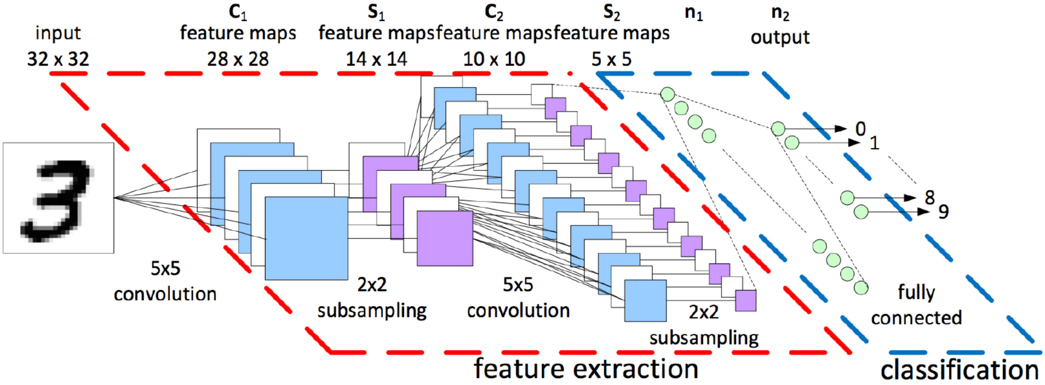
Advances of Momentum in Optimization Algorithm and Neural Architecture Design

Bao Wang
Department of Mathematics
Scientific Computing and Imaging Institute
University of Utah

Partially supported by DoE, NSF, and Univ of Utah

Deep Learning (DL)



DL = Big Data + Deep Nets + SGD + HPC

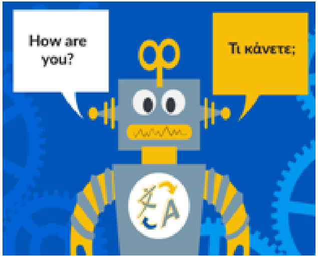
Deep Learning: Revolution in Technology

Face ID

Autonomous Cars

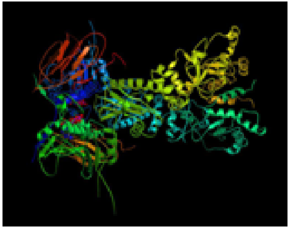
Alpha Go

Machine Translation

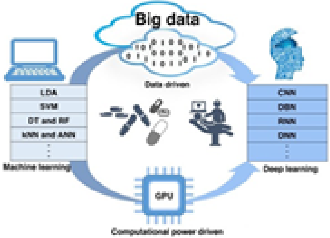


Deep Learning: Revolution in Science

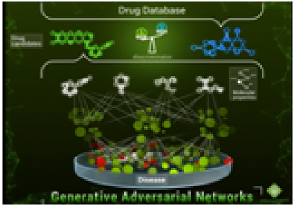
Protein Structure Prediction



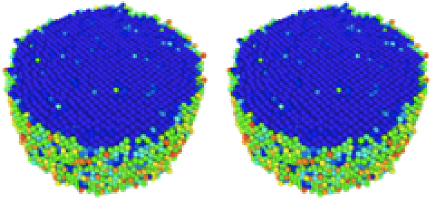
Drug Discovery



Molecular Generation



Material Design



Deep Learning is Expensive!

1. Neural architecture design is mostly art instead of science!

2. Training deep neural networks is expensive:

   2.1 No principled approach in selecting optimization algorithm!

   2.2 Slow convergence!

Deep Learning is Very Expensive

AlphaGO	**Lee Se-dol**
1202 CPUs, 176 GPUs, 100+ Scientists.	1 Human Brain, 1 Coffee.

Simple and principled approaches converge with working machine learning algorithms!

A few examples:

- **Nesterov Accelerated SGD with Restart I**
- **Integrate Momentum into Recurrent Neural Network II**

I. Scheduled Restart Momentum for Accelerated Stochastic Gradient Descent

B. Wang, T. Nguyen, T. Sun, A. Bertozzi, R. Baraniuk, and S. Osher, Scheduled Restart Momentum for Stochastic Gradient Descent, arXiv:2002.10583, 2020.
Code: https://github.com/minhtannguyen/SRSGD

Blog: http://almostconvergent.blogs.rice.edu/2020/02/21/srsgd/

Empirical Risk Minimization (ERM)

Consider training a machine learning model

$$y = g(\mathbf{x}, \mathbf{w}), \quad \mathbf{w} \in \mathbb{R}^d.$$

Empirical Risk Minimization (ERM)

$$\min_{\mathbf{w}} f(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(g(\mathbf{x}_i, \mathbf{w}), y_i),$$

where $\mathcal{L}$ is the loss between the predicted label $\hat{y}_i$ and the ground-truth label y_i.

Classification: cross-entropy loss $\mathcal{L}(\hat{y}_i, y_i) = -\sum_{j=1}^{c} y_i^j \log(p_i^j)$. where p_i^j is the predicted probability that y_i is belong to j-th class.

Regression: mean squared error $\mathcal{L}(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2$.

Challenges: $d \sim 10^{10}$, $N \sim 10^{10}$, and $f(\mathbf{w})$ is nonconvex.

Gradient Descent

Suppose $f(\mathbf{w})$ is L-smooth, i.e., $\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{v})\|_2 \leq L\|\mathbf{w} - \mathbf{v}\|_2$.

Start from $\mathbf{w}_0$, gradient descent performs the following iteration

$$\mathbf{w_k} = \mathbf{w_{k-1}} - \mathbf{s}\nabla \mathbf{f}(\mathbf{w_{k-1}}).$$

1. $f(\mathbf{w})$ is μ-strongly convex (bounded below by a quadratic function), let $s = 2/(\mu + L)$, we have

$$\|\mathbf{w}_k - \mathbf{w}_*\|_2 \leq \left(\frac{L/\mu - 1}{L/\mu + 1}\right)^k \|\mathbf{w}_0 - \mathbf{w}_*\|_2, \quad \mathbf{w}_* \text{ is the minimum.}$$

2. $f(\mathbf{w})$ is convex, let $s = 1/L$, we have

$$f(\mathbf{w}_k) - f(\mathbf{w}_*) \leq \frac{2L\|\mathbf{w}_0 - \mathbf{w}_*\|_2^2}{k}.$$

3. $f(\mathbf{w})$ is nonconvex, let $s = 1/L$, we have

$$\|\nabla f(\mathbf{w}_k)\|_2 \leq \sqrt{\frac{2L(f(\mathbf{w}_0) - f(\mathbf{w}_*))}{k}}.$$

A. Cauchy, 1847

Consider

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T \mathbf{L}\mathbf{w} - \mathbf{w}^T \mathbf{e}_1,$$

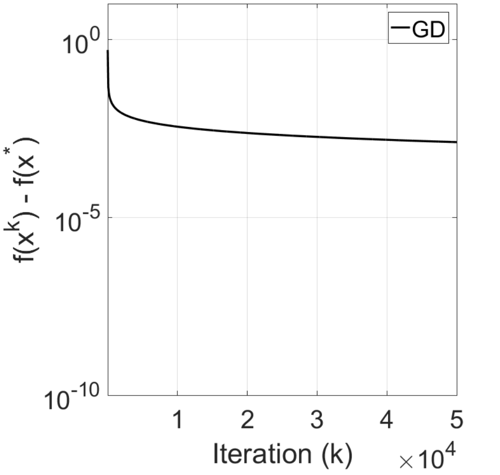
where

$$\mathbf{L} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ -1 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}_{1000\times 1000},$$

and $\mathbf{e}_1$ is a 1000-dim vector whose first entry is 1 and all the other entries are 0.

