2020 25th International Conference on Pattern Recognition (ICPR)
Milan, Italy, Jan 10-15, 2021

RNN Training along Locally Optimal Trajectories via Frank-Wolfe Algorithm

Yun Yue*, Ming Li*, Venkatesh Saligrama† and Ziming Zhang*
* Worcester Polytechnic Institute, Worcester, MA 01609
† Boston University, Boston, MA 02215
Email: {yyue, mli12, zzhang15}@wpi.edu, srv@bu.edu

2020 25th International Conference on Pattern Recognition (ICPR) | 978-1-7281-8808-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICPR48806.2021.9412188

***Abstract*—We propose a novel and efficient training method for RNNs by iteratively seeking a local minima on the loss surface within a small region, and leverage this directional vector for the update, in an outer-loop. We propose to utilize the Frank-Wolfe (FW) algorithm in this context. Although, FW implicitly involves normalized gradients, which can lead to a slow convergence rate, we develop a novel RNN training method that, surprisingly, even with the additional cost, the overall training cost is empirically observed to be lower than back-propagation. Our method leads to a new Frank-Wolfe method, that is in essence an SGD algorithm with a restart scheme. We prove that under certain conditions our algorithm has a sublinear convergence rate of $O(1/\epsilon)$ for ϵ error. We then conduct empirical experiments on several benchmark datasets including those that exhibit long-term dependencies, and show significant performance improvement. We also experiment with deep RNN architectures and show efficient training performance. Finally, we demonstrate that our training method is robust to noisy data.**

I. Introduction

Consider the problem of training RNNs based on minimizing empirical risk over minibatches, $\mathcal{B}$ that are sampled uniformly at random from training examples $\mathcal{X} \times \mathcal{Y}$ of feature-label pairs (x, y) over M time steps. Let us denote the instance at m-step as x_m, and the hidden state as $\mathbf{z}_m$. The batch-averaged empirical risk can be written as:

$$\min_{\omega} \left[F(\omega) \overset{def}{=} \mathbb{E}_{\mathcal{B}\sim\mathcal{X}\times\mathcal{Y}} f(\mathcal{B};\omega) \overset{def}{=} \sum_{(x,y)\sim\mathcal{B}} \ell(y, \mathbf{z}_M; \omega_\ell) \right]$$
$$\text{s.t. } \mathbf{z}_m = h(x_m, \mathbf{z}_{m-1}; \omega_h), \ \forall \ m \in [M], \quad (1)$$

where $\omega = \{\omega_\ell, \omega_h\}$ denotes the RNN weights, ℓ, h denotes the loss and (nonconvex) state transmission functions parameterized by ω_ℓ, ω_h, and $\mathbb{E}$ denotes the expectation operator.

Vanishing and Exploding Gradients. Training stability of RNNs is regarded as a fundamental aspect, attracting much attention in the literature [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. In this context, gradient explosion/decay is identified as one of the key reasons that prevent RNNs from being trained efficiently and effectively, where the gradient magnitude is either too small or too large, leading to severe training instability [11]. This issue has been attributed to:

P1. The number of time steps, M, is large where long-term dependencies exist among the data;

P2. The state transmission function, h, involves multiple hidden states such as in deep RNNs;

P3. The data samples, $\mathcal{X}$, are very noisy or the true signal is weak.

Proposed Method. Different from prior works that propose methods based on novel designs [12] or architectures [13] as a means to mitigate gradient decay or explosion, we propose to directly modify the well-known back-propagation algorithm.

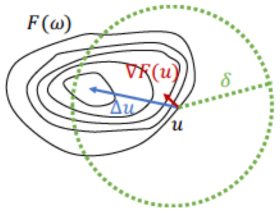

Fig. 1: Proposed method.

At a high-level, we propose to estimate the *stable* (approximate) gradients in RNNs. In Fig. 1, u denotes the current realization for function $F(\omega)$ whose gradient is $\nabla F(u)$. Δu denotes the desired output vector that points towards the local minimum from u, and $\delta \geq 0$ denotes the radius of the search region in the parameter space centered at u (denoted by the dotted circle). Obviously, $\nabla F(u)$ and Δu could be quite different, and our goal is to learn Δu, by looking around in a sufficiently small neighborhood.

Trust-Region *vs.* Projected Gradient *vs.* Frank-Wolfe. All of the three methods are potentially applicable for our learning purpose. Trust-region optimization [14, 15, 16] usually utilizes quadratic approximation to locate a local minimum given the current solution. In deep learning, however, computing the Hessian matrix in the high dimensional space is very challenging. If simplified with the identity matrix, the corresponding closed-form solution is equivalent to the ℓ_2-normalized gradient. Projected gradient [17] for training RNNs may lead to slow convergence due to the vanishing/exploding gradients, because the inner loop for locating the local minimum relies on the RNN gradients as well. In contrast, we propose to utilize the Frank-Wolfe (FW) algorithm [18], one of the simplest and earliest iterative algorithms for constrained optimization due to its projection-free property and large-scale applicability. FW maintains a feasible solution satisfying the constraints, without the need to explicitly incorporate the constraints. Nevertheless, we are generally agnostic to the approach used, and our focus is on RNN training methods that are based on locally optimized trajectory. We collect a few salient aspects of our method and FW in particular to further build intuition.

