w.r.t.* θ is only dependent on the second term, *i.e.,* least squares, with no training stability issue. Such relaxation has modified the RNN network architectures that leads to new challenges in such methods for RNN training, for instance:

- The size of variables $\{h_t\}$ increases linearly *w.r.t.* the sequence length T, leading to large memory.
- The learned parameters ω, θ may not work well in the original RNN at test time, due to the penalty.

How to solve these challenges in the relaxed training approaches still remains an open question [3].

Bilevel Optimization (BLO) & Ordinary Differential Equations (ODEs). Alternatively, based on a similar idea of penalty methods, a trivial yet equivalent modification of Eq. 1 is listed as Eq. 4 below, which essentially defines a bilevel optimization problem where one optimization problem is embedded within the other (*e.g.,* see [4, 5] and the references therein):

$$\min_{\omega,\theta} \mathbb{E}_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \ell(h_T, y; \omega), \text{ s.t. } h_t = \arg\min_{h} \|h - f(h_{t-1}, \mathbf{x}_t; \theta)\|^2, \forall t \in [T] \tag{4}$$

$$\equiv \min_{\omega,\theta} \mathbb{E}_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \ell(h_T, y; \omega), \text{ s.t. } h'_t = h_t - f(h_{t-1}, \mathbf{x}_t; \theta), \forall t \in [T], \tag{5}$$

where h'_t denotes the change rate of variable h_t over time in the *gradient flow*, and $h'_t = \mathbf{0}$ when h_t converges to a fixed point, if exists. In fact, some of recent ODE based RNNs have similar transition functions to Eq. 5. For instance, Kag et al. [6] proposed an incremental RNN (iRNN) with a transition function of $-\beta h'_t = \alpha\left(h_t + h_{t-1}\right) - f\left(h_t + h_{t-1}, \mathbf{x}_t; \theta\right), h_t(0) = \mathbf{0}$, and proved that there is no vanishing/exploding gradient in training iRNN. As we can see, the key difference between iRNN and Eq. 5 is that the current hidden state variable h_t is introduced into function f, leading to a sequence of updates for h_t through variable discretization. However, to the best of our knowledge, so far no work has been proposed to formulate RNNs using BLO.

RNNs, Bilevel Optimization, and Meta-Learning. Meta-learning often refers to a family of algorithms that can automatically learn the hyperparameters by themselves (*e.g.,* see [7, 8] and the references therein). RNNs have been used in meta-learning to model the hyperparameters in other algorithms. For instance, Ravi and Larochelle [9] proposed an LSTM-based meta-learner to efficiently update the learner's (*i.e.,* the original model) parameters using a small support set so that the learner can adapt to the new task quickly. Recently BLO regains attention in meta-learning. For instance, Franceschi et al. [10] proposed a framework based on bilevel programming to unify gradient-based hyperparameter optimization and meta-learning. Rajeswaran et al. [11] proposed an implicit MAML [12] algorithm that depends only on the solution to the lower-level optimization and not the path taken by the inner loop optimizer. Jenni and Favaro [13] proposed a regularization approach for training networks based on BLO that incorporates the principles of cross-validation. Mounsaveng et al. [14] proposed using online BLO to tune the hyperparameters for data augmentation.

Our Contributions. *In contrast to the literature, in this paper we consider the training of RNNs itself as a meta-learning problem.* Specifically, we propose a new family of RNNs, namely *meta-RNN*, that can be formulated using online bilevel optimization (OBLO), where we consider hidden state vectors as intermediate auxiliary variables that depend on the RNN model parameters, *i.e.,* the hyperparameters in meta-learning. These auxiliary variables are optimized in the lower-level problem of OBLO, and the model parameters as well as the predictor are learned by optimizing the outer objective function. We utilize stochastic gradient descent (SGD) and its momentum variants to solve such OBLO problems, where in return the optimization process of the lower-level problem can be (approximately) represented by meta-RNN. We also prove that under mild conditions, our meta-RNN can avoid vanishing or exploding gradients in training by selecting proper learning rates for the lower-level problem. We conduct comprehensive experiments on several benchmark datasets to evaluate our approach, and achieve superior results.

In summary, our key contributions in this paper are listed as follows:

- We propose a new family of RNNs, namely meta-RNN, that are the *first* in the literature, to the best of our knowledge, to formulate RNNs using online bilevel optimization as a meta-learning problem for streaming data.
- We prove that our networks manage to obtain good training stability by selecting proper learning rates for the lower-level problem to avoid vanishing and exploding gradients.
- We demonstrate superior performance empirically with fewer parameters, less training data, and much faster convergence.

2 Related Work

RNN Architectures. To address the training stability issue, there are significant amount of works on developing RNN architectures such as, just to name a few, long short-term memory (LSTM) [15], gated recurrent unit (GRU) [16, 17], unitary RNNs [18, 19, 20, 21], deep RNNs [22, 23, 24], linear RNNs [25, 26, 27], residual/skip RNNs [28, 29, 30, 31, 32], ODE RNNs [33, 34, 35, 32, 36, 37, 6, 38, 39, 40], and implicit RNN [41]. We summarize the literature as the following groups:

- *Recurrent units:* LSTM applies gate-controlled memory cells to mitigate the vanishing/exploding gradient issue in sequence-based tasks. GRU is another widely-used variant of RNNs, and similar to LSTM it can stabilize the training [42]. Recently, Jose et al. [43] proposed a flexible Kronecker recurrent unit (KRU) based on Kronecker product. Nguyen et al. [44] proposed MomentumRNN by integrating momentum into the recurrent unit.
- *Residual/Skip RNNs:* This family of RNNs feed-forward state vectors with the help of skip or residual connections to serve as a middle ground between feed-forward and recurrent models and to mitigate gradient decay. Such network architectures can also be used to search for equilibrium (or fixed) points in ODEs, such as iRNN [6].
- *Stable recurrent models:* Miller and Hardt [45] proved that stable recurrent neural networks are well approximated by feed-forward networks for the purpose of both inference and training by gradient descent. The analysis is based on the contraction assumption in the state transition function. Revay and Manchester [41] proposed an implicit model structure that allows for a convex parametrization of stable models using contraction analysis of non-linear systems. Collins et al. [46] showed experimentally that all common RNN architectures achieve nearly the same per-task and per-unit capacity bounds with careful training, for a variety of tasks and stacking depths.
- *Unitary and orthogonal RNNs:* These RNNs aim to preserve the norm of hidden features (*i.e.,* $\|\frac{\partial h_t}{\partial h_k}\|$ in Eq. 2) by controlling the eigenvalues, explicitly or implicitly, that has been studied extensively in recent years [18, 19, 20, 21, 40, 47, 48, 49, 50, 51]. For instance, Lezcano-Casado and Martınez-Rubio [48] proposed expRNN by performing the first-order optimization with orthogonal and unitary constraints based on a parametrization stemming from Lie group theory through the exponential map.
- *RNNs with equilibrium hidden states:* In such RNNs, the hidden states are represented by the equilibrium points, if exist, of certain dynamical systems defined by ODEs, for instance. Typical works include AntisymmetricRNNs [35], FastRNN [52], iRNN [6], Lipschitz RNN [40]. It seems that all these works can prove good training stability from different perspectives such as generalization bound or norm of gradients, with empirical demonstration.
- *Deep RNNs:* RNNs are inherently deep in time. Inspired by this property, researchers are seeking to develop new networks to investigate the benefits of depth in space of RNN architectures. For instance, [53] combined multiple recurrent levels on the basis of bi-directional LSTM [54, 55] to improve RNN performance in speech recognition task. Another study in [56], with a deeper analysis of the different emergent time scales, also proposed a similar stacking architecture. Some deep RNNs have been proposed in the literature as well [57, 58, 59, 60, 61, 62, 22, 63, 64, 65].

Optimization for RNNs. Same as training other deep networks, SGD is widely used in training RNNs, but there is no guarantee of good training stability. In contrast, some works focus on optimization in RNNs to stabilize gradients. Typical works include truncated backpropagation through time (TBPTT) [66], real-time recurrent learning (RTRL) [67], Frank-Wolfe algorithm [68], gradient clipping [1, 64, 69], identity or orthogonal weight initialization [70, 18, 19, 71, 51, 72, 73], weight matrix reparametrization [74]. Different from these works, we propose using OBLO to interpret RNNs and further train them based on a two-loop algorithm where the inner loop solves the OBLO problems and the outer loop optimizes the model parameters for RNNs.

Training Stability Analysis in RNNs. The stability analysis of different network architectures often comes with the eigenvalues of the Jacobian of the hidden state dynamics, because RNNs can be considered as dynamical systems. For instance, Engelken et al. [75] studied the Lyapunov spectra of chaotic recurrent neural networks. Vogt et al. [76] proposed using Lyapunov exponents to understand the information propagation in RNNs, but unfortunately there is no discussion on how to introduce such nice Lyapunov stability into the development of RNNs. Drgona et al. [77] proposed viewing neural networks from a dynamical systems perspective as pointwise affine maps. However, the theoretical results are adapted from dynamical system analysis and the assumptions for deep neural

Algorithm 1 Training algorithm for meta-RNN

Input : loss function ℓ parametrized by ω, state transition function F parametrized by θ, step size η, training data $\mathcal{X}$ and labels $\mathcal{Y}$

Output : minimizer ω^*, θ^*

Randomly initialize ω^*, θ^*;

repeat

  /* lower-level optimization: RNN inference */

  foreach $x \in \mathcal{X}$ **do**

    $\mathrm{h}_0 \leftarrow \mathbf{0}$;

    foreach $t \in [T]$ **do**

      $\mathrm{h}_t \leftarrow h_{t-1} - \eta \nabla F\left(h_{t-1}, \mathbf{x}_t; \theta^*\right)$; // Or momentum SGD using Eq. 8 or 9

    end

  end

  /* upper-level optimization: hyperparameter learning */

  $\omega^*, \theta^* \in \arg\min_{\omega,\theta} \mathbb{E}_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \ell\left(h_T(\theta), y; \omega\right)|_{\omega=\omega^*,\theta=\theta^*}$,

until *Converge*;

return ω^*, θ^*;

networks are too strong to be met in practice. Tuor et al. [78] proposed an ODE based network implementation to guarantee stability as well as incorporating prior knowledge. Recently, Ribeiro et al. [79] proposed analyzing RNN training using attractors and smoothness as alternatives.