A. Nemirovski et al, 1985

Gradient Descent

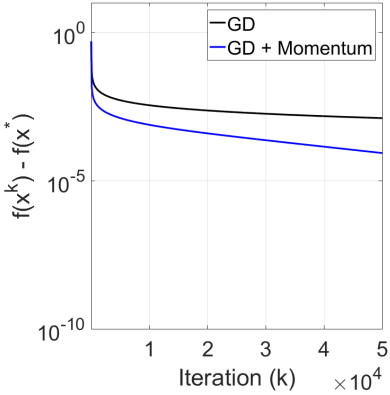


$O(1/k)$ convergence rate! Very slow!

Gradient Descent + (Lookahead/Nesterov) Momentum

$$\mathbf{v}_k = \mathbf{w}_{k-1} - s\nabla f(\mathbf{w}_{k-1}),$$

$$\mathbf{w}_k = \mathbf{v}_k + \mu(\mathbf{v}_k - \mathbf{v}_{k-1}).$$



$O(1/k)$ convergence rate!

Heavy Ball

$$\mathbf{w}_k = \mathbf{w}_{k-1} - s\nabla f(\mathbf{w}_{k-1}) + \mu(\mathbf{w}_{k-1} - \mathbf{w}_{k-2}).$$

$O\left(1/k\right)$ convergence rate!

B. Polyak, 1964

Why momentum works

High dimensional problem is usually ill-conditioned!

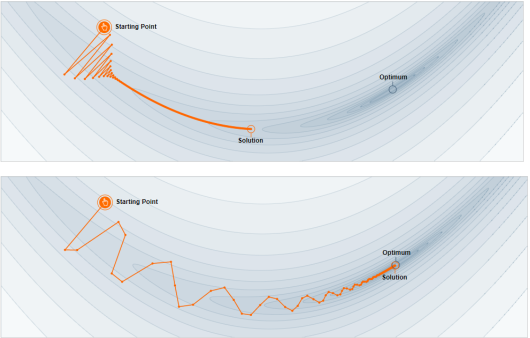


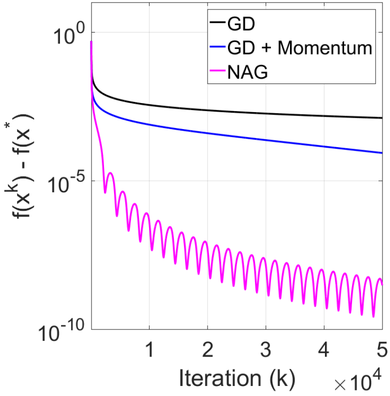
Figure: Top: no momentum; Bottom: with momentum.

Momentum smooths the trajectory and significantly speeds up gradient descent.

G. Goh, Why momentum really works. Distill, 2017

Nesterov Accelerated Gradient (NAG)

$$\mathbf{v}_k = \mathbf{w}_{k-1} - s\nabla f(\mathbf{w}_{k-1}),$$
$$\mathbf{w}_k = \mathbf{v}_k + \frac{k-1}{k+2}(\mathbf{v}_k - \mathbf{v}_{k-1}).$$



$O(1/k^2)$ convergence rate! NAG oscillates.

Nesterov Accelerated Gradient (NAG)

One of the most beautiful and mysterious results in optimization!

Not a descent method! (ripples/bumps in the traces of cost values)

Continuous dynamics

$$\ddot{X}(t) + \frac{3}{t}\dot{X}(t) + \nabla f(X(t)) = 0,$$

which satisfies $f(X(t)) - f(X^*) \leq O\left(\frac{1}{t^2}\right)$.

We can prove the above result by considering the following Lyapunov function

$$\mathcal{E}(t) := t^2(f(X(t)) - f(X^*)) + 2\|X(t) + \frac{t}{2}\dot{X}(t) - X^*\|_2^2.$$

Can we further accelerate NAG? NAG is not monotonically converge!

Y. Nesterov, 1983.

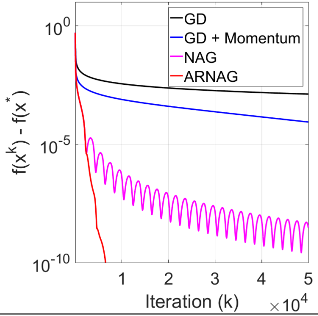
Su, Boyd, and Candes, 2014.

Adaptive Restart NAG (ARNAG)

$$\mathbf{v}_k = \mathbf{w}_{k-1} - s\nabla f(\mathbf{w}_{k-1}),$$
$$\mathbf{w}_k = \mathbf{v}_k + \frac{m(k-1)-1}{m(k-1)+2}(\mathbf{v}_k - \mathbf{v}_{k-1}),$$

where

$$m(k) = \begin{cases} m(k-1)+1, & \text{if } f(\mathbf{w}_k) \le f(\mathbf{w}_{k-1}), \\ 1, & \text{otherwise.} \end{cases}$$



V. Roulet et al. NIPS 2017

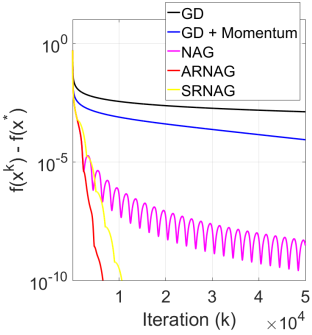
$O(\alpha^k)$, geometric convergence with convex and sharpness assumption!

Sharpness: $\frac{\mu}{r} d(\mathbf{w}, \mathbf{w}_*)^r \le f(\mathbf{w}) - f(\mathbf{w}_*), \quad \mu > 0, r > 1.$

Scheduled Restart NAG (SRNAG)

Let $(0, T] = \bigcup_{i=1}^{m} I_i = \bigcup_{i=1}^{m} (T_{i-1}, T_i]$. In each I_i, we restart the momentum after F_i iterations as follows:

$$\mathbf{v}_k = \mathbf{w}_{k-1} - s\nabla f(\mathbf{w}_{k-1}),$$
$$\mathbf{w}_k = \mathbf{v}_k + \frac{(k \bmod F_i)}{(k \bmod F_i) + 3}(\mathbf{v}_k - \mathbf{v}_{k-1}).$$



$O(\beta^k)$, geometric convergence with convex and sharpness assumption!

V. Roulet et al. NIPS 2017

What If We Do Not Have Exact Gradient?

In ERM,

$$\min_{\mathbf{w}} f(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(g(\mathbf{x}_i, \mathbf{w}), y_i),$$

when $N \gg 1$, compute $\nabla f(\mathbf{w})$ will be very expensive.

Stochastic Gradient:

$$\nabla f(\mathbf{w}) \approx \frac{1}{n}\sum_{j=1}^{n} f_{i_j}(\mathbf{w}), \text{ with } [n] \subset [N] \text{ and } n \ll N.$$

Can NAG still accelerate convergence with Stochastic Gradient?

