Wang et al. *Res Math Sci(2022)9:57*
https://doi.org/10.1007/s40687-022-00352-0

Research in the Mathematical Sciences

RESEARCH

How does momentum benefit deep neural networks architecture design? A few case studies

Bao Wang[1*], Hedi Xia[2], Tan Nguyen[2] and Stanley Osher[2]

*Correspondence: wangbaonj@gmail.com
[1]Department of Mathematics, Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA
Full list of author information is available at the end of the article

Abstract

We present and review an algorithmic and theoretical framework for improving neural network architecture design via momentum. As case studies, we consider how momentum can improve the architecture design for recurrent neural networks (RNNs), neural ordinary differential equations (ODEs), and transformers. We show that integrating momentum into neural network architectures has several remarkable theoretical and empirical benefits, including (1) integrating momentum into RNNs and neural ODEs can overcome the vanishing gradient issues in training RNNs and neural ODEs, resulting in effective learning long-term dependencies; (2) momentum in neural ODEs can reduce the stiffness of the ODE dynamics, which significantly enhances the computational efficiency in training and testing; (3) momentum can improve the efficiency and accuracy of transformers.

1 Introduction

Deep learning has radically advanced artificial intelligence [49], which has achieved state-of-the-art performance in various applications, including computer vision [23,99], natural language processing [9,22], and control [87]. Nevertheless, deep neural network (DNNs) designs are mostly heuristic, and the resulting architectures have many well-known issues: (1) convolutional neural networks (CNNs) are not robust to unperceptible adversarial attacks [92]; (2) recurrent neural networks (RNNs) cannot learn long-term dependencies effectively due to vanishing gradients [69]; (3) training neural ordinary differential equations (ODEs) can take an excessive number of function evaluations (NFEs) [14]; and (4) training transformers suffers from quadratic computational time and memory costs [101]. See Sects. 2, 3, and 4, respectively, for the details of these problems (2)–(4).

Addressing the above grand challenges is at the forefront of deep learning research. (1) Adversarial defense [56] has been proposed to train robust neural networks against adversarial attacks; a survey of adversarial defense algorithms is available, see, e.g., [12]. Training ResNets has also been interpreted as solving a control problem of the transport equation [105,109], resulting in PDE-motivated adversarial defenses [104,107,109]. (2) Learning long-term dependencies with improved RNNs has been an active research area for sev-

eral decades; celebrated works include long short-term memory [37] and gated recurrent unit [18]. (3) Several algorithms and techniques have been proposed to reduce NFEs in training neural ODEs, including input augmentation [25], regularizing solvers and learning dynamics [27,29,41,66], high-order ODE [65], data control [57], and depth variance [57]. (4) Transformers are the current state-of-the-art machine learning (ML) models for sequential learning [101], which processes the input sequence concurrently and can learn long-term dependencies effectively. However, transformers suffer from quadratic computational time and memory costs with respect to the input sequence length; see Sect. 4 for details. In response, efficient attention has been proposed leveraging sparse and low-rank approximation of the attention matrix [1,5,17,40,54,68,110,116], locality-sensitive hashing [45], clustered attention [103], and decomposed near-field and far-field attention [63].

1.1 Background: momentum acceleration for gradient descent

In this subsection, we review several well-established momentum techniques for accelerating gradient descent. Let us first recall the heavy-ball momentum, a.k.a. classical momentum [71], for accelerating gradient descent in solving $\min_{\boldsymbol{x}\in\mathbb{R}^d} F(\boldsymbol{x})$. Starting from $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$, the heavy-ball method iterates as follows:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - s\nabla F(\boldsymbol{x}^k) + \beta(\boldsymbol{x}^k - \boldsymbol{x}^{k-1}), \tag{1}$$

where $s > 0$ is the step size and $0 \le \beta < 1$ is the momentum parameter. By introducing the momentum state $\boldsymbol{m}$, we can rewrite the HB method as

$$\boldsymbol{m}^{k+1} = \beta\boldsymbol{m}^k + \nabla F(\boldsymbol{x}^k); \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k - s\boldsymbol{m}^{k+1}. \tag{2}$$

In contrast, gradient descent updates at each step as follows

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - s\nabla F(\boldsymbol{x}^k). \tag{3}$$

Another momentum due to Nesterov [60] accelerates gradient descent via the following iteration

$$\boldsymbol{m}^{k+1} = \boldsymbol{x}^k - s\nabla F(\boldsymbol{x}^k); \quad \boldsymbol{x}^{k+1} = \boldsymbol{m}^{k+1} + \beta(\boldsymbol{m}^{k+1} - \boldsymbol{m}^k), \tag{4}$$

where $s > 0$ and $0 \le \beta < 1$ are defined the same as that in (2). If we replace the constant β in (4) with a time-dependent coefficient, e.g., $(k-1)/(k+2)$ [90], we arrive at the celebrated Nesterov's accelerated gradient.