3 meta-RNN

3.1 Online Bilevel Optimization Formulation

Recall that the goal of RNNs is to learn discriminant representations for different data sequences, regardless of how the intermediate hidden states are defined. In our approach, we first assume that a data sequence $x = \{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$ is generated by a certain stochastic process with distribution $\mathcal{P}_y$ that is conditional on the label y of x. That is, each $\mathbf{x}_t, \forall t \in [T]$ is sampled from $\mathcal{P}_y$, *i.e.,* $\mathbf{x}_t \sim \mathcal{P}_y$. We observe that this assumption can work well in practice, even for the sequences with strong dependencies among samples. Based on this assumption, we propose the following online bilevel optimization (OBLO) for learning meta-RNN with streaming data:

$$\min_{\omega,\theta} \mathbb{E}_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \ell\left(h^*, y; \omega\right), \text{s.t. } h^* \in \arg\min_{h\in\mathcal{H}} \mathbb{E}_{\mathbf{x}_t\sim\mathcal{P}_y} F\left(h, \mathbf{x}_t; \theta\right), h_0 = \mathbf{0}, \forall \mathbf{x}_t \in x, \forall t \in [T], \quad (6)$$

where F is a proper, differentiable everywhere, and lower-bounded function in the feasible space $\mathcal{H}$.

As we can see, the lower-level optimization in our formulation aims to learn the data representations h on-the-fly, while the upper-level optimization aims to learn the hyperparameter θ of h as well as the predictor ω simultaneously. This is why we name our approach as meta-RNN. Different from Eq. 4, for instance, our approach utilizes the stationary points of function F as the representations of data sequences, and θ is learned so that such stationary points can be separated well for prediction. Similar ideas have been explored in the orthogonal RNNs such as expRNN [48] where the transition functions are nonexpansive *w.r.t.* hidden states (but no guarantee of convergence). It has been demonstrated that such nonexpansive constraints in learning can significantly improve the performance of RNNs. In this paper we further demonstrate that such convergence in the lower-level optimization can not only improve performance but also accelerate the training (see our experimental section).

3.2 Learning

Lower-level Optimization. We show our general training algorithm in Alg. 1, where the lower-level optimization is solved using SGD. We can also employ momentum SGD as our solver to train meta-RNN. Specifically, below we list three popular variants of SGD, namely vanilla SGD, stochastic heavy-ball (SHB) momentum, and stochastic Nesterov accelerated gradient (SNAG) momentum:

$$\text{SGD: } h_t = h_{t-1} - \eta \nabla F_t, h_0 = \mathbf{0}, h^* = h_T, \ \forall t \in [T], \quad (7)$$

$$\text{SHB: } \tilde{h}_t = \mu \tilde{h}_{t-1} - \eta \nabla F_t, h_t = h_{t-1} + \tilde{h}_t, \tilde{h}_0 = h_0 = \mathbf{0}, h^* = h_T, \ \forall t \in [T], \quad (8)$$

$$\text{SNAG: } \tilde{h}_t = h_{t-1} - \eta \nabla F_t, h_t = \tilde{h}_t + \mu(\tilde{h}_t - \tilde{h}_{t-1}), \tilde{h}_0 = h_0 = \mathbf{0}, h^* = h_T, \ \forall t \in [T], \quad (9)$$

where ∇ denotes the gradient operator, $\nabla F_t = \nabla_h F\left(h_{t-1}, \mathbf{x}_t, \theta\right)$ denotes the current gradient *w.r.t.* h, $\eta \geq 0$ denotes the step size, and $\mu \in [0, 1]$ denotes the momentum parameter. Note that SHB and SNAG is identical to SGD when $\mu = 0$.

RNN Interpretation. Eq. 7, 8 and 9 define three different iterative methods that take the previous hidden state as input and output the current hidden state, leading to three different meta-RNN network architectures (see Fig. 2). Note that our meta-RNN shares similarities in some recent RNNs such as FastRNN [52] and iRNN [6] that have skip connections in the architectures and have been demonstrated to be trained stably. Recently, momentumRNN [44] was proposed by drawing a connection between the dynamics of hidden states and GD. It is clear, however, that in terms of formulation the "momentum" here is only used for accumulating the weighted data samples without any purpose in optimization. In other words, its update rule has nothing to do with GD or momentum variants. In contrast, in our meta-RNN we utilize first-order methods to really solve the lower-level optimization problems, which makes the gradient term in our meta-RNN analytically integralable.

Instantiation. In this paper, we demonstrate our approach using the following simple function:

$$F(h, \mathbf{x}_t; \theta) \overset{def}{=} \frac{1}{2} \left\| \alpha h - \phi\left(\mathbf{U}^T h + \mathbf{V}^T \mathbf{x}_t + \mathbf{b}\right) \right\|^2, \ h \in \mathbb{R}^D, \alpha \in \mathbb{R}, \tag{10}$$

where $\mathbf{U} \in \mathbb{R}^{D \times D}, \mathbf{V} \in \mathbb{R}^{d \times D}, \mathbf{b} \in \mathbb{R}^D, \alpha$ together denote the model parameters θ, and $\phi : \mathbb{R}^D \to \mathbb{R}^D$ denotes an entry-wise nonconvex and differentiable function such as ReLU and $\tanh$. Here Eq. 10 borrows the transition function in vanilla RNNs that aim to enforce the hidden state vectors to satisfy certain properties induced by ϕ. Correspondingly, the gradient of this model is list as follows:

$$\nabla F_t = \left(\alpha \mathbf{I} - \nabla_h \phi_t\right)\left(\alpha h_{t-1} - \phi_t\right), \phi_t = \phi\left(\mathbf{U}^T h_{t-1} + \mathbf{V}^T \mathbf{x}_t + \mathbf{b}\right), \tag{11}$$

where $\mathbf{I} \in \mathbb{R}^{D \times D}$ denotes the identity matrix. Now we can substitute this gradient into Eq. 7, 8 and 9, respectively, to solve the lower-level optimization defined in Eq. 10.