Normalized Gradient *vs.* Frank-Wolfe. Since the magnitude of vanishing or exploding gradients limits RNN trainability, one plausible approach is to normalize gradients. While normalization has been been explored in practice for training

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

deep models [19], results in [20] demonstrate that the iteration complexity of ℓ_2-normalized stochastic gradient descent (SGD) is $O(1/\epsilon^4)$ in order to achieve ϵ-stationary solutions. In contrast, although normalization is inherent in the Frank-Wolfe algorithm as well *w.r.t.* $\ell_p (p \geq 1)$-norm constraints, such normalization guarantees the solution per iteration to be inside the local region towards a local minimum. Additionally, we show that our proposed Frank-Wolfe algorithm converges faster.

Stable Direction. Gradient is singularly local in information, and as such, it does not fully capture the landscape, say, within a small ball around the point. Obviously, if the gradients were small within the entire ball, this would correspond to essentially being on a flat surface. In other cases, we are often in a situation, where the gradient vectors change rapidly, and one can make progress "if it was possible to look around in suitable neighborhood." A measure for changing landscape within a closed convex set, $\mathcal{C}$, for convex smooth functions is the curvature constant M [21]. For a smooth convex function, $g(\cdot)$, its curvature is defined as the maximum Bregman Divergence, namely,

$$M = \max_{\substack{x, y, s \in \mathcal{C} \\ y = (1-\gamma)x + \gamma s}} \frac{2}{\gamma^2}(g(y) - g(x) - \langle \nabla g(x), y - x \rangle). \tag{2}$$

Note that the curvature M captures the notion of maximum error over all linear approximation errors within the set $\mathcal{C}$, and as such it can be significantly different from the gradient at any specific location. The notion of curvature is intimately tied to the Frank-Wolfe algorithm. A remarkable fact is that the Frank-Wolfe algorithm, is *affine invariant*, and with step size $\gamma_k = 2/(k+1), \forall k \geq 1$, asserts that, at iteration k, we can guarantee $g(x^k) - g(x^*) \leq \frac{M}{k+2}$, where $g(x^*)$ is the minimum in the set $\mathcal{C}$. Now, in our context of RNNs, the function $F(\omega)$ is not globally convex, and this convergence rate is no longer true globally. Nevertheless, if one were in a position that satisfies local convexity within a sufficiently small ball of radius δ, it is then possible to assert this fact. While, even this cannot be guaranteed, the situation only demands that along the path trajectory of the updates, we are in a locally convex region. Additionally, one can often realize local convexity by adding a suitably small quadratic regularizer, which is adapted to the current location in the parameter space. In Sec. IV, under suitable technical conditions, which fall short of global convexity, we show that our proposed algorithm converges at a sublinear rate of $O(1/\epsilon)$ to achieve ϵ approximation error.

The key insight from Eq. 2 for training RNNs is that *for convergence guarantees, local curvature constant, rather than the noisy gradient at the current location is of relevance.* This provides us a means to mitigate vanishing gradients, through leveraging Frank-Wolfe algorithm.

Contributions. Our key contributions in this paper are:

- We propose a novel yet simple RNN optimizer based on the Frank-Wolfe method;
- We theoretically analyze the convergence of our algorithm and its benefits in RNN training;
- We empirically conduct comprehensive experiments to demonstrate the effectiveness and efficiency of our algorithm in various settings that cover all the scenarios of P1, P2, P3.

II. Related Work

Below we only summarize the related works in the literature of RNNs.

Optimization in RNNs. Truncated backpropagation through time (TBPTT) [22] is a widely used technique to avoid vanishing/exploding gradients in RNNs by controlling the maximum number of time steps in gradient calculation, although it has been demonstrated to be not robust to long-term dependencies [23]. Gradient clipping [11, 24] is another common technique to prevent the exploding (not vanishing) gradients[1] by, for instance, rescaling gradients when their norms are over a predefined threshold. It has been proved in [20] that gradient clipping can accelerate the training of deep neural networks. The scheme of initializing recurrent weight matrices to be identity or orthogonal has been widely studied such as [25, 26, 27, 28, 29, 30, 31]. Weight matrix reparametrization has been explored as well [32]. Some other optimization approaches for training RNNs are proposed as well such as real-time recurrent learning (RTRL) [33]. To the best of our knowledge, trust-region (although it has been explored in other applications of deep learning such as reinforcement learning [34]) or projected gradient methods have not been studied widely as an RNN optimizer, which we consider as one of our future works.

Novel Network Architecture Development for RNNs. Recently there are significant amount of works on developing variants of RNNs such as, just to name a few, long short-term memory (LSTM) [13], gated recurrent unit (GRU) [35, 36], Fourier recurrent unit [37], UGRNN [38], FastGRNN [39], unitary RNNs [26, 27, 40, 41, 42], deep RNNs [43, 44, 45], linear RNNs [46, 47, 48], residual/skip RNNs [39, 49, 50, 51, 52], ordinary differential equation (ODE) based RNNs [12, 39, 53, 54, 55, 56, 57]. For instance, FastGRNN [39] feed-forwards state vectors to induce skip or residual connections, to serve as a middle ground between feed-forward and recurrent models, and to mitigate gradient decay. Incremental RNNs (iRNNs) [57] are developed based on ODE with theoretical guarantee of identity gradients in the intermediate steps in chain rule for gradient calculation. Independently RNN (IndRNN) [24] is a network structure where neurons in the same layer are independent of each other and multiple IndRNNs can be stacked to construct a deep network.

III. Frank-Wolfe RNN Optimizer

Recall that in our proposed method, given the current point ω in the parameter space, we attempt to solve a local minima in small ball around ω. To this end, we consider the following

[1]Therefore, in practice vanishing gradients are more often for performance degradation of RNNs.