HB method (2) can also be integrated with adaptive step size via the second moment, resulting in the following celebrated Adam algorithm [44][1]

$$\begin{aligned} \boldsymbol{m}^{k+1} &= \beta\boldsymbol{m}^k + (1-\beta)\nabla F(\boldsymbol{x}^k); \\ \boldsymbol{v}^{k+1} &= \alpha\boldsymbol{v}^k + (1-\alpha)[\nabla F(\boldsymbol{x}^k)]^2; \\ \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k - s\frac{\boldsymbol{m}^{k+1}}{[\boldsymbol{v}^{k+1}]^{1/2}}, \end{aligned} \tag{5}$$

[1] Here, for the sake of exposition, we omit the bias corrections.

where $s > 0$ and $0 \leq \beta < 1$ are two parameters inherited from (2), and $0 \leq \alpha < 1$ is another parameter. $[\cdot]^2$ and $[\cdot]^{1/2}$ denote the element-wise square and square-root, respectively.

1.2 Contribution

This paper aims to present and review an algorithmic and theoretical framework for improving neural network architecture design via momentum, a well-established first-order optimization tool [61,71]. In particular, we focus on leveraging momentum to design new RNNs and neural ODEs to accelerate their training and testing and improve learning long-term dependencies with theoretical guarantees. Moreover, we present a new efficient attention mechanism with momentum augmentation, which significantly improves computational efficiency over transformers [101] and accuracy over linear transformers [40]. Finally, we present some perspectives of how momentum can further improve neural networks design and solve existing grand challenges.

1.3 Notations

We denote scalars by lower- or upper-case letters. We also denote vectors and matrices by lower- and upper-case boldface letters, respectively. For a vector $\boldsymbol{x} = (x_1, \cdots, x_d)^\top \in \mathbb{R}^d$, where $(x_1, \cdots, x_d)^\top$ denotes the transpose of the vector $(x_1, \cdots, x_d)$, we use $\|\boldsymbol{x}\| = (\sum_{i=1}^d |x_i|^2)^{1/2}$ to denote its ℓ_2 norm. We denote the vector whose entries are all 0s as $\mathbf{0}$. For a matrix $\boldsymbol{A}$, we use $\boldsymbol{A}^\top$, $\boldsymbol{A}^{-1}$, and $\|\boldsymbol{A}\|$ to denote its transpose, inverse, and spectral norm, respectively. We use $\boldsymbol{I}$ to denote the identity matrix, whose dimension can be determined in its context. For a function $f(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$, we denote its gradient as $\nabla f(\boldsymbol{x})$. Given two sequences $\{a_n\}$ and $\{b_n\}$, we write $a_n = \mathcal{O}(b_n)$ if there exists a positive constant $0 < C < +\infty$ such that $a_n \leq Cb_n$.

1.4 Organization

We organize this paper as follows: In Sect. 2, we present RNN models and their difficulties in learning long-term dependencies. We also show how to integrate momentum into RNNs to accelerate training RNNs and enhance RNNs' capability in learning long-term dependencies. In Sect. 3, we show how the ODE limit of momentum can improve neural ODEs in terms of training and test efficiency and learning long-term dependencies. In Sect. 4, we show how momentum can be integrated into efficient transformers and improve their accuracy. We conclude and present potential new directions in Sect. 5.