Convergence. Under some mild conditions, to minimize *strongly-convex and smooth* objectives, all the three stochastic optimizers can converge with the same rates as their deterministic counterparts [80]. In the *convex and smooth* setting, recent results in [81] proved that their convergence rates can be arbitrarily close to $o(1/\sqrt{t})$, and can be exactly $o(1/t)$ in the overparametrized case. In both *nonconvex and smooth* and *nonconvex and nonsmooth* settings, the convergence rates are changed to $O(1/\sqrt{t})$ [82, 83]. Overall, our solvers should be able to locate solutions around stationary points.

To verify the convergence of SGD, SHB and SNAG for our lower-level optimization with Eq. 10, we generate some toy data and illustrate the results in Fig. 1. We sample $\mathbf{U}, \mathbf{V}, \mathbf{b}$ from a uniform distribution between -1 and 1, set $\alpha = 1$, and fix them for all the settings. We first generate i.i.d. data from the same uniform distribution, and then generate another set of data based on a multi-variant normal distribution. As we see in Fig. 1, for both models all the three optimizers can converge over time, no matter how the samples are sampled. Besides, SHB and SNAG behave almost identically with much faster convergence speed than SGD. Note that such observations hold not only for the toy data but also for the benchmark data in our experiments.

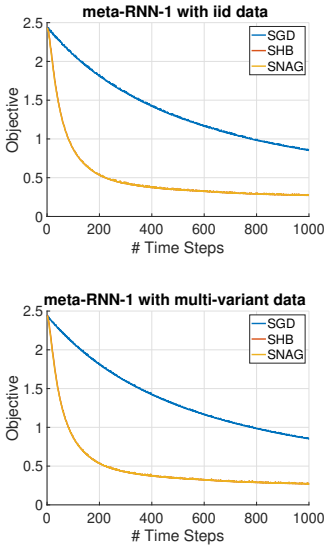


Figure 1: Convergence illustration of SGD, SHB and SNAG for solving the lower-level optimization with F in Eq. 10 using a sequence of i.i.d. samples or multi-variant samples, where the curves for SHB and SNAG are significantly overlapped with each other.

Network Architectures. We illustrate three architectures of meta-RNN in Fig. 2, where each directed edge towards $\bigoplus$ is associated with a certain learnable weight that is either 1 or related to one of α, μ, η in Eq. 7, 8, 9 and 10. As we see, Fig. 2(a) is essentially a variant of ResNet [84], and Fig. 2(b) and (c) are similar to dual-path networks (DPN) [85] where information is propagated between two parallel paths. Note that though both (b) and (c) come from momentum SGD, the distributions of h and $\tilde{h}$ are totally different due to their weighting strategies for gradients. The output h_T is fed to a predictor parametrized by ω in Eq. 6. In our

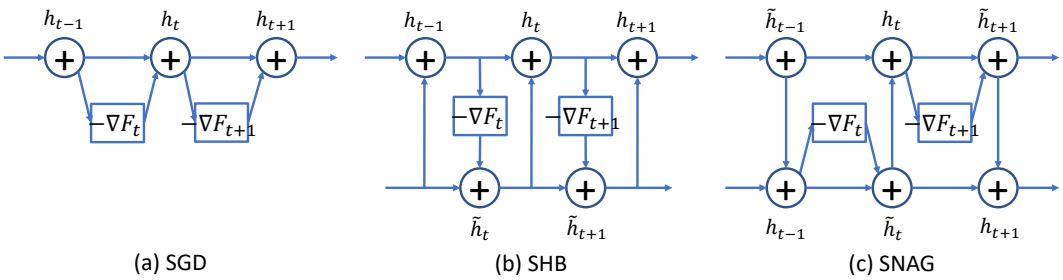


Figure 2: Illustration of meta-RNN architectures using the optimizers in Eq. 7, 8 and 9, respectively.

implementation we use a linear function that is a fully-connected layer in the network. Such networks can implement Alg. 1 effectively and efficiently.

Remark. We also tried second-order optimizers such as stochastic Gauss-Newton methods [86] to solve the lower-level optimization, but unfortunately in our implementation we did not observe consistent convergence behavior as first-order methods. One of the key reasons is that the estimation of the Hessian matrix is quite unstable, leading to large variations in updates. Therefore, in this paper we only show the results with first-order methods.

3.3 Analysis

Theorem 1 (Upper Bound of Hidden Feature Distance)**.** *Let* $\forall t \in [T], x = \{\mathbf{x}_t\}, x' = \{\mathbf{x}'_t\}$ *and* $\{h_t\}, \{h'_t\}$ *denote two data sequences and their corresponding hidden features, repectively. Suppose that function* $F(h, \mathbf{x}_t; \theta)$ *is differentiable,* L_h*-smooth* w.r.t. h, i.e., $\|\nabla F(h, \mathbf{x}_t; \theta) - \nabla F(h', \mathbf{x}_t; \theta)\| \leq L_h \|h - h'\|$, *and its gradient* w.r.t. $\mathbf{x}_t$ *is* L_x*-Lipschitz,* i.e., $\|\nabla F(h, \mathbf{x}_t; \theta) - \nabla F(h, \mathbf{x}'_t; \theta)\| \leq L_x \|\mathbf{x}_t - \mathbf{x}'_t\|$. *Then for SGD in Eq. 7, we have*

$$\|h_t - h'_t\| \leq \sum_{k \in [t]} \eta L_x (1 + \eta L_h)^{t-k} \|\mathbf{x}_k - \mathbf{x}'_k\| \leq t\eta\gamma_t L_x = O(t\gamma_t), \tag{12}$$

where $\gamma_t = \max_{k \in [t]} \|\mathbf{x}_k - \mathbf{x}'_k\|$ *and* $\eta L_h \ll 1$.