A Motivating Example – Gaussian Noise Corrupted Gradient – Case I: Decaying Variance

Consider

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{L}\mathbf{w} - \mathbf{w}^T\mathbf{e}_1,$$

where

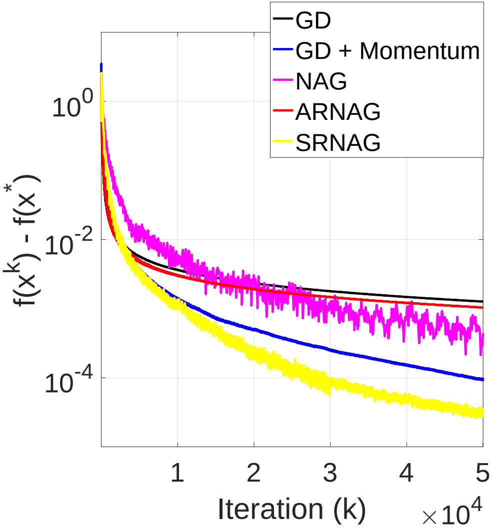
$$\mathbf{L} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ -1 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}_{1000\times 1000},$$

and $\mathbf{e}_1$ is a 1000-dim vector whose first entry is 1 and all the other entries are 0.

Gaussian Noise Corrupted Gradient:

$$\nabla f(\mathbf{w}) = \mathbf{L}\mathbf{w} - \mathbf{e}_1 + \mathbf{n}, \quad \mathbf{n} \sim \mathcal{N}(0, (\frac{0.1}{\lfloor k/100 \rfloor + 1})^2).$$

A Motivating Example – Gaussian Noise Corrupted Gradient – Case I: Decaying Variance



NAG accumulates error when an inexact gradient is used. ARNAG restarts too often and almost degenerates into GD. SRNAG performs the best.

A Motivating Example – Gaussian Noise Corrupted Gradient – Case II: Constant Variance

Consider

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{L}\mathbf{w} - \mathbf{w}^T\mathbf{e}_1,$$

where

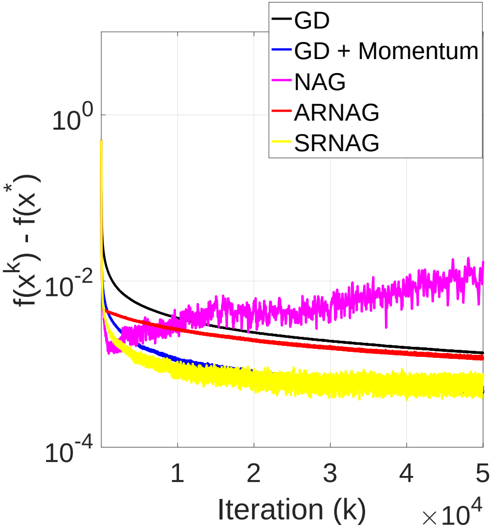
$$\mathbf{L} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ -1 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}_{1000\times 1000},$$

and $\mathbf{e}_1$ is a 1000-dim vector whose first entry is 1 and all the other entries are 0.

Gaussian Noise Corrupted Gradient:

$$\nabla f(\mathbf{w}) = \mathbf{L}\mathbf{w} - \mathbf{e}_1 + \mathbf{n}, \quad \mathbf{n} \sim \mathcal{N}(0, 0.001^2).$$

A Motivating Example – Gaussian Noise Corrupted Gradient – Case II: Constant Variance



NAG accumulates error when an inexact gradient is used. ARNAG restarts too often and almost degenerates into GD. SRNAG performs the best.

A Motivating Example – Logistic Regression – Case III

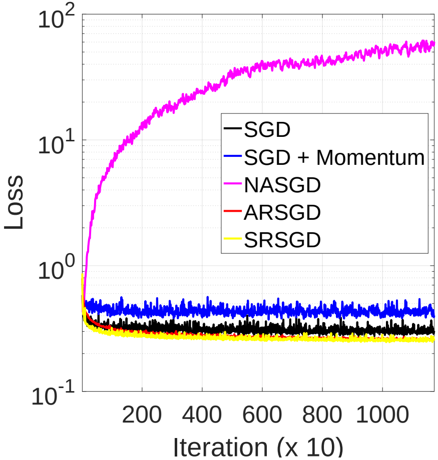


Figure: Training loss of logistic regression for MNIST classification.

NAG still accumulates error, and SRNAG performs the best.

Error Accumulation of NAG with Stochastic Gradient

Theorem Let $f(\mathbf{w})$ be a convex and L-smooth function. The sequence $\{\mathbf{w}^k\}_{k\geq 0}$ generated by NAG with mini-batch stochastic gradient using any constant step size $s \leq 1/L$, satisfies

$$\mathbb{E}\left(f(\mathbf{w}^k) - f(\mathbf{w}^*)\right) = O(k),$$

where $\mathbf{w}^*$ is the minimum of f, and the expectation is taken over the random mini-batch samples.

Nesterov Accelerated SGD accumulates error, which diverges!

B. Wang, T. Nguyen, T. Sun, A. Bertozzi, R. Baraniuk, and S. Osher, 2020.

NAG with Restart (Inexact Oracle)

Adaptive Restart NAG with Inexact Oracle: restart too often, degenerates to GD without momentum.

Scheduled Restart NAG with Inexact Oracle: appropriate restart scheduling can lead to an optimal trade-off between convergence and error accumulation.

Scheduled Restart SGD (SRSGD)

$$\mathbf{v}^k = \mathbf{w}^{k-1} - s\frac{1}{m}\sum_{j=1}^{m}\nabla f_{i_j}(\mathbf{w}^{k-1}),$$

$$\mathbf{w}^k = \mathbf{v}^k + \frac{(k \bmod F_i)}{(k \bmod F_i) + 3}(\mathbf{v}^k - \mathbf{v}^{k-1}).$$

where m is the batch size.

SRSGD resets the Nesterov momentum according to a fixed schedule when stochastic gradients are used.

B. Wang, T. Nguyen, T. Sun, A. Bertozzi, R. Baraniuk, and S. Osher, 2020.

Convergence of SRSGD

Theorem Suppose $f(\mathbf{w})$ is L-smooth. Consider the sequence $\{\mathbf{w}^k\}_{k\geq 0}$ generated by SRSGD with mini-batch stochastic gradient and any restart frequency F using any constant step size $s \leq 1/L$. Assume that the set $\mathcal{A} := \{k \in \mathbb{Z}^+ | \mathbb{E}f(\mathbf{w}^{k+1}) \geq \mathbb{E}f(\mathbf{w}^k)\}$ is finite, then we have

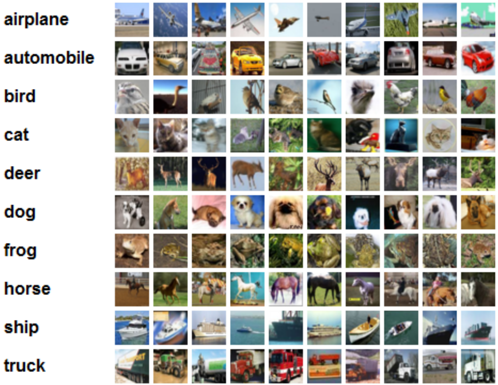
$$\min_{1\leq k\leq K} \{\mathbb{E}\|\nabla f(\mathbf{w}^k)\|_2^2\} = O(s + \frac{1}{sK}).$$

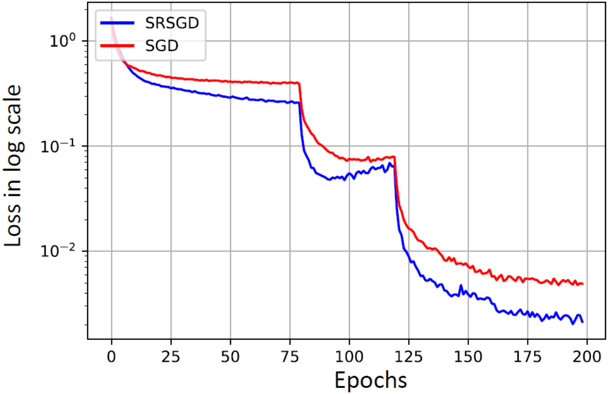
Therefore for $\forall \epsilon > 0$, to get ϵ error, we just need to set $s = O(\epsilon)$ and $K = O(1/\epsilon^2)$.