2 Recurrent neural networks

In this section, we present how to leverage momentum to improve RNN architecture design. The main results have been presented at NeurIPS 2020 [62]. In Sect. 2.1, we recap the architectures of RNN and LSTM in their standard forms to provide readers with some essential background on RNN and LSTM. In Sect. 2.2, we rewrite the standard form of the RNN into a different form to match the form of gradient descent, which motivates the natural approach to integrating momentum into RNNs. In Sect. 2.3, we show that integrating momentum into RNNs can effectively alleviate the vanishing gradient issue in training RNNs using backpropagation through time. We discuss how to integrate other types of momentum into RNNs and integrate momentum into LSTMs in Sects. 2.4 and

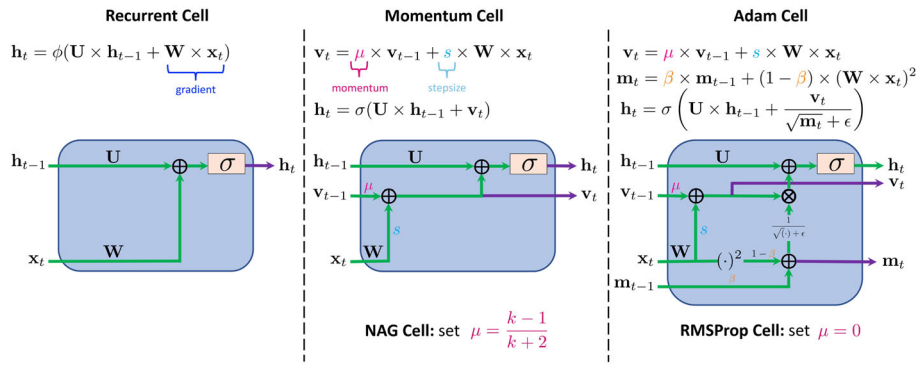

Fig. 1 Illustration of the recurrent cell (left), momentum/NAG cell (middle), and Adam/RMSProp cell (right). We draw a connection between the dynamics of hidden states in the recurrent cell and GD. We then introduce momentum to recurrent cell as an analogy of the momentum-accelerated GD

2.5, respectively. Finally, we verify the efficacy of the RNNs and LSTMs with momentum integration in Sect. 2.6.

2.1 Recap on RNNs and LSTM

Recurrent cells are the building blocks of RNNs. A recurrent cell employs a cyclic connection to update the current hidden state ($\boldsymbol{h}_t$) using the past hidden state ($\boldsymbol{h}_{t-1}$) and the current input data ($\boldsymbol{x}_t$) [26]; the dependence of $\boldsymbol{h}_t$ on $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ in a recurrent cell can be written as

$$\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}),\ \boldsymbol{x}_t \in \mathbb{R}^d,\ \text{and } \boldsymbol{h}_{t-1}, \boldsymbol{h}_t \in \mathbb{R}^h,\ \ t = 1, 2, \cdots, T, \tag{6}$$

where $\boldsymbol{U} \in \mathbb{R}^{h\times h}$, $\boldsymbol{W} \in \mathbb{R}^{h\times d}$, and $\boldsymbol{b} \in \mathbb{R}^h$ are trainable parameters; $\sigma(\cdot)$ is a nonlinear activation function, e.g., sigmoid or hyperbolic tangent; see Fig. 1 (left) for an illustration of the RNN cell. Error backpropagation through time is used to train RNN, but it tends to result in exploding or vanishing gradients [6]. Thus, RNNs may fail to learn long-term dependencies.

LSTM cells augment the recurrent cell with "gates" [37] and results in

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{U}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i), && (\boldsymbol{i}_t : \text{input gate})\\
\widetilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{U}_{\widetilde{c}h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\widetilde{c}x}\boldsymbol{x}_t + \boldsymbol{b}_{\widetilde{c}}), && (\widetilde{\boldsymbol{c}}_t : \text{cell input})\\
\boldsymbol{c}_t &= \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}}_t, && (\boldsymbol{c}_t : \text{cell state})\\
\boldsymbol{o}_t &= \sigma(\boldsymbol{U}_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t + \boldsymbol{b}_o), && (\boldsymbol{o}_t : \text{output gate})\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh \boldsymbol{c}_t, && (\boldsymbol{h}_t : \text{hidden state})
\end{aligned} \tag{7}$$

where $\boldsymbol{U}_* \in \mathbb{R}^{h\times h}$, $\boldsymbol{W}_* \in \mathbb{R}^{h\times d}$, and $\boldsymbol{b}_* \in \mathbb{R}^h$ are learnable parameters, and $\odot$ denotes the Hadamard product. The input gate decides what new information to be stored in the cell state, and the output gate decides what information to output based on the cell state value. The gating mechanism in LSTMs can lead to the issue of saturation [13,100].