Please refer to our proof in the appendix. In fact, this result provides us a hypothesis on our faster convergence empirically (see our experiments) that is beneficial from the factor γ_t in the upper bound. Since we assume $\mathbf{x}_t \sim \mathcal{P}_y$, it is very likely that γ_t for the data from the same class will be smaller than that from different classes, leading to clusters for different classes. We illustrate this intuition in Fig. 3, where red and green colors denote two different classes. The smaller upper bound encourages the data from the same class to form a cluster that can be easily separate from the data from different classes. This behavior would accelerate the learning of discriminative features in the networks, leading to fast convergence.

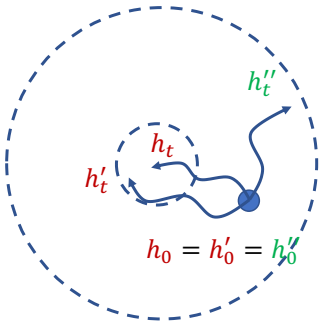


Figure 3: Illustration of clustering in the hidden feature space based on Thm. 1.

Corollary 1. *For SHB in Eq. 8 and SNAG in Eq. 9, supposing that the assumptions in Thm. 1 hold as well, then we still have* $\|h_t - h'_t\| \leq O(t\gamma_t), \forall t \in [T]$.

To see this, we can simply scale the upper bound with a factor of T.

Theorem 2 (Training Stability)**.** *Suppose that function* F *is differentiable and* L*-smooth* w.r.t. h, i.e., $\nabla^2 F \preceq L\mathbf{I}$ *or* $\|\nabla^2 F\| \leq L$ *where* $\mathbf{I}$ *denotes the identity matrix. Then by properly choosing* $\eta \in (0, \frac{\epsilon}{TL}]$ *where* T *is the sequence length and* $\epsilon \in (0, 1)$ *is a sufficiently small constant, we can guarantee that there is no vanishing/exploding gradient for training meta-RNN using SGD in Eq. 7.*

Please refer to our proof in the appendix. Note that the choice of η should also guarantee the convergence of the lower-level optimization.

Corollary 2. *For SHB in Eq. 8 and SNAG in Eq. 9, supposing that the assumptions in Thm. 2 hold, we still can guarantee that there is no vanishing and exploding gradient when training meta-RNN.*

To see this, we can rewrite the update rules as accumulation of the previous gradients. Then based on the fact that $\frac{\partial h_{k-1}}{\partial h_{t-1}} = \mathbf{0}, \forall k \in [t-1]$ holds, we can still achieve $\frac{\partial h_t}{\partial h_{t-1}} \propto \mathbf{I} - \eta \nabla F_t$ that leads to the same conclusion in Thm. 2.

4 Experiments

We conduct comprehensive experiments to demonstrate the effectiveness of meta-RNN from the following three perspectives, compared with other baseline RNNs:

- *Fewer parameters:* We simply set $\mathbf{U} = \beta \mathbf{I}$ where $\beta \in \mathbb{R}$ is a learnable parameter. In this way, we reduce the model size (including $\mathbf{U}, \mathbf{V}, \mathbf{b}, \alpha, \mu, \eta$) from $D(D \times d + 1) + 3$ to $D(d + 1) + 4$.
- *Less training data:* We randomly select 1% and 10% of the original training data to train different models and test them using the original test data.
- *Faster convergence:* We illustrate the training loss curves as well as test accuracy curves.

Datasets. We evaluate our method on long-sequence benchmarks with varying difficulties, and list the statistics of these datasets in Table 1. We strictly follow the experimental settings in the previous works (*e.g.,* [6, 52]) for fair comparison.

Table 1: Statistics and dimension of hidden state vectors in each benchmark dataset.

Dataset	Statistics					Dim.
	#Train	#Test	#Time	#Feat.	#Cls.	D
Pixel-MNIST	60k	10k	784	1	10	128
Perm-MNIST	60k	10k	784	1	10	128
HAR-2	7.3k	2.9k	128	9	2	128

Specifically, *Pixel-MNIST* refers to pixel-by-pixel sequences of images in MNIST [87] where each 28 $\times$ 28 image is flattened into a 784 time-step sequence vector, while a random permutation to the Pixel-MNIST is applied to generate a harder time sequence dataset as *Perm-MNIST*. Both datasets are normalized as zero mean and unit variance during training and prediction. *HAR-2* [32] was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone for human activity recognition tasks with zero mean and unit variance.