SRSGD converges even when stochastic gradients are used.

B. Wang, T. Nguyen, T. Sun, A. Bertozzi, R. Baraniuk, and S. Osher, 2020.

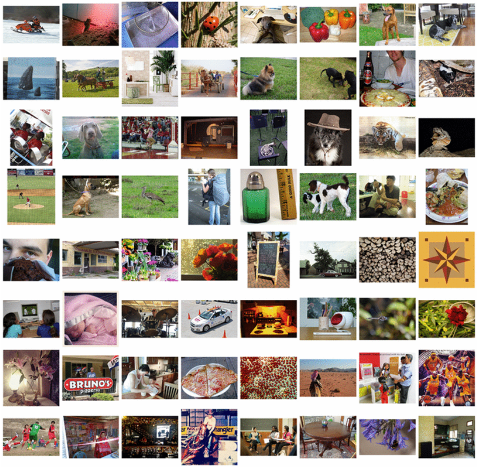
SRSGD for Deep Learning – CIFAR10/CIFAR100 Classification

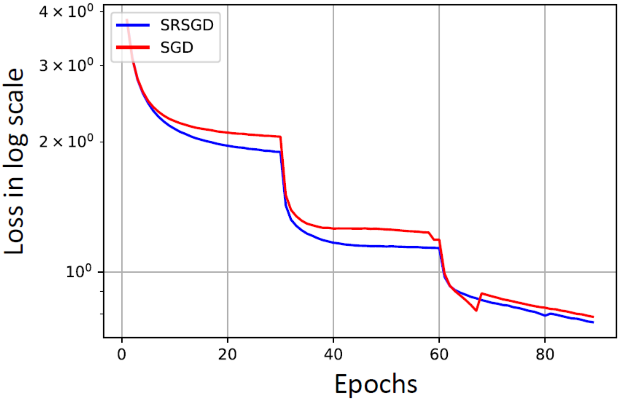




SRSGD converges faster than SGD.

SRSGD for Deep Learning – ImageNet Classification





SRSGD converges faster than SGD.

Improving Testing Error

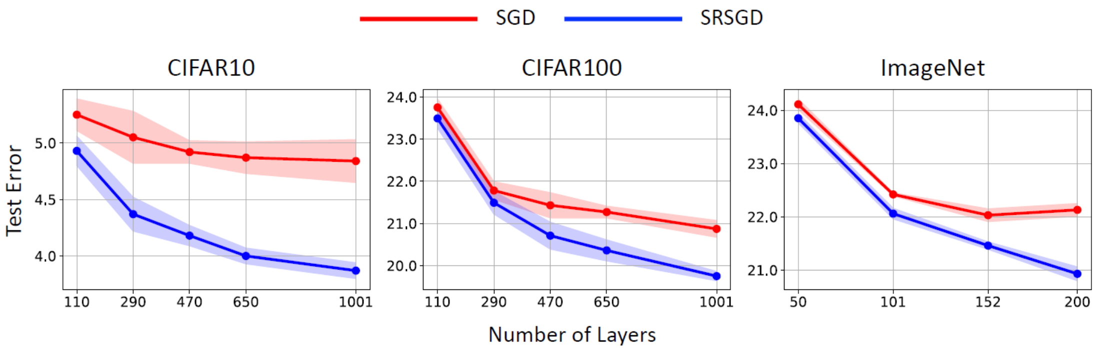


Figure: Error vs. depth of ResNet.

The improvement of SRSGD over SGD continues to grow with depth. Since SRSGD oscillates, it can escape bad minima and avoid overfitting in very deep networks.

Reducing the Training Epochs

SRSGD Training in 100 epochs
vs. SGD Training in 200 epochs

CIFAR10

Network	SRSGD	Improvement over SGD baseline
Pre-ResNet-110	5.43 ± 0.18	-0.18
Pre-ResNet-290	4.83 ± 0.11	0.22
Pre-ResNet-470	4.64 ± 0.17	0.28
Pre-ResNet-650	4.43 ± 0.14	0.44
Pre-ResNet-1001	4.17 ± 0.20	0.67
Pre-ResNet-110	5.25 ± 0.10 (110 epochs)	0.00

CIFAR100

Network	SRSGD	Improvement over SGD baseline
Pre-ResNet-110	23.85 ± 0.19	-0.10
Pre-ResNet-290	21.77 ± 0.43	0.01
Pre-ResNet-470	21.42 ± 0.19	0.01
Pre-ResNet-650	21.04 ± 0.20	0.23
Pre-ResNet-1001	20.27 ± 0.11	0.60
Pre-ResNet-110	23.73 ± 0.23 (140 epochs)	0.02

SRSGD Training with Fewer Epochs
vs. SGD Training in 90 epochs

ImageNet

Network	SRSGD	Epoch Reduction	Improvement over SGD
ResNet-50	24.30 ± 0.21	10	-0.19
ResNet-101	22.32 ± 0.06	10	0.1
ResNet-152	21.79 ± 0.07	15	0.24
ResNet-200	21.92 ± 0.17	30	0.21

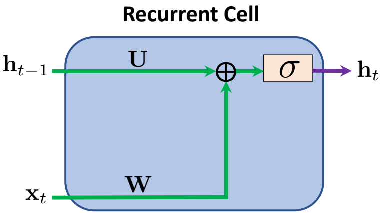
SRSGD training with fewer epochs achieves comparable results to the SGD baseline.

II. MomentumRNN: Integrating Momentum into Recurrent Neural Networks

T. Nguyen, A. Bertozzi, R. Baraniuk, S. Osher, and B. Wang, MomentumRNN: Integrating Momentum into Recurrent Neural Networks, arXiv:2006.06919, 2020.

Code: `https://github.com/minhtannguyen/MomentumRNN`

$$\underbrace{\mathbf{h}_t}_{\text{hidden state}} = \underbrace{\sigma}_{\text{e.g., sigmoid}} \Bigg(\underbrace{\mathbf{U}}_{\text{weight matrix}} \times \underbrace{\mathbf{h}_{t-1}}_{\text{hidden state}} + \underbrace{\mathbf{W}}_{\text{weight matrix}} \times \underbrace{\mathbf{x}_t}_{\text{input data}} \Bigg).$$



Universal Approximation Theorem of RNN (Informal). A RNN with enough capacity and sigmoid activation can approximate with arbitrary accuracy to the following nonlinear dynamical system

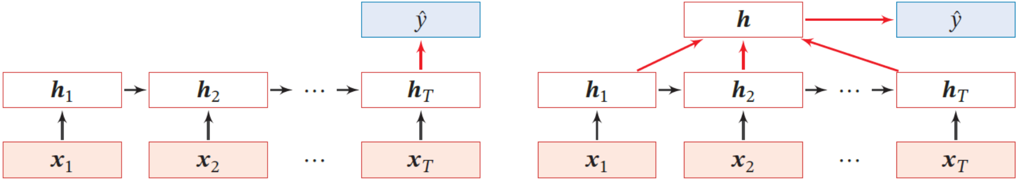
$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t),$$

where $\mathbf{h}_t$ is the hidden state at time t and $\mathbf{x}_t$ is the external input. $g(\cdot)$ is a measurable function. [a]

[a] S. Haykin, Neural networks and learning machines, 2009.