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Frank-Wolfe RNN Optimizer

```
Input  : objective $f$, norm $p$, local radius $\delta_t, \forall t$, max numbers
         of iterations $K, T$
Output : RNN weights $\omega$
Randomly initialize $\omega_0$;
for $t = 1, \cdots, T$ do
    $\Delta\omega_{t,0} \leftarrow \mathbf{0}$;
    for $k = 1, \cdots, K$ do
        $s_{t,k} \leftarrow \arg\min_{s \in \mathcal{C}(p,\delta_t)} \langle s, \nabla_{\Delta\omega} F(\omega_{t-1} + \Delta\omega_{t,k-1}) \rangle$;
        $\Delta\omega_{t,k} \leftarrow (1 - \frac{1}{k})\Delta\omega_{t,k-1} + \frac{1}{k} s_{t,k}$;
    end
    $\omega_t \leftarrow \omega_{t-1} + \eta \Delta\omega_{t,K}$;
end
return $\omega_T$;
```

general constrained optimization problem, which is also a central aspect of the Frank-Wolfe method:

$$\min_{\|\Delta\omega\|_p \le \delta} F(\omega + \Delta\omega), \tag{3}$$

where $p > 1$ is any ℓ_p norm. For our proposed method (see Alg. 1), this constrained optimization is carried out in an inner loop, by means of the FW method. Once, a satisfactory point is reached (which usually is quite fast), we then update the parameters in an outer loop.

Frank-Wolfe Algorithm. We apply the stochastic Frank-Wolfe algorithm to solve Eq. 3, and list our novel and simple Frank-Wolfe based RNN optimizer in Alg. 1. In contrast to GD, the iterations in the Frank-Wolfe algorithm are as follows, in general:

$$\begin{aligned} s^{(k-1)} &= \arg\min_{s \in \mathcal{C}} \left\langle s, \nabla f(x^{(k-1)}) \right\rangle, \\ x^{(k)} &= (1 - \gamma_k) x^{(k-1)} + \gamma_k s^{(k-1)}, \end{aligned} \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors, and $\gamma_k \in [0, 1]$. For $\ell_p (p \ge 1)$-norm constraints, *i.e.,* $\mathcal{C}(p, \delta) = \{s \mid \|s\|_p \le \delta\}$, there exist close-form solutions for $\arg\min$ in Eq. 4, as

$$\begin{gathered} s^{(k-1)} = -\alpha \cdot \mathrm{sgn}(\nabla f(x^{(k-1)})) \cdot |\nabla f(x^{(k-1)})|^{\frac{p}{q}}, \\ \text{s.t. } 1/p + 1/q = 1, \end{gathered} \tag{5}$$

where sgn and $|\cdot|$ denote entry-wise sign and absolute operators, respectively, and $\alpha \ge 0$ is a scalar satisfying $\|s^{(k-1)}\|_q = \delta$. In particular,

- $p = 2 \Rightarrow s^{(k-1)} = -\delta \cdot \frac{\nabla f(x^{(k-1)})}{\|\nabla f(x^{(k-1)})\|_2}$, *i.e.,* δ-scaled ℓ_2 normalized gradient;
- $p \to \infty \Rightarrow s^{(k-1)} = -\delta \cdot \mathrm{sgn}(\nabla f(x^{(k-1)}))$, *i.e.,* δ-scaled sign gradient.

In Alg. 1 we set $\gamma_k = \frac{1}{k}$ to ensure that $\Delta\omega$ always lies inside the local region with convergence guarantee [58]. Similarly we decrease δ_t with the learning rate. Note that Adam [59] can be used to update ω. Besides, Alg. 1 can be applied to train other deep models as well. Similar two-loop algorithmic structures have been explored in the literature such as [60] for training convolutional neural networks (CNNs).

In this paper we focus on the analysis and experiments with $p = 2$, although our analysis generally holds for $p \ge 1$. In practice, we utilize $p \to \infty$ to train IndRNN [24] and achieve 98.37% on Pixel-MNIST with $K = 30$ over 400 epochs, similar to the numbers in Table II.

IV. Analysis

Zhu *et al.* in [61] proved that for training RNNs, when the number of neurons is sufficiently large, meaning *polynomial* in the training data size and in network depth, then SGD is capable of minimizing the regression loss in the linear convergence rate. In contrast, we discuss the convergence of Alg. 1 for an arbitrary RNN. We start our analysis from a special case with ℓ_2-normalized gradients as follows:

Theorem 1 (ℓ_2-Normalized Gradients). *Suppose that the following assumptions hold globally:*

- *F is lower-bounded, twice differentiable, and (L_0, L_1)-smoothness,* i.e., $\|\nabla^2 F(\omega)\|_2 \le L_0 + L_1 \|\nabla F(\omega)\|_2, \forall\omega, \exists L_1, L_2 > 0$*;*
- *There exist a $\tau > 0$ such that $\|\nabla f(\mathcal{B}, \omega) - \nabla F(\omega)\|_2 \le \tau, \forall\mathcal{B}, \forall\omega$ holds.*

Then for $K = 1$ in Alg. 1, there exists $\delta > 0$ so that in order to achieve ϵ-stationary points the iteration complexity for Alg. 1 is upper bounded by $O(1/\epsilon^4)$, at least.

To prove this theorem, please refer to Thm. 7 in [20]. Below we extend our analysis to general cases.