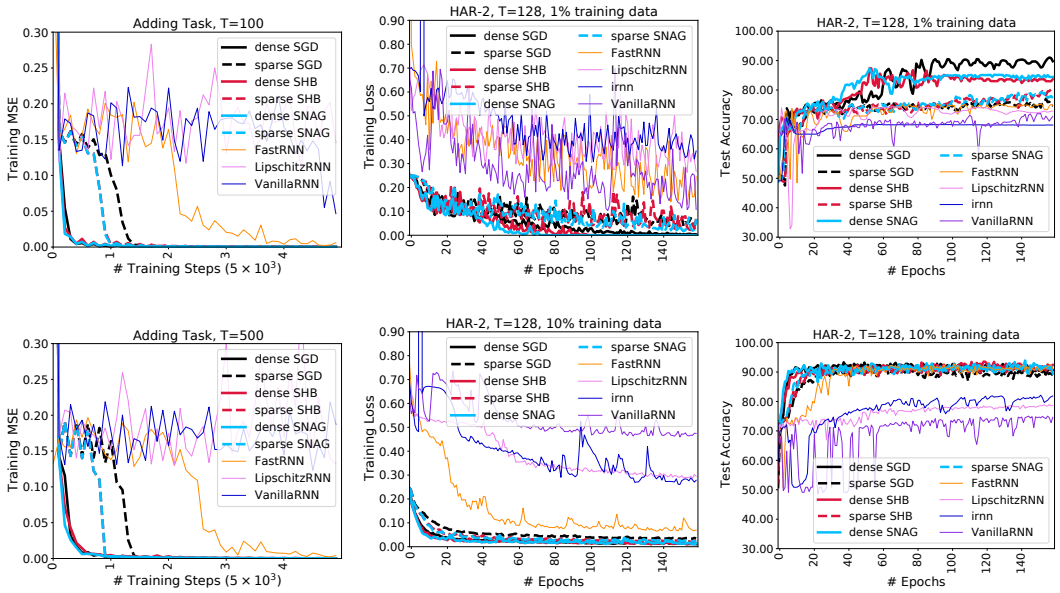
Baseline Algorithms. We compare our results with baselines including vanilla RNN, LSTM [15], AntisymmetricRNN [35], FastRNN [52], FastGRNN [52], ShaRNN [65], IndRNN [64], iRNN [6], LipschitzRNN [40] and MomentumRNN [44]. Note that using the public code we have verified the results of each competitor on the datasets that were reported in the references. For simplicity and consistency we cite the numbers from the references, if exist, otherwise, we report our reproduced results with best tuned hyperparameters instead.

Training & Testing Protocols. In our implementation we utilize ReLU as our activation function (*i.e.,* ϕ in Eq. 10), as we observed that this activation works better than others such as $\tanh$ in terms of both accuracy and convergence. This observation is consistent with the state-of-the-art methods. We replicate the same benchmark training/testing split with 20% of training data for validation to tune hyperparameters. Then we retrain the models using best tuned hyperparameters using the full training set and test them on the test set. We report our results (*i.e.,* accuracy and running time) over three trials with randomization wherever needed such as parameter initialization. All the experiments were run on an Nvidia GeForce RTX 2080 Ti GPU server.

Hyperparameters. We use the grid search to fine-tune the hyperparameters of each baseline as well as ours on the validation datasets whenever is necessary. We follow [52, 6, 35] to set the hidden feature dimension D, as listed in Table 1. We further set $\eta = 10^{-3}$ in Eq. 7, 8 and 9 in all the experiments as we observe that this number works well and consistently that leads the lower-level optimization to converge. The batch size of 64 is used across all the datasets for all the methods. Adam [88] is used as the optimizer for all the methods. The learning rate for training meta-RNN architectures is always initialized to 10^{-3} with linear scheduling of weight decay. For benchmark datasets, we utilize the cross-entropy (CE) loss for ℓ in Eq. 6 that is consistent with other RNNs.

4.1 Toy Example: Adding Task

We first demonstrate our method on the Adding Task that is widely used for RNN evaluation. We strictly follow the adding task defined by [18] to generate the data using the public code. There are two sequences with length $T = 100, 500$, respectively. The first sequence is sampled uniformly at



(a) Adding Task (b) HAR-2: Training Loss (c) HAR-2: Test Accuracy

Figure 4: **(a)** Training loss on the Adding Task. **(b-c)** Training loss and test accuracy on HAR-2.

Table 2: Performance summary of different variants of our meta-RNN on HAR-2. The training/test time is for per batch with 64 sequences.

	dense SGD	dense SHB	dense SNAG	sparse SGD	sparse SHB	sparse SNAG
1% data acc. (%)	**90.25± 0.71**	86.59±0.74	87.19±0.14	**79.18±1.45**	78.55±1.45	77.89±1.41
10% data acc. (%)	93.25± 0.07	93.44±0.67	**93.52±0.38**	90.76±0.32	**91.98±0.49**	91.89±0.59
training time (s)	**0.60±0.07**	0.65±0.06	0.67±0.10	**0.54±0.02**	0.55±0.04	0.58±0.04
test time (s)	**0.30±0.02**	0.33±0.05	0.31±0.02	0.28±0.01	**0.26±0.01**	0.27±0.01

random $\mathcal{U}[0, 1]$. The second sequence is filled with 0 except for two entries of 1. The two entries of 1 are located uniformly at random position i_1, i_2 in the first half and second half of the sequence. The prediction value is the sum of the first sequence between $[i_1, i_2]$. The ground-truth mean squared error is 0.167 when a model simply guesses 1 as the output regardless of the input sequence. We use the value 0.167 as the baseline. Mean-square-error (MSE) loss is used in training, and the hidden feature dimension is $D = 128$.

Fig. 4(a) summarizes our comparison results. As we see, in both figures all the variants of our meta-RNN ("dense" for $\mathbf{U}$ as a dense matrix and "sparse" for $\mathbf{U} = \beta\mathbf{I}$) converge significantly faster than the other competitors. Clearly, the dense variants converge faster than the sparse counterparts, and the momentum variants converge faster than the SGD counterparts. With the increase of T, the convergence behavior of our meta-RNN does not vary a lot in terms of number of training steps.

