RNN Training along Locally Optimal Trajectories via Frank-Wolfe Algorithm

Yun Yue *, Ming Li *, Venkatesh Saligrama † and Ziming Zhang*

*Worcester Polytechnic Institute, †Boston University

Background

Recurrent neural networks (RNNs) have been widely used in learning complex patterns for sequential input data. However, gradient explosion/decay is identified as one of the key reasons that prevent RNNs from being trained efficiently and effectively. The issue is mainly caused by:

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

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

P3. The data samples are very noisy or the true signal is weak.

Contributions

- 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.

Frank-Wolfe RNN Optimizer

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.

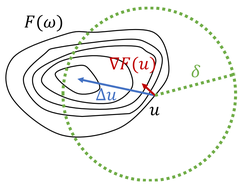


Fig. 1: Proposed method.

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 ω

Randomly initialize ω_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 ω_T;

Experiment Results

Fig. 2 illustrates the loss change of our algorithm compared with SGD on adding task when the time sequence is long. 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.

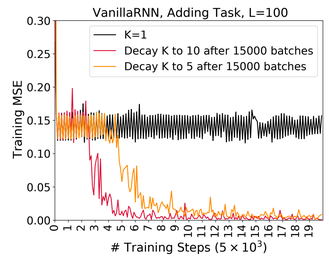
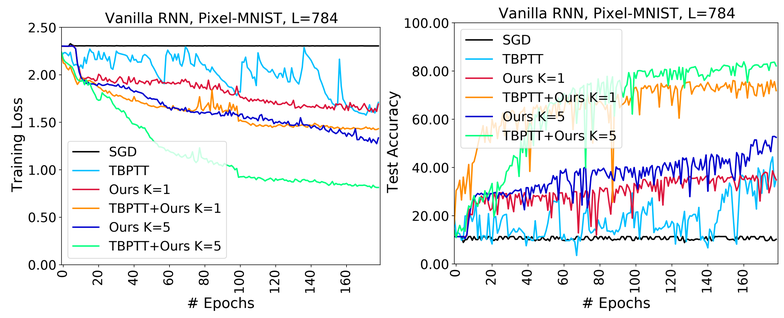


Fig. 2: Training loss of Adding Task.



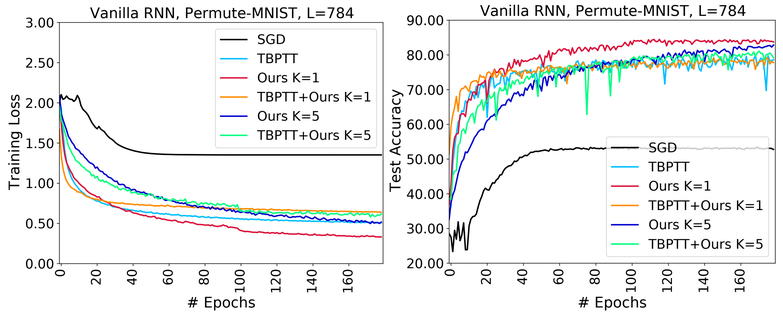


Fig. 3: Training loss and test accuracy on Pixel-MNIST and Permute-MNIST.

Fig. 3 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.

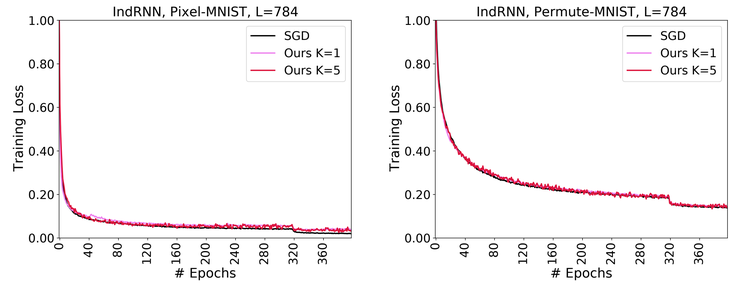


Fig. 4: Training loss on Pixel-MNIST and Permute-MNIST with indRNN.

Table 1: 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)

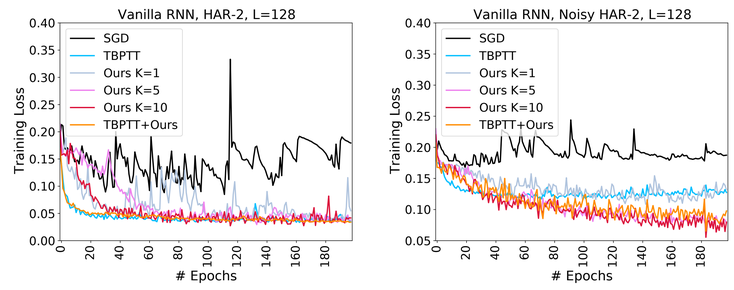


Fig. 5: Training loss of RNN on HAR-2 and Noisy-HAR-2.

Table 2: 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

Outlook

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.

Contacts: {yyue, mli12, zzhang15}@wpi.edu, srv@bu.edu