Definition 1 (Star-convexity [62]). *Let ω^* be a global minimizer of a smooth function F. Then, F is said to be star-convex at a point ω provided that $F(\omega) - F(\omega^*) + \langle \omega^* - \omega, \nabla F(\omega) \rangle \le 0, \forall\omega$.*

Theorem 2 (Convergence). *Let $\{\omega_t\}_{t \in [T]}$ be the sequence of the weight updates from Alg. 1. Suppose*

A1. F in Eq. 1 is locally convex *within each radius $\delta_t, \forall t \in [T]$* w.r.t. *$\ell_2$ norm centered at ω_{t-1};*

A2. F in Eq. 1 is also star-convex, *given a global minimizer ω^*,* i.e., *F is lower bounded by $F(\omega^*)$;*

A3. F in Eq. 1 is differentiable and its gradient is Lipschitz continuous with constant $L > 0$, i.e., $\|\nabla F(\omega_1) - \nabla F(\omega_2)\|_2 \le L \|\omega_1 - \omega_2\|_2, \forall\omega_1, \omega_2$*;*

A4. $\omega_t, \forall t$ is upper bounded w.r.t. *ω^*,* i.e., $\|\omega_t - \omega^*\|_2 \le \alpha < +\infty, \forall\omega, \exists\alpha$*;*

A5. It holds that $\beta \le \|\nabla F(\omega_{t-1}) - \Delta\omega_{t,K}\|_2 \le (1 - L\eta)^{\frac{1}{2}} \|\Delta\omega_{t,K}\|_2, \forall t, \exists\beta > 0, \exists\eta \le \frac{1}{L}$.

Then we have that the output of Alg. 1, ω_T, satisfies

$$F(\omega_T) - F(\omega^*) \le \frac{\|\omega_0 - \omega^*\|_2^2 + 2\eta\rho}{2\eta T}, \tag{6}$$

where $\rho = \left(\frac{\alpha}{\beta} - \frac{\eta}{2}\right)(1 - L\eta) \sum_t \|\Delta\omega_{t,K}\|_2^2 \in \mathbb{R}$, i.e., *a real number, and ω_0 denotes the initialization of the network weights. In particular, ω_T will converge to ω^** asymptotically *if*

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

$\lim_{T\to+\infty}\sum_{t=1}^{T}\delta_t^2 < +\infty$ *holds. Further, if* $0 \le \|\omega_0-\omega^*\|_2^2 + 2\eta\rho < +\infty$ *holds, then* ω_T *will converge to* ω^* sublinearly.

Proof. Based on Assmp. A1, A3 and A5, we have

$$F(\omega_t) \le F(\omega_{t-1}) + \langle \nabla F(\omega_{t-1}), \omega_t - \omega_{t-1}\rangle + \frac{L}{2}\|\omega_t - \omega_{t-1}\|_2^2$$
$$= F(\omega_{t-1}) - \eta\langle \nabla F(\omega_{t-1}), \Delta\omega_{t,K}\rangle + \frac{L\eta^2}{2}\|\Delta\omega_{t,K}\|_2^2$$
$$= F(\omega_{t-1}) + \frac{\eta}{2}\|\nabla F(\omega_{t-1}) - \Delta\omega_{t,K}\|_2^2$$
$$- \frac{\eta}{2}\|\nabla F(\omega_{t-1})\|_2^2 + \frac{L\eta^2-\eta}{2}\|\Delta\omega_{t,K}\|_2^2 \le F(\omega_{t-1}). \tag{7}$$

Further, based on Assmp. A2, A4 and A5, we have

$$F(\omega_t) - F(\omega^*) \le \langle \nabla F(\omega_{t-1}), \omega_{t-1} - \omega^*\rangle - \frac{\eta}{2}\|\nabla F(\omega_{t-1})\|_2^2$$
$$+ \frac{\eta}{2}\|\nabla F(\omega_{t-1}) - \Delta\omega_{t,K}\|_2^2 + \frac{L\eta^2-\eta}{2}\|\Delta\omega_{t,K}\|_2^2$$
$$= \frac{1}{2\eta}\left(\|\omega_{t-1}-\omega^*\|_2^2 - \|\omega_t - \omega^*\|_2^2\right)$$
$$+ \langle \nabla F(\omega_{t-1}) - \Delta\omega_{t,K}, \omega_t - \omega^*\rangle + \frac{L\eta^2-\eta}{2}\|\Delta\omega_{t,K}\|_2^2$$
$$\le \frac{1}{2\eta}\left(\|\omega_{t-1}-\omega^*\|_2^2 - \|\omega_t - \omega^*\|_2^2\right)$$
$$+ \left(\frac{\alpha}{\beta} - \frac{\eta}{2}\right)(1-L\eta)\|\Delta\omega_{t,K}\|_2^2. \tag{8}$$

Now based on Eq. 7 and Eq. 8, we can complete our proof by

$$F(\omega_T) - F(\omega^*) \le \frac{1}{T}\sum_t F(\omega_t) - F(\omega^*)$$
$$\le \frac{\|\omega_0-\omega^*\|_2^2}{2\eta T} + \frac{1}{T}\left(\frac{\alpha}{\beta} - \frac{\eta}{2}\right)(1-L\eta)\sum_t \|\Delta\omega_{t,K}\|_2^2.$$

□

Equivalently, Thm. 2 states that our algorithm needs $O(\frac{1}{\epsilon})$ updates of ω, independent on the inner loops K in Alg. 1, in order to achieve an ϵ-stationary solution. Note that the constant ρ is highly correlated with the number of inner loops, K, in the algorithm. Therefore, empirically it is challenging to tell which choice of K will be the best. From our experiments, we found that often small K's can work better than SGD for training RNNs.