4.2 Case Study on HAR-2

To better study the properties of our meta-RNN, we conduct comprehensive experiments on HAR-2 dataset, and summarize our comparison in Fig. 4(b-c) and Table 2. Overall, all of our models perform significantly better than the other competitors with much faster convergence when the training data is less. Specifically,

- *Fewer parameters:* On this dataset our sparse models contain only 1.5K parameters, in contrast to the dense models with 17.9K parameters, that is more than one order of magnitude reduction. In terms of convergence, sparse models are slightly worse than the dense models, leading to worse test accuracy as well. From Table 2, we can see that the running time of sparse models in both training and testing is significantly shorter then dense models, with smaller std as well (*i.e.,* better stability).

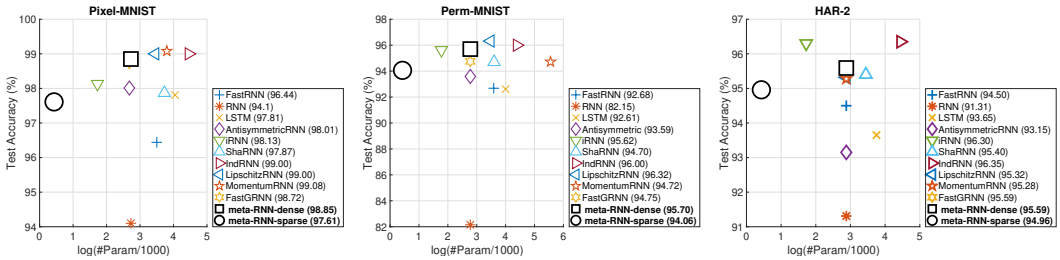


Figure 5: Benchmark comparison using different RNNs.

In summary, our sparse models demonstrate strong potential in hardware implementation under resource-limited circumstances, achieving similar performance to those with larger model sizes.

- *Less training data:* One of the benefits from meta-learning is to learn models using less data. Our results exactly validate this. Using either 1% or 10% of the original training data, our method can outperform the competitors significantly by $\sim$10% in terms of test accuracy. Also in such cases, our method converges faster when data is more, and our dense models seem to consistently work better than sparse ones. The number of data has more impact on sparse models than dense models (accuracy increases by $\sim$13% and $\sim$3%, respectively).
- *Faster convergence:* Similar to the Adding Task, here we clearly see the much faster convergence of all of our models than the competitors, in terms of both training loss and the test accuracy. We hypothesize that such behavior has strong correlation with the lower-level optimization that enforces data to cluster, as shown in Thm. 1.

Also, from Table 2 we can see that the three optimizers for the lower-level optimization often achieve similar performance with small margins, especially when the training data is sufficiently large. Momentum variants bring more computation in both feedforward and backpropagation in training networks and inference. Therefore, in practice SGD based meta-RNN seems to be sufficient. Note that we have similar observations on the other datasets as well.

4.3 State-of-the-art Comparison

We compare our performance with other proposed baseline RNNs in Fig. 5, where "meta-RNN-dense" and "meta-RNN-sparse" refer to our dense and sparse models based on the SGD lower-level optimizer in Eq. 7, respectively. The x-axis measures the number of parameters in each network (including the classifier) in log-scale. As we see, our sparse model has orders of magnitude fewer parameters but always produces comparable accuracy to the state-of-the-art. For instance, FastGRNN [52] was designed as a tiny-size RNN. However, our sparse model is $\sim$10x smaller with only $\sim$1% accuracy loss. On the other hand, our dense model can always achieve superior performance. Recall that the F function in Eq. 10 is so simple that its variant is used in vanilla RNNs, while all the methods that achieve better accuracy, *e.g.,* IndRNN [64], have very complicated hidden state transition functions, even with larger model sizes. In summary, our meta-RNN has great potential to achieve the state-of-the-art using simpler transition functions and smaller model sizes.

5 Conclusion

In this paper we propose a new family of RNNs, namely meta-RNN, that can be formulated using online bilevel optimization as a meta-learning problem. Using SGD we convert the meta-learning problem into an RNN architecture, where the lower-level optimization is mapped to the sub-network for learning hidden features and the upper-level optimization defines the predictor based on the learned features. We prove that under mild conditions there is no vanishing/exploding gradient in the training of our meta-RNN, and thus our training is easy and stabilized. Furthermore, we demonstrate the superior performance of meta-RNN on several benchmark datasets with faster convergence, even based on fewer parameters and less training data.

Currently, we lack strong theorems to explain the fast convergence behavior of meta-RNN, though we have the hypothesis of data clustering, due to the lower-level optimization, that may significantly contribute to this. Also, our current setting has difficulty in adapting to deep RNNs such as IndRNN. In our future work, we will explore more on these.

References

[1] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[2] Ziming Zhang and Matthew Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *NeurIPS*, 2017.

[3] Fangda Gu, Armin Askari, and Laurent El Ghaoui. Fenchel lifted networks: A lagrange relaxation of neural network training. In *International Conference on Artificial Intelligence and Statistics*, pages 3362–3371. PMLR, 2020.

[4] Luís N. Vicente and Paul H. Calamai. Bilevel and multilevel programming: A bibliography review. *Journal of Global Optimization*, 5(3):291–306, 1994.

[5] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2): 276–295, 2017.

[6] Anil Kag, Ziming Zhang, and Venkatesh Saligrama. Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*, 2020.

[7] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[8] Mike Huisman, Jan N van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, pages 1–59, 2021.

[9] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[10] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018.

[11] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. 2019.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[13] Simon Jenni and Paolo Favaro. Deep bilevel learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 618–633, 2018.

[14] Sayprasseuth Mounsaveng, Issam Laradji, Ismail Ben Ayed, David Vazquez, and Marco Pedersoli. Learning data augmentation with online bilevel optimization for image classification. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1691–1700, 2021.