Recurrent Neural Networks – Application I

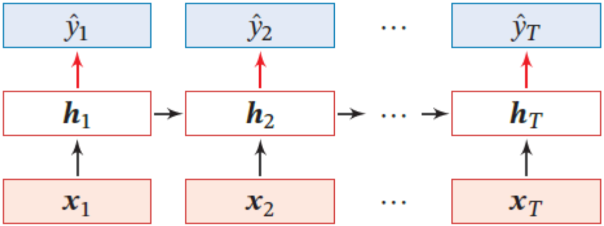
Sequence to label: Input a sequence, output a label.



Applications: text classification (left – the label is inferred from the last hidden state), image captioning (right – the label is inferred from all hidden states),...

Recurrent Neural Networks – Application II

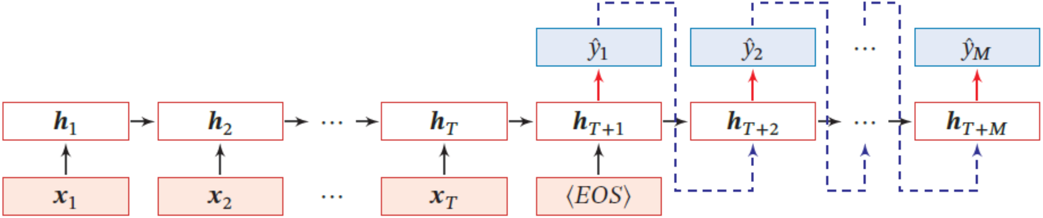
Sequence to sequence (synchronized): Input a sequence, output a sequence.



Applications: sequence labeling, part-of-speech tagging,...

Recurrent Neural Networks – Application III

Sequence to sequence (asynchronized): Input a sequence, output a sequence.



Applications: text summarization, machine translation,...

Back-propagation through time (BPTT)!

Given any training sample $(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$ being an input sequence of length T and $\mathbf{y} = (y_1, y_2, \cdots, y_T)$ being the sequence of labels. Let $\mathcal{L}_t$ be the loss at the time step t and the total loss on the whole sequence is

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t.$$

For any $1 \leq t \leq T$, we can compute the gradient of the loss $\mathcal{L}_t$ with respect to the parameter $\mathbf{U}$ as

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} = \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial \mathbf{U}} \cdot \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} \cdot \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial \mathbf{U}} \cdot \frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} \cdot \prod_{k=1}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k},$$

where $\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \mathbf{D}_k \mathbf{U}^{\mathrm{T}}$ with $\mathbf{D}_k = \mathrm{diag}\left(\sigma'(\mathbf{U}\mathbf{h}_k + \mathbf{W}\mathbf{x}_{k+1})\right)$.

$\prod_{k=1}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}$ **affects learning long-term dependency.**

Recurrent Neural Networks – Learning Long-Term Dependency?

$$\prod_{k=1}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \prod_{k=1}^{t-1} \mathbf{D}_k \mathbf{U}^{\mathrm{T}}$$

If $\|\mathbf{D}_k \mathbf{U}^{\mathrm{T}}\|_2 > 1$, $\prod_{k=1}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \to \infty$ as $t - k \to \infty$.

Solution: gradient clipping, regularize $\mathbf{U}^{\mathrm{T}}$,...

If $\|\mathbf{D}_k \mathbf{U}^{\mathrm{T}}\|_2 < 1$, $\prod_{k=1}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \to \mathbf{0}$ as $t - k \to \infty$.

Major obstacle to learning long-term dependency!

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{U}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ix}\mathbf{x}_t), && (\mathbf{i}_t : \text{input gate}) \\
\widetilde{\mathbf{c}}_t &= \tanh\left(\mathbf{U}_{\widetilde{c}h}\mathbf{h}_{t-1} + \mathbf{W}_{\widetilde{c}x}\mathbf{x}_t\right), && (\widetilde{\mathbf{c}}_t : \text{cell input}) \\
\mathbf{c}_t &= \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t, && (\mathbf{c}_t : \text{cell state}) \\
\mathbf{o}_t &= \sigma(\mathbf{U}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{ox}\mathbf{x}_t), && (\mathbf{o}_t : \text{output gate}) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh \mathbf{c}_t, && (\mathbf{h}_t : \text{hidden state})
\end{aligned}
$$

where $\mathbf{U}_* \in \mathbb{R}^{h\times h}$ and $\mathbf{W}_* \in \mathbb{R}^{h\times d}$ are learnable parameters, and $\odot$ denotes the Hadamard product.

After quite complicated computations, we can find that

$$\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \mathbf{D}_k$$

Instead of $\mathbf{D_k U}^{\mathrm{T}}$

Learning long-term dependency in LSTMs can be derived using the similar approach as in RNNs.

S. Hochreiter, J. Schmidhuber, Long short-term memory, 1997.

Enforce the matrix U to be unitary!

Efficient numerical algorithm: exponential parameterization.

M. Arjovsky, A. Shah, and Y. Bengio, Unitary Evolution Recurrent Neural Networks, ICML, 2016.
L. Mario, Trivializations for Gradient-Based Optimization on Manifolds, NIPS, 2019.

Integrating momentum into RNNs!

Background: Momentum Accelerated Dynamical System for Optimization

Consider

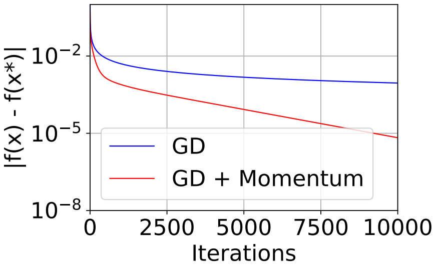
$$\min_{\mathbf{x}} f(\mathbf{x}), \ \mathbf{x} \in \mathbb{R}^d.$$

Start from $\mathbf{x}_0$, gradient descent (GD) iterates as follows

$$\mathbf{x}_t = \mathbf{x}_{t-1} - s\nabla f(\mathbf{x}_t), \ s > 0 \text{ is the step size.}$$

Momentum accelerated gradient descent

$$\mathbf{p}_0 = \mathbf{x}_0; \ \mathbf{p}_t = \mu\mathbf{p}_{t-1} + s\nabla f(\mathbf{x}_t); \ \mathbf{x}_t = \mathbf{x}_{t-1} - \mathbf{p}_t, \ u \geq 0.$$



$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{L}\mathbf{x} - \mathbf{x}^T\mathbf{e}_1, \ \mathbf{x} \in \mathbb{R}^{500}.$$

where $\mathbf{L}$ is the Laplacian of a cycle graph.

Momentum accelerates gradient descent.