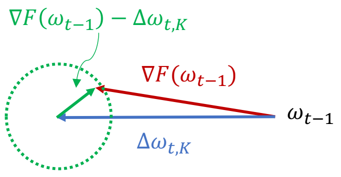

Fig. 2: Geometric interpretation.

Geometric Interpretation of Assmp. A5. In Fig. 2, we illustrate the geometric relationship between $\nabla F(\omega_{t-1}), \Delta\omega_{t,K}$ and $\nabla F(\omega_{t-1}) - \Delta\omega_{t,K}$ given the current solution ω_{t-1} and $\beta \to 0$. Clearly, any point within the green dotted circle (with radius $(1-L\eta)^{\frac{1}{2}}\|\Delta\omega_{t,K}\|_2$) will be a candidate for $\nabla F(\omega_{t-1})$ so that A5 holds. Therefore, the angle between any pair of $\nabla F(\omega_{t-1})$ and $\Delta\omega_{t,K}$ should be no more than $\pm 45^o$. In other words, $\nabla F(\omega_{t-1})$ and $\Delta\omega_{t,K}$ should have very similar directions. We verified this on the HAR-2 dataset (see Sec. V) by computing such angles to see the distribution using 200 epochs. Statistically these angles are within $\pm 45^o$ with mean -4.19^o and std 0.51^o.

Inexact Update in Frank-Wolfe. In the stochastic setting with limited K and large-scale data, however, $\nabla F(\omega_{t-1}), \Delta\omega_{t,K}$ are infeasible to compute exactly. To address this problem, we borrow the concept of approximation quality in the inexact Frank-Wolfe method in [63].

Definition 2 ([63])**.** *In the Frank-Wolfe update,* $s_{t,k}(\|s_{t,k}\|_2 \le \delta_t)$ *is used so that it holds that*

$$\langle s_{t,k}, \nabla F(\omega_{t-1} + \Delta\omega_{t,k-1})\rangle \le$$
$$\min_{\|s\|_2\le\delta_t}\langle s, \nabla F(\omega_{t-1} + \Delta\omega_{t,k-1})\rangle + \frac{\lambda M_F}{k+1}, \ \forall k \ge 1, \tag{9}$$

where $\lambda \ge 0$ *denotes an arbitrary accuracy parameter that controls the upper bound of the convergence rate linearly from factor* 1 *to* $(1+\lambda)$*, and* M_F *denotes the curvature constant of* F*.*

Note that Eq. 9 will still hold by setting $\lambda = \max_k\left\{\frac{2\delta_t(k+1)}{M_F}\|\nabla F(\omega_{t-1} + \Delta\omega_{t,k-1})\|_2\right\} < +\infty$. Clearly, as long as $\|\nabla F(\omega_{t-1} + \Delta\omega_{t,k-1})\|_2 = O(1/k)$ is met, the convergence of Alg. 1 can be always guaranteed.

Stochastic Frank-Wolfe in Alg. 1. In our implementation, we utilize the stochastic Frank-Wolfe method [58] instead for computational efficiency. Essentially stochastic FW can be considered as a realization of inexact update in FW, and thus all the analysis above holds for this case as well.

TABLE I: Dataset Statistics

Dataset	#Train	#Test	#TimeStep	#Feature
HAR-2	7,352	2,947	128	9
Noisy-HAR-2	7,352	2,947	128	9
Pixel-MNIST	60,000	10,000	784	1
Permute-MNIST	60,000	10,000	784	1

V. Experiments

Datasets. We test our RNN optimizer on the following benchmark datasets with all the statistics listed in Table I:

- *Adding task:* We strictly follow the adding task[2] defined by [13, 26] to generate the dataset. There are two sequences with length $T = 100$. The first sequence is sampled uniformly at random $\mathcal{U}[0, 1]$. The second sequence is filled with 0 except for two entries of 1. The two entries of 1 are located uniformly at random position i_1, i_2 in the first half and second half of the sequence. The prediction value is the sum of the first sequence between $[i_1, i_2]$.
- *Pixel-MNIST & Permute-MNIST:* Pixel-MNIST refers to pixel-by-pixel sequences of images in MNIST where each 28×28 image is flattened into a 784 time sequence vector, while a random permutation to the Pixel-MNIST is applied to generate a harder time sequence dataset as Permute-MNIST. All datasets are normalized as zero mean and unit variance during training and prediction.

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

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

- *HAR-2 [39]:* HAR-2 was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone, and all samples are normalized with zero mean and unit variance.
- *Noisy-HAR-2:* This dataset is generated by adding Gaussian noise to HAR-2 with a mean of zero and a variance of two to HAR-2 to evaluate the robustness of our algorithm.

Baseline Algorithms and Implementation. We compare our Frank-Wolfe based RNN optimizer with SGD (gradient clipping involved if necessary) and TBPTT comprehensively, using (1) a vanilla RNN with one-layer transition function consisting of a linear function followed by a $\tanh$ activation (same as the literature. See [64]) and (2) IndRNN with six layers[3]. Mean-square-error (MSE) and cross-entropy losses are applied for binary and multi-class classification tasks, respectively. Adam [59] is used as the optimizer for all the methods. We implement our experiments using PyTorch, and run all the experiments on an Nvidia GeForce RTX 2080 Ti GPU with CUDA 10.2 and cuDNN 7.6.5 on a machine with Intel Xeon 2.20 GHz CPU with 48 cores. The code of our optimizer can be found here [4].