2.2 Gradient descent analogy for RNN and MomentumRNN

Now, we are going to establish a connection between RNN and gradient descent and further leverage momentum to improve RNNs. Let $\widetilde{\boldsymbol{W}} = [\boldsymbol{W}, \boldsymbol{b}]$ and $\widetilde{\boldsymbol{x}}_t = [\boldsymbol{x}_t, 1]^\top$ in (6), then we have $\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{h}_{t-1} + \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{x}}_t)$. For the ease of notation, without ambiguity we denote $\boldsymbol{W} := \widetilde{\boldsymbol{W}}$ and $\boldsymbol{x}_t := \widetilde{\boldsymbol{x}}_t$. Then, the recurrent cell can be reformulated as:

$$\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t). \tag{8}$$

Moreover, let $\phi(\cdot) := \sigma(\boldsymbol{U}(\cdot))$ and $\boldsymbol{u}_t := \boldsymbol{U}^{-1}\boldsymbol{W}\boldsymbol{x}_t$, we can rewrite (8) as

$$\boldsymbol{h}_t = \phi(\boldsymbol{h}_{t-1} + \boldsymbol{u}_t). \tag{9}$$

If we regard $-\boldsymbol{u}_t$ as the "gradient" at the t-th iteration, then we can consider (9) as the dynamical system which updates the hidden state by the gradient and then transforms the updated hidden state by the nonlinear activation function ϕ. We propose the following accelerated dynamical system to accelerate the dynamics of (9), which is principled by the accelerated gradient descent theory (see Sect. 1.1):

$$\boldsymbol{p}_t = \mu\boldsymbol{p}_{t-1} - s\boldsymbol{u}_t; \;\; \boldsymbol{h}_t = \phi(\boldsymbol{h}_{t-1} - \boldsymbol{p}_t), \tag{10}$$

where $\mu \geq 0, s > 0$ are two hyperparameters, which are the analogies of the momentum coefficient and step size in the momentum-accelerated GD, respectively. Let $\boldsymbol{v}_t := -\boldsymbol{U}\boldsymbol{p}_t$; we arrive at the following dynamical system:

$$\boldsymbol{v}_t = \mu\boldsymbol{v}_{t-1} + s\boldsymbol{W}\boldsymbol{x}_t; \;\; \boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{v}_t). \tag{11}$$

The architecture of the momentum cell that corresponds to the dynamical system (11) is plotted in Fig. 1 (middle). Compared with the recurrent cell, the momentum cell introduces an auxiliary momentum state in each update and scales the dynamical system with two positive hyperparameters μ and s.

Remark 1 Different parameterizations of (10) can result in different momentum cell architectures. For instance, if we let $\boldsymbol{v}_t = -\boldsymbol{p}_t$, we end up with the following dynamical system:

$$\boldsymbol{v}_t = \mu\boldsymbol{v}_{t-1} + s\widehat{\boldsymbol{W}}\boldsymbol{x}_t; \;\; \boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{v}_t), \tag{12}$$

where $\widehat{\boldsymbol{W}} := \boldsymbol{U}^{-1}\boldsymbol{W}$ is the trainable weight matrix. Even though (11) and (12) are mathematically equivalent, the training procedure might cause the MomentumRNNs that are derived from different parameterizations to have different performances.

Remark 2 We put the nonlinear activation in the second equation of (10) to ensure that the value of $\boldsymbol{h}_t$ is in the same range as the original recurrent cell.

Remark 3 The derivation above also applies to the dynamical systems in the LSTM cells, and we can design the MomentumLSTM in the same way as designing the MomentumRNN.

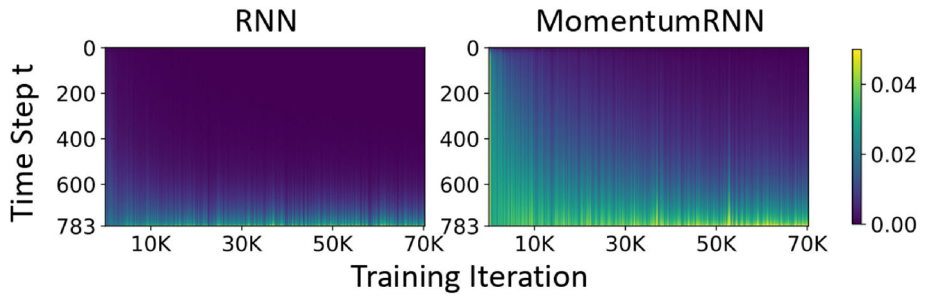

Fig. 2 ℓ_2 norm of the gradients of the loss $\mathcal{L}$ w.r.t. the state vector $\boldsymbol{h}_t$ at each time step t for RNN (left) and MomentumRNN (right). MomentumRNN does not suffer from vanishing gradients