Background: Momentum Accelerated Dynamical System for Sampling

Consider sampling the distribution

$$\pi \propto \exp(-f(\mathbf{x})),\ \mathbf{x} \in \mathbb{R}^d.$$

Langevin Monte Carlo (LMC)

$$\mathbf{x}_t = \mathbf{x}_{t-1} - s\nabla f(\mathbf{x}_t) + \sqrt{2s}\epsilon_t,\ s \geq 0,\ t \geq 1,\ \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d\times d}).$$

Hamiltonian Monte Carlo (HMC)

$$\mathbf{p}_0 = \mathbf{x}_0;\ \mathbf{p}_t = \mathbf{p}_{t-1} - \gamma s\mathbf{p}_{t-1} - s\eta\nabla f(\mathbf{x}_{t-1}) + \sqrt{2\gamma s\eta}\epsilon_t;\ \mathbf{x}_t = \mathbf{x}_{t-1} + s\mathbf{p}_t,\quad t \geq 1.$$

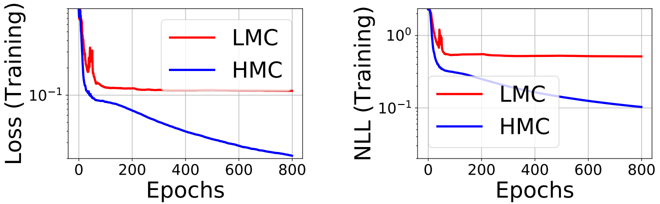


Figure: LMC vs. HMC in training Bayesian neural network for MNIST classification.

Momentum accelerates MCMC sampling.

MomentumRNN: Integrating Momentum into RNN

Let $\phi(\cdot) := \sigma(\mathbf{U}(\cdot))$ and $\mathbf{u}_t := \mathbf{U}^{-1}\mathbf{W}\mathbf{x}_t$, we can rewrite the recurrent cell as

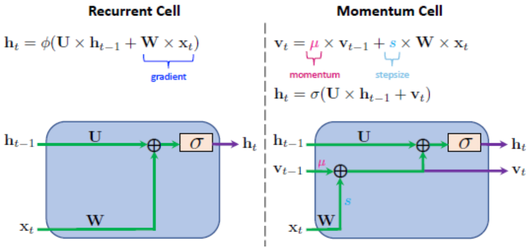
$$\mathbf{h}_t = \phi(\mathbf{h}_{t-1} + \underbrace{\mathbf{u}_t}_{\text{regard } -\mathbf{u}_t \text{ as ``gradient''}}).$$

Add momentum to the recurrent cell yields

$$\mathbf{p}_t = \mu\mathbf{p}_{t-1} - s\mathbf{u}_t; \quad \mathbf{h}_t = \phi(\mathbf{h}_{t-1} - \mathbf{p}_t).$$

Let $\mathbf{v}_t := -\mathbf{U}\mathbf{p}_t$, we get the following momentum cell

$$\mathbf{v}_t = \mu\mathbf{v}_{t-1} + s\mathbf{W}\mathbf{x}_t; \quad \mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{v}_t).$$



MomentumRNN replaces the gradient update in RNN by a momentum-based update.

MomentumRNN: Alleviating the Vanishing Gradient Issue

BPTT for RNN

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \cdot \prod_{k=t}^{T-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \cdot \prod_{k=t}^{T-1} (\mathbf{D}_k \mathbf{U}^{\mathrm{T}}).$$

BPTT for MomentumRNN

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \cdot \prod_{k=t}^{T-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \cdot \prod_{k=t}^{T-1} \widehat{\mathbf{D}}_k [\mathbf{U}^{\mathrm{T}} + \mu \mathbf{\Sigma}_k],$$

where $\widehat{\mathbf{D}}_k = \mathrm{diag}(\sigma'(\mathbf{U}(\mathbf{h}_k + \mu \mathbf{h}_{k-1}) + \mu \sigma^{-1}(\mathbf{h}_k) + s\mathbf{W}\mathbf{x}_{k+1}))$ and $\mathbf{\Sigma} = \mathrm{diag}((\sigma^{-1})'(\mathbf{h}_k))$. For mostly used σ, e.g., sigmoid and tanh, $(\sigma^{-1}(\cdot))' > 1$ and $\mu \mathbf{\Sigma}_k$ dominants $\mathbf{U}^{\mathrm{T}}$.

$\mathbf{\Sigma}_k$ is the dominating term, and choose a proper momentum constant μ in MomentumRNN helps alleviate the vanishing gradient problem.

MomentumRNN: Alleviating the Vanishing Gradient Issue – Illustration

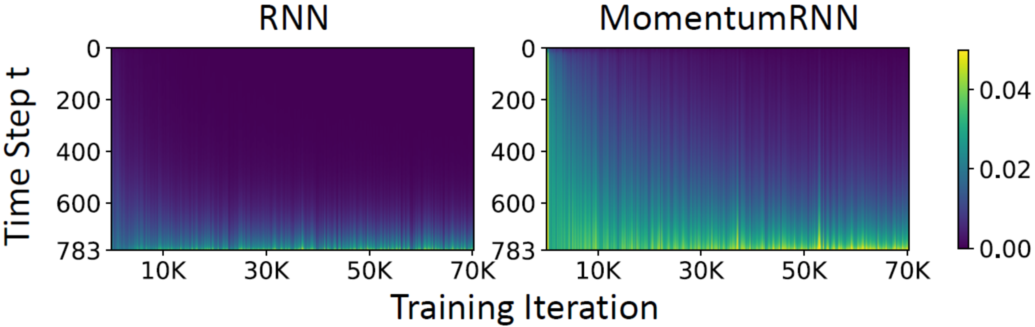


Figure: ℓ_2 norm of the gradients of the loss $\mathcal{L}$ w.r.t. the state vector $\mathbf{h}_t$ at each time step t for RNN (left) and MomentumRNN (right). Experiment: training RNN for pixel-by-pixel MNIST classification.

ℓ_2 norm of the gradients in MomentumRNN is more significant than in RNN and, therefore, helps alleviating the vanishing gradient issue.

Other MomentumRNNs – From Different Parameterizations

Let $\mathbf{v}_t = -\mathbf{p}_t$ in

$$\mathbf{p}_t = \mu \mathbf{p}_{t-1} - s\mathbf{u}_t; \quad \mathbf{h}_t = \phi(\mathbf{h}_{t-1} - \mathbf{p}_t),$$

we get

$$\mathbf{v}_t = \mu \mathbf{v}_{t-1} - s\hat{\mathbf{W}}\mathbf{x}_t; \quad \mathbf{h}_t = \phi(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{v}_t),$$

where $\hat{\mathbf{W}} := \mathbf{U}^{-1}\mathbf{W}$ is the trainable weight matrix.

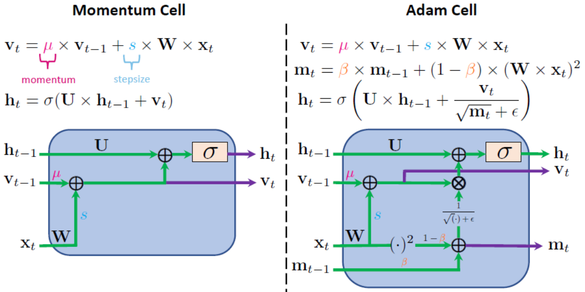
Different parameterizations can result in different momentum RNN architectures.

Other MomentumRNNs – From Different Optimization Algorithms

Nesterov Accelerated Gradient (NAG): Replace μ with $\frac{t \mod F}{(t \mod F)+3}$ with F being the restart frequency.