Hyperparameters. The hyperparameters of the vanilla RNN and IndRNN are the same as the literature [24, 64, 65], if applicable, as from our experiments they seem to be the best settings. For our optimizer we perform a grid search over several learning rates {2e-5, 2e-4, 6e-4, 1e-3}, batch size {32, 64, 128, 256, 512} and learning rate decay factor {0.1, 0.5, 0.9} on validation data (from a small portion of training data) to ensure that a good parameter is used in our algorithm. We use hidden dimension 128 for MNIST datasets and the Adding task. When training HAR-2 related tasks, we use hidden dimension 80. We report the best test accuracy.

A. Results

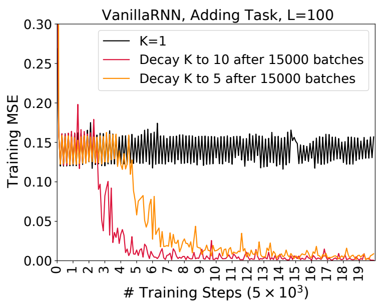

Fig. 3: Training loss of Adding Task

Adding Task. The Adding task is designed to evaluate the capability of RNNs to capture long-term dependency among the data [13, 24, 25, 26, 26, 40]. Fig. 3 illustrates the loss change of our algorithm compared with SGD on this task when the time sequence is long. It is clear that our algorithm converges after a reasonable number of iterations while SGD lost the learning ability in this task. We hypothesize that at the beginning all the algorithms search for a good direction within a certain region. Given sufficient updates later, our algorithm starts to move towards informative directions, leading to significantly fast convergence.

Vanilla RNN on Pixel-MNIST & Permute-MNIST. We further apply our algorithm to Pixel-MNIST and Permute-MNIST and compare it with Adam and TBPTT. The forward and backward steps of TBPTT are set to 196, which means each 784 time sequence is partitioned into 4 segments. For our algorithm, we perform the inner iteration K=1 and 5. Since our algorithm can be easily combined with TBPTT, we also include the combination experiments on the two datasets.

[3]https://github.com/Sunnydreamrain/IndRNN_pytorch
[4]https://github.com/YunYunY/FW_RNN_optimizer

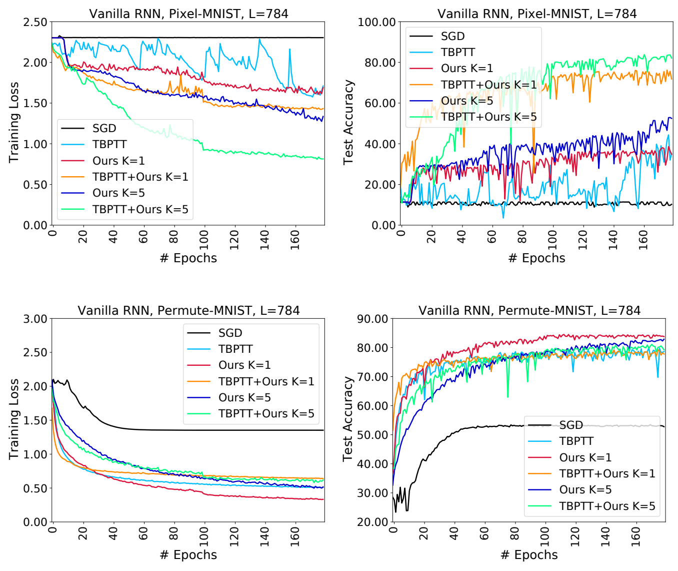

Fig. 4: Training loss and test accuracy on Pixel-MNIST and Permute-MNIST

Fig. 4 shows the change of training cross-entropy and test accuracy of RNN with the epoch for Pixel-MNIST and Permute-MNIST. Without extra optimization techniques, SGD shows no convergence or very slow convergence. We observe that TBPTT does help the convergence for the Permute-MNIST, however, the performance of TBPTT is only slightly better than the baseline SGD in the Pixel-MNIST case with sporadically increases and decreases of loss. As a contrast, our algorithm shows a faster convergence rate and a much more stable performance on both datasets. When TBPTT is combined with our algorithm, the model achieves faster convergence and higher test accuracy than the baseline for Pixel-MNIST. As for Permute-MNIST, the combination method eventually reaches higher test accuracy with more training epochs. It is worth mentioning that when the inner iteration K increases in our algorithm, the total number of gradient updates needed for convergence decreases.

IndRNN on Pixel-MNIST & Permute-MNIST. This experiment addresses the application of our algorithm on training deep models. Since IndRNN can be stacked to construct a deep network [24], we apply a six-layer IndRNN structure with the two benchmark datasets Pixel-MNIST and Permute-MNIST. The model has proved the state-of-the-art performance on the two datasets with Adam optimizer [24], thus we also use Adam as the baseline algorithm to compare with ours. The model structure we use follows exactly [24].The inner iteration K is tested with values $\{1, 5, 10, 30\}$.

The training loss and test accuracy results of the two datasets trained with IndRNN are shown in Fig. 5. IndRNN applied batch normalization (BN) to accelerate Pixel-MNIST training.

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

Due to the truncated training process of TBPTT, the relevant statistics over the mini-batch changes over time. It is not suitable to apply BN to TBPTT. Thus in Fig. 5 we only compare our algorithm with the baseline IndRNN. Our algorithm has the same performance as Adam optimizer. In terms of accuracy, our algorithm achieves comparable test accuracy after 400 epochs using about half training time when K=5.