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

[16] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[17] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and Trainability in Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1611.09913, November 2016.

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

[19] 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.

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

[21] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Advances in Neural Information Processing Systems 30*, pages 4785–4795. 2017.

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

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

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

[25] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *CoRR*, abs/1611.01576, 2016. URL http://arxiv.org/abs/1611.01576.

[26] 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.

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

[28] 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.

[29] 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.

[30] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.

[31] 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. *arXiv preprint arXiv:1708.06834*, 2017.

[32] 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.

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

[34] Murphy Yuezhen Niu, Lior Horesh, and Isaac Chuang. Recurrent neural networks in the eye of differential equations. *arXiv preprint arXiv:1904.12933*, 2019.

[35] 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.

[36] 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.

[37] 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.

[38] T. Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (coRNN): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2021.

[39] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019.

[40] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, and Michael W Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.

[41] Max Revay and Ian Manchester. Contracting implicit recurrent neural networks: Stable models with improved trainability. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, pages 393–403, 2020.

[42] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[43] Cijo Jose, Moustapha Cissé, and Francois Fleuret. Kronecker recurrent units. In *International Conference on Machine Learning*, pages 2380–2389. PMLR, 2018.

[44] Tan M Nguyen, Richard G Baraniuk, Andrea L Bertozzi, Stanley J Osher, and Bao Wang. Momentumrnn: Integrating momentum into recurrent neural networks. *arXiv preprint arXiv:2006.06919*, 2020.

[45] John Miller and Moritz Hardt. Stable recurrent models. In *International Conference on Learning Representations*, 2019.

[46] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

[47] 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 *Advances in Neural Information Processing Systems*, pages 13613–13623, 2019.

[48] Mario Lezcano-Casado and David Martınez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pages 3794–3803, 2019.

[49] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978, 2018.

[50] Kehelwala DG Maduranga, Kyle E Helfrich, and Qiang Ye. Complex unitary recurrent neural networks using scaled cayley transform. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4528–4535, 2019.

[51] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *International Conference on Machine Learning*, pages 2401–2409. PMLR, 2017.

[52] 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*, pages 9017–9028, 2018.

[53] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[54] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

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

[56] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems*, pages 190–198, 2013.

[57] Wen-Yuan Chen, Yuan-Fu Liao, and Sin-Horng Chen. Speech recognition with hierarchical recurrent neural networks. *Pattern Recognition*, 28(6):795–805, 1995.

[58] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499, 1996.

[59] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. Sequence labelling in structured domains with hierarchical recurrent neural networks. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, 2007.

[60] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[61] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[62] Herbert Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. Technical report, Jacobs University Bremen, 2007.

[63] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning (ICML)*, 2014.

[64] 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*, pages 5457–5466, 2018.

[65] 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*, pages 12896–12906, 2019.

[66] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[67] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[68] Yun Yue, Ming Li, Venkatesh Saligrama, and Ziming Zhang. Rnn training along locally optimal trajectoriesvia frank-wolfe algorithm. *arXiv preprint arXiv:2010.05397*, 2020.

[69] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations*, 2020.

[70] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[71] Cijo Jose, Moustpaha Cisse, and Francois Fleuret. Kronecker recurrent units. *arXiv preprint arXiv:1705.10142*, 2017.

[72] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in neural information processing systems*, pages 4880–4888, 2016.

[73] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning*, pages 3570–3578, 2017.

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

[75] Rainer Engelken, Fred Wolf, and LF Abbott. Lyapunov spectra of chaotic recurrent neural networks. *arXiv preprint arXiv:2006.02427*, 2020.

[76] Ryan Vogt, Maximilian Puelma Touzel, Eli Shlizerman, and Guillaume Lajoie. On lyapunov exponents for rnns: Understanding information propagation using dynamical systems tools. *arXiv preprint arXiv:2006.14123*, 2020.

[77] Jan Drgona, Elliott Skomski, Soumya Vasisht, Aaron Tuor, and Draguna Vrabie. Spectral analysis and stability of deep neural dynamics. *arXiv preprint arXiv:2011.13492*, 2020.

[78] Aaron Tuor, Jan Drgona, and Draguna Vrabie. Constrained neural ordinary differential equations with stability guarantees. *arXiv preprint arXiv:2004.10883*, 2020.

[79] António H Ribeiro, Koen Tiels, Luis A Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In *International Conference on Artificial Intelligence and Statistics*, pages 2370–2380. PMLR, 2020.

[80] Mahmoud Assran and Michael Rabbat. On the convergence of nesterov's accelerated gradient method in stochastic settings. *arXiv preprint arXiv:2002.12414*, 2020.

[81] Othmane Sebbouh, Robert Gower, and Aaron Defazio. Almost sure convergence rates for stochastic gradient descent and stochastic heavy ball. 2021.

[82] Yan Yan, Tianbao Yang, Zhe Li, Qihang Lin, and Yi Yang. A unified analysis of stochastic momentum methods for deep learning. *arXiv preprint arXiv:1808.10396*, 2018.

[83] Vien Mai and Mikael Johansson. Convergence of a stochastic gradient method with momentum for non-smooth non-convex optimization. In *International Conference on Machine Learning*, pages 6630–6639, 2020.

[84] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[85] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4470–4478, 2017.

[86] Quoc Tran-Dinh, Nhan Pham, and Lam Nguyen. Stochastic Gauss-Newton algorithms for nonconvex compositional optimization. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9572–9582, 2020.

[87] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

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

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]