Adam:

$$\mathbf{v}_t = \mu \mathbf{v}_{t-1} + s\mathbf{W}\mathbf{x}_t; \quad \mathbf{m}_t = \beta \mathbf{m}_{t-1} + (1-\beta)(\mathbf{W}\mathbf{x}_t \odot \mathbf{W}\mathbf{x}_t); \quad \mathbf{h}_t = \sigma\left(\mathbf{U}\mathbf{h}_{t-1} + \frac{\mathbf{v}_t}{\sqrt{\mathbf{m}_t} + \epsilon}\right).$$



NAG Cell: set $\mu = \dfrac{k-1}{k+2}$

RMSProp Cell: set $\mu = 0$

Our momentum-based framework can take advantage of advanced optimizers to further improve RNN.

Wang, et al. Scheduled restart momentum for accelerated SGD, arXiv:2002.10583.

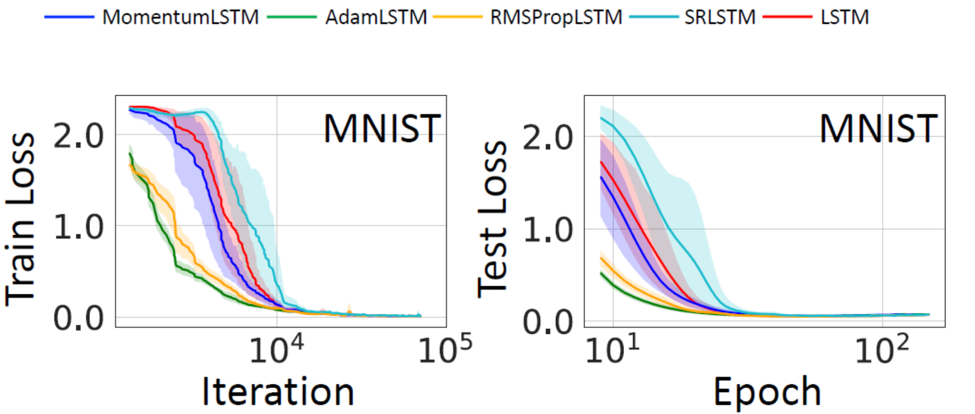
MomentumLSTMs

The momentum can also be integrated into LSTM and other RNN models easily!

Experimental Results: MomentumRNN Improves the Performance of RNN on Various Data Modalities

MomentumRNNs – Converge Faster (MNIST)

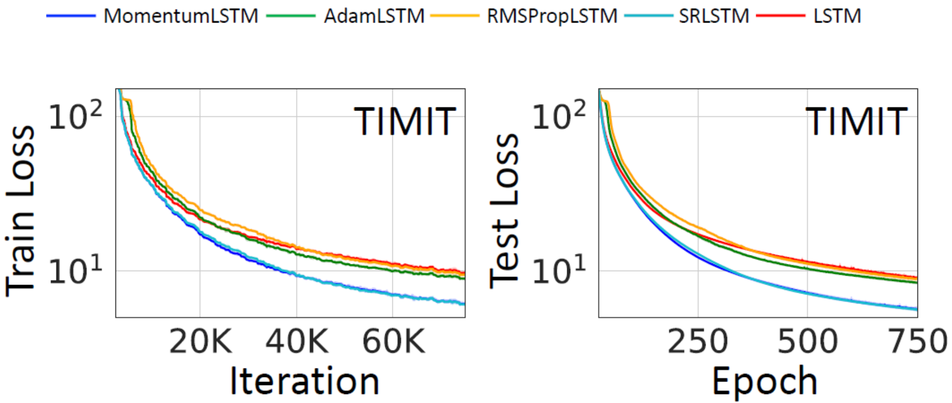
We flatten the MNIST image and feed it into the model as sequence of length 784. The original one denoted as MNIST, and we also permute the sequence and get the PMNIST dataset.



AdamLSTM and RMSPropLSTM converge fastest on MNIST tasks.

MomentumRNNs – Converge Faster (TIMIT)

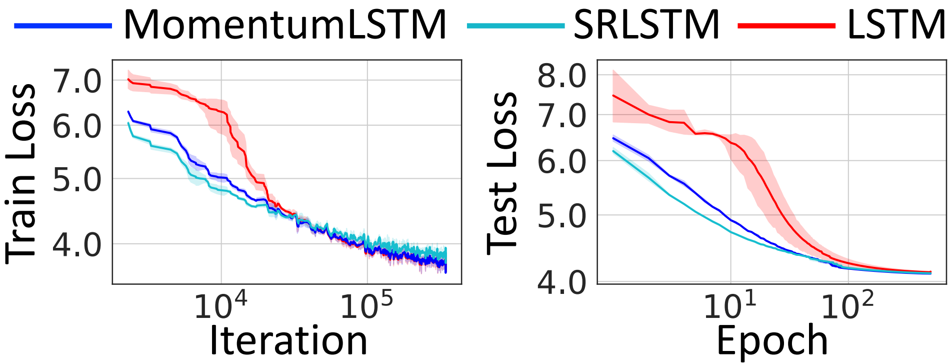
TIMIT speech dataset is a collection of real-world speech recordings. The recording are downsampled to 8kHz and then transformed into log-magnitudes via a short-time Fourier transform (STFT). The task accounts for predicting the next log-magnitude given the previous ones.



Vanilla MomentumLSTM and Scheduled Restart LSTM (SRLSTM) converge fastest on TIMIT tasks.

MomentumRNNs – Converge Faster (Word-Level Penn TreeBank)

We perform language modeling over a preprocessed PTB dataset (predict the next word). We use a three-layer LSTM model, which contains three concatenated LSTM cells.



Both MomentumLSTM and SRLSTM converges faster than the baseline LSTM on PTB tasks.

MomentumRNNs – Improve Accuracy (MNIST)

All momentum-based models achieve better accuracy than the baseline LSTM.

MomentumRNNs – Improve Accuracy (MNIST)

Table 1: Best test accuracy at the MNIST and PMNIST tasks (%). We use the baseline results reported in [21], [58], [56]. All of our proposed models outperform the baseline LSTM. Among the models using $N = 256$ hidden units, RMSPropLSTM yields the best results in both tasks.

MODEL	N	# PARAMS	MNIST	PMNIST
LSTM	128	$\approx 68K$	98.70[21],97.30 [56]	92.00 [21],92.62 [56]
LSTM	256	$\approx 270K$	98.90 [21], 98.50 [58]	92.29 [21], 92.10 [58]
MOMENTUMLSTM	128	$\approx 68K$	$\mathbf{99.04 \pm 0.04}$	$\mathbf{93.40 \pm 0.25}$
MOMENTUMLSTM	256	$\approx 270K$	$\mathbf{99.08 \pm 0.05}$	$\mathbf{94.72 \pm 0.16}$
ADAMLSTM	256	$\approx 270K$	99.09 ± 0.03	95.05 ± 0.37
RMSPROPLSTM	256	$\approx 270K$	$\mathbf{99.15 \pm 0.06}$	$\mathbf{95.38 \pm 0.19}$
SRLSTM	256	$\approx 270K$	99.01 ± 0.07	93.82 ± 1.85

RMSPropLSTM achieves the best accuracy on MNIST tasks.

MomentumRNNs – Improve Accuracy (TIMIT)

Table 2: Test and validation MSEs at the end of the epoch with the lowest validation MSE for the TIMIT task. All of our proposed models outperform the baseline LSTM. Among models using $N = 158$ hidden units, SRLSTM performs the best.