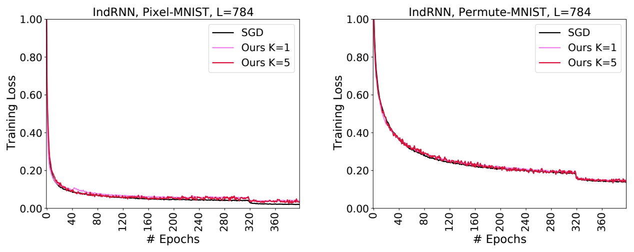

Fig. 5: Training loss on Pixel-MNIST and Permute-MNIST with indRNN

TABLE II: Test accuracy (%) (training hours) of IndRNN

Dataset	Acc. (Time)		
	Baseline	Ours K=1	Ours K=5
Pixel-MNIST	98.88 (4.84)	98.73 (3.46)	98.82 (2.55)
Permute-MNIST	93.00 (4.92)	92.87 (3.68)	92.59 (2.41)

RNNs on HAR-2 & Noisy HAR-2. This experiment focuses on noise-free and noisy sequences. When the data samples are very noisy, RNNs usually exhibit unstable performance. To verify the robustness of our algorithm on noisy input, we compare the performance of the proposed optimizer with Adam and TBPTT on plain RNN with HAR-2 and Noisy-HAR-2 as input. The task is binary classification after observing a long sequence. HAR-2 has a 128 time sequence. Adding Gaussian noise with a mean of zero and a variance of two makes the task harder. We set forward and backward steps in TBPTT as 16. Thus the sequence is split into 16 segments. Same as Experiment 1, we also include the combination of TBPTT and our algorithm in the experiment. We also demonstrate the compatibility of our algorithm with LSTM and BN using the two datasets. We estimate the MSE of each experiment.

Fig. 6 and Table III show that our algorithm is not only robust to long-term dependency tasks but also robust to noisy time sequences. No matter the input data is noise-free or noisy, the training losses of SGD oscillate up and down. The TBPTT reaches a desirable MSE in the noise-free case because the forward and backward steps are set up relatively short. TBPTT is capable of avoiding the vanishing gradient issue in this setting. The literature shows RNN with SGD usually reaches a test accuracy 91.31% with the learning that takes at least 300 epochs on HAR-2 dataset [39]. With the same initial learning rate setting, our algorithm outperforms the baseline within less training epochs. It is shown in Table III that gradient clipping does improve the model performance on noise-free data, however, it loses its advantage when applied to Noisy-HAR-2 dataset. When combining our algorithm with TBPTT,

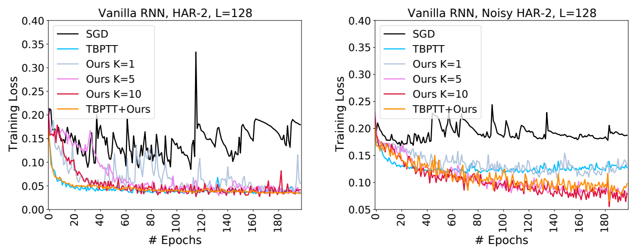

Fig. 6: Training loss of RNN on HAR-2 and Noisy-HAR-2

TABLE III: Test accuracy (%) and training time (hr) of RNN

Method	HAR-2	Time	Noisy-HAR-2	Time
SGD	87.66	0.17	74.38	0.17
SGD+Clipping	93.36	0.13	74.38	0.13
TBPTT	93.62	0.38	86.20	0.56
LSTM+Adam	94.40	0.14	92.12	0.17
Ours K=1	93.52	0.15	86.04	0.14
Ours K=5	94.11	0.14	89.36	0.14
Ours K=10	93.65	0.37	89.52	0.35
Ours+BN	94.37	0.36	89.38	0.41
TBPTT+Ours	94.01	0.35	89.28	0.84
LSTM+Ours	94.95	0.19	**92.41**	0.42
IndRNN	95.73	0.46	91.20	0.45
IndRNN+Ours	**96.55**	0.13	92.15	0.17

we reach the test accuracy 94.01%. The baseline accuracy of SGD is 15.14% less than our algorithm on the Noisy-HAR-2 task. We also verified that our optimizer works well with LSTM and BN as listed in Table III. When our algorithm is combined with IndRNN, it overpasses the original IndRNN with less running time.

VI. Conclusions and Future Directions

In this paper, we propose a novel and simple RNN optimizer based on the Frank-Wolfe method. We provide a theoretical proof of the convergence of our algorithm. The empirical experiments on several benchmark datasets demonstrate that the proposed RNN optimizer is an effective solver for the training stability of RNNs. Our algorithm outperforms SGD in all experiments and boosts the TBPTT performances. It also reaches a comparable test accuracy with the baseline algorithm in deep RNNs while requiring fewer gradient updates and less training time. The algorithm shows robustness in the noisy data classification experiment with an improvement of 15.14%. This work motivates the RNN training on a distributed system. In future work, we will investigate the application of our algorithm in a distributed setting which can reach significant speed-ups at no or nearly no loss of accuracy.

Acknowledgment

Yun Yue, Ming Li, and Ziming Zhang were supported in part by NSF CCF-2006738.

References

[1] M. A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 815–826, 1983.

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

[2] A. Guez, V. Protopopsecu, and J. Barhen, “On the stability, storage capacity, and design of nonlinear continuous neural networks,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 80–87, 1988.

[3] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[4] ——, “Neurons with graded response have collective computational properties like those of two-state neurons,” *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[5] D. G. Kelly, “Stability in contractive nonlinear neural networks,” *IEEE Transactions on Biomedical Engineering*, vol. 37, no. 3, pp. 231–242, 1990.

[6] K. Matsuoka, “Stability conditions for nonlinear continuous neural networks with asymmetric connection weights,” *Neural networks*, vol. 5, no. 3, pp. 495–500, 1992.

[7] S. Hui and S. H. Zak, “Dynamical analysis of the brain-state-in-a-box (bsb) neural models,” *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 86–94, 1992.

[8] A. N. Michel, J. Si, and G. Yen, “Analysis and synthesis of a class of discrete-time neural networks described on hypercubes,” in *IEEE International Symposium on Circuits and Systems*. IEEE, 1990, pp. 700–703.

[9] J. Collins, J. Sohl-Dickstein, and D. Sussillo, “Capacity and trainability in recurrent neural networks,” *arXiv preprint arXiv:1611.09913*, 2016.

[10] J. Miller and M. Hardt, “Stable recurrent models,” *arXiv preprint arXiv:1805.10369*, 2018.

[11] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International conference on machine learning*, 2013, pp. 1310–1318.

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

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

[14] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, “A trust region algorithm for nonlinearly constrained optimization,” *SIAM Journal on Numerical Analysis*, vol. 24, no. 5, pp. 1152–1170, 1987.

[15] C. Fortin and H. Wolkowicz, “The trust region subproblem and semidefinite programming,” *Optimization methods and software*, vol. 19, no. 1, pp. 41–67, 2004.

[16] N. M. Alexandrov, J. Dennis, R. M. Lewis, and V. Torczon, “A trust-region framework for managing the use of approximation models in optimization,” *Structural optimization*, vol. 15, no. 1, pp. 16–23, 1998.

[17] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[18] M. Frank and P. Wolfe, “An algorithm for quadratic programming,” *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[19] Z. Zhang, Y. Wu, and G. Wang, “Bpgrad: Towards global optimality in deep learning via branch and pruning,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[20] J. Zhang, T. He, S. Sra, and A. Jadbabaie, “Why gradient clipping accelerates training: A theoretical justification for adaptivity,” in *International Conference on Learning Representations*, 2020.

[21] M. Jaggi and M. Sulovskỳ, “A simple algorithm for nuclear norm regularized problems,” 2010.

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

[23] C. Tallec and Y. Ollivier, “Unbiasing truncated backpropagation through time,” *arXiv preprint arXiv:1705.08209*, 2017.

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

[25] Q. V. Le, N. Jaitly, and G. E. Hinton, “A simple way to initialize recurrent networks of rectified linear units,” *arXiv preprint arXiv:1504.00941*, 2015.

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

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

[28] C. Jose, M. Cisse, and F. Fleuret, “Kronecker recurrent units,” *arXiv preprint arXiv:1705.10142*, 2017.

[29] Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey, “Efficient orthogonal parametrisation of recurrent neural networks using householder reflections,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2401–2409.

[30] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, “Full-capacity unitary recurrent neural networks,” in *Advances in neural information processing systems*, 2016, pp. 4880–4888.

[31] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal, “On orthogonality and learning recurrent networks with long term dependencies,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3570–3578.

[32] J. Zhang, Q. Lei, and I. S. Dhillon, “Stabilizing gradients for deep neural networks via efficient svd parameterization,” *arXiv preprint arXiv:1803.09327*, 2018.

[33] R. J. Williams and D. Zipser, “A learning algorithm for continually running fully recurrent neural networks,” *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[34] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, 2015, pp. 1889–1897.

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

[36] S. Kanai, Y. Fujiwara, and S. Iwamura, “Preventing gradient explosions in gated recurrent units,” in *Advances in neural information processing systems*, 2017, pp. 435–444.

[37] J. Zhang, Y. Lin, Z. Song, and I. Dhillon, “Learning long term dependencies via fourier recurrent units,” in *International Conference on Machine Learning*, 2018, pp. 5815–5823.

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

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

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

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

[42] J. Pennington, S. Schoenholz, and S. Ganguli, “Resurrecting the sigmoid in deep learning through dynamical isometry: theory

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.

and practice," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., 2017, pp. 4785–4795.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[57] A. Kag, Z. Zhang, and V. Saligrama, "Rnns evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients?" *arXiv preprint arXiv:1908.08574*, 2019.

[58] S. J. Reddi, S. Sra, B. Póczos, and A. Smola, "Stochastic frank-wolfe methods for nonconvex optimization," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 1244–1251.

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

[60] Z. Zhang, W. Xu, and A. Sullivan, "Time-delay momentum: A regularization perspective on the convergence and generalization of stochastic momentum for deep learning," *arXiv preprint arXiv:1903.00760*, 2019.

[61] Z. Allen-Zhu, Y. Li, and Z. Song, "On the convergence rate of training recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 6673–6685.

[62] Y. Zhou, J. Yang, H. Zhang, Y. Liang, and V. Tarokh, "Sgd converges to global minimum in deep learning via star-convex path," in *International Conference on Learning Representations*, 2018.

[63] M. Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization." in *Proceedings of the 30th international conference on machine learning*, no. CONF, 2013, pp. 427–435.

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

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

Authorized licensed use limited to: Worcester Polytechnic Institute. Downloaded on September 30,2021 at 21:04:59 UTC from IEEE Xplore. Restrictions apply.