MODEL	N	# PARAMS	VAL. MSE	TEST MSE
LSTM	84	$\approx 83K$	14.87 ± 0.15 (15.42 [21, 32])	14.94 ± 0.15 (14.30 [21, 32])
LSTM	120	$\approx 135K$	11.77 ± 0.14 (13.93 [21, 32])	11.83 ± 0.12 (12.95 [21, 32])
LSTM	158	$\approx 200K$	9.33 ± 0.14 (13.66 [21, 32])	9.37 ± 0.14 (12.62 [21, 32])
MOMENTUMLSTM	84	$\approx 83K$	$\mathbf{10.90 \pm 0.19}$	$\mathbf{10.98 \pm 0.18}$
MOMENTUMLSTM	120	$\approx 135K$	$\mathbf{8.00 \pm 0.30}$	$\mathbf{8.04 \pm 0.30}$
MOMENTUMLSTM	158	$\approx 200K$	$\mathbf{5.86 \pm 0.14}$	$\mathbf{5.87 \pm 0.15}$
ADAMLSTM	158	$\approx 200K$	8.66 ± 0.15	8.69 ± 0.14
RMSPROPLSTM	158	$\approx 200K$	9.13 ± 0.33	9.17 ± 0.33
SRLSTM	158	$\approx 200K$	$\mathbf{5.81 \pm 0.10}$	$\mathbf{5.83 \pm 0.10}$

SRLSTM achieves the best accuracy on TIMIT tasks.

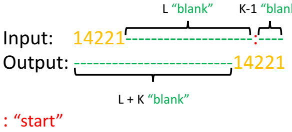
MomentumRNNs – Improve Accuracy (Penn TreeBank)

Table 3: Model test perplexity at the end of the epoch with the lowest validation perplexity for the Penn Treebank language modeling task (word level).

MODEL	# PARAMS	VAL. PPL	TEST PPL
LSTM	$\approx 24M$	61.96 ± 0.83	59.71 ± 0.99 (58.80 [34])
MOMENTUMLSTM	$\approx 24M$	$\mathbf{60.71 \pm 0.24}$	$\mathbf{58.62 \pm 0.22}$
SRLSTM	$\approx 24M$	61.12 ± 0.68	58.83 ± 0.62

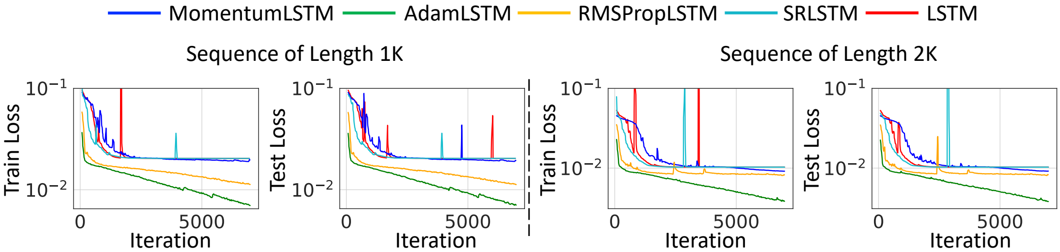
MomentumLSTM achieves the best accuracy on PTB tasks.

MomentumRNNs – Converge Faster and Achieve Better Loss (Copying Task)



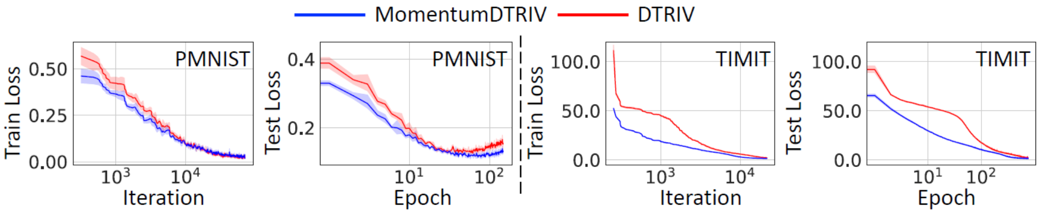
1. Consider set A of N alphabets, e.g. A = {1,2,3,4}, N=4
2. The alphabet character sequence of length K is sampled i.i.d. uniformly from A, e.g. 14221, K=5
3. The input is the character sequence followed by L “blank” characters, a “start” character, and then K-1 “blank” characters.

Task: output a sequence containing K + L “blank” characters followed by the alphabet character sequence, e.g. 14211



AdamLSTM significantly outperforms other models.

MomentumDTRIV – Integrate Momentum into Orthogonal RNN



MomentumDTRIV converges faster than DTRIV.

MomentumDTRIV – Integrate Momentum into Orthogonal RNN

Table 4: Best test accuracy on the PMNIST tasks (%) for MomentumDTRIV and DTRIV. We provide both our reproduced baseline results and those reported in [6]. MomentumDTRIV yields better results than the baseline DTRIV in all settings.

N	# PARAMS	PMNIST (DTRIV)	PMNIST (MOMENTUMDTRIV)
170	$\approx 16K$	95.21 ± 0.10 (95.20 [6])	$\mathbf{95.37 \pm 0.09}$
360	$\approx 69K$	96.45 ± 0.10 (96.50 [6])	$\mathbf{96.73 \pm 0.08}$
512	$\approx 137K$	96.62 ± 0.12 (96.80 [6])	$\mathbf{96.89 \pm 0.08}$

Table 5: Test and validation MSE of MomentumDTRIV vs. DTRIV at the epoch with the lowest validation MSE for the TIMIT task. MomentumDTRIV yields much better results than DTRIV.

MODEL	N	# PARAMS	VAL. MSE	TEST MSE
DTRIV	224	$\approx 83K$	4.74 ± 0.06 (4.75 [6])	4.70 ± 0.07 (4.71 [6])
DTRIV	322	$\approx 135K$	1.92 ± 0.17 (3.39 [6])	1.87 ± 0.17 (3.76 [6])
MOMENTUMDTRIV	224	$\approx 83K$	$\mathbf{3.10 \pm 0.09}$	$\mathbf{3.06 \pm 0.09}$
MOMENTUMDTRIV	322	$\approx 135K$	$\mathbf{1.21 \pm 0.05}$	$\mathbf{1.17 \pm 0.05}$

MomentumDTRIV achieves better accuracy than DTRIV.

Computational Time Analysis

Computation Time per Sample when Evaluating on PMNIST

Model	Training Time (μs/sample)	Evaluation Time (μs/sample)
LSTM	6.18	2.52
MomentumLSTM	7.43	3.16
AdamLSTM	10.34	4.07
RMSPropLSTM	9.94	3.96
SRLSTM	8.34	3.16

Total computation time to reach the same 92.29% test accuracy of LSTM when Evaluating on PMNIST

Model	Time (*seconds*)
LSTM	46015
MomentumLSTM	33036
AdamLSTM	**13484**
RMSPropLSTM	24931
SRLSTM	20881

Taking the whole training into account, Momentum-based LSTMs are much more efficient than the baseline LSTM.

Thank You

I. Scheduled Restart NAG Momentum

- Accelerate convergence
- Better generalization accuracy

II. MomentumRNN

- Mitigating the vanishing gradient issue
- Speed-up training of RNNs
- Improve performance of the trained RNNs

1. B. Wang, T. Nguyen, T. Sun, A. Bertozzi, R. Baraniuk, and S. Osher, arXiv:2002.10583, 2020.

2. T. Nguyen, R. Baraniuk, A. Bertozzi, S. Osher, and B. Wang, arXiv:2006.06919, 2020.