2.3 Analysis of the vanishing gradient: momentum cell vs. recurrent cell

Let $\boldsymbol{h}_T$ and $\boldsymbol{h}_t$ be the state vectors at the time step T and t, respectively, and we suppose $T \gg t$. Furthermore, assume that $\mathcal{L}$ is the objective to minimize, then

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} (\boldsymbol{D}_k \boldsymbol{U}^\top), \tag{13}$$

where $\boldsymbol{U}^\top$ is the transpose of $\boldsymbol{U}$ and $\boldsymbol{D}_k = \mathrm{diag}(\sigma'(\boldsymbol{U}\boldsymbol{h}_k + \boldsymbol{W}\boldsymbol{x}_{k+1}))$ is a diagonal matrix with $\sigma'(\boldsymbol{U}\boldsymbol{h}_k + \boldsymbol{W}\boldsymbol{x}_{k+1})$ being its diagonal entries. $\| \prod_{k=t}^{T-1} (\boldsymbol{D}_k \boldsymbol{U}^\top) \|_2$ tends to either vanish or explode [6]. We can use regularization or gradient clipping to mitigate the exploding gradient, leaving vanishing gradient as the major obstacle to training RNN to learn long-term dependency [69]. We can rewrite (11) as

$$\boldsymbol{h}_t = \sigma\left(\boldsymbol{U}(\boldsymbol{h}_{t-1} - \mu \boldsymbol{h}_{t-2}) + \mu \sigma^{-1}(\boldsymbol{h}_{t-1}) + s\boldsymbol{W}\boldsymbol{x}_t\right), \tag{14}$$

where $\sigma^{-1}(\cdot)$ is the inverse function of $\sigma(\cdot)$. We compute $\partial \mathcal{L}/\partial \boldsymbol{h}_t$ as follows

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} \widehat{\boldsymbol{D}}_k [\boldsymbol{U}^\top + \mu \boldsymbol{\Sigma}_k], \tag{15}$$

where $\widehat{\boldsymbol{D}}_k = \mathrm{diag}(\sigma'(\boldsymbol{U}(\boldsymbol{h}_k - \mu \boldsymbol{h}_{k-1}) + \mu \sigma^{-1}(\boldsymbol{h}_k) + s\boldsymbol{W}\boldsymbol{x}_{k+1}))$ and $\boldsymbol{\Sigma} = \mathrm{diag}((\sigma^{-1})'(\boldsymbol{h}_k))$. For mostly used σ, e.g., sigmoid and tanh, $(\sigma^{-1}(\cdot))' > 1$ and $\mu \boldsymbol{\Sigma}_k$ dominates $\boldsymbol{U}^\top$.[2] Therefore, with an appropriate choice of μ, the momentum cell can alleviate vanishing gradient and accelerate training.

We empirically corroborate that momentum cells can alleviate vanishing gradients by training a MomentumRNN and its corresponding RNN on the PMNIST classification task and plot $\|\partial \mathcal{L}/\partial \boldsymbol{h}_t\|_2$ for each time step t. Figure 2 confirms that unlike in RNN, the gradients in MomentumRNN do not vanish.

2.4 Beyond MomentumRNN: NAG and Adam principled RNNs

There are several other advanced formalisms of momentum existing in optimization, which can be leveraged for RNN architecture design. In this subsection, we present two

[2] In the vanishing gradient scenario, $\|\boldsymbol{U}\|_2$ is small; also it can be controlled by regularizing the loss function.

additional variants of MomentumRNN that are derived from the Nesterov-accelerated gradient (NAG)-style momentum with restart [61,106] and Adam [44].

NAG-Principled RNNs. The momentum-accelerated GD can be further accelerated by replacing the constant momentum coefficient μ in (11) with the NAG-style momentum, i.e., setting μ to $(t-1)/(t+2)$ at the t-th iteration. Furthermore, we can accelerate NAG by resetting the momentum to 0 after every F iterations, i.e., $\mu = (t \mod F)/((t \mod F)+3)$, which is the NAG-style momentum with a scheduled restart of the appropriately selected frequency F [106]. For convex optimization, NAG has a convergence rate $O(1/t^2)$, which is significantly faster than GD or GD with constant momentum whose convergence rate is $O(1/t)$. Scheduled restart not only accelerates NAG to a linear convergence rate $O(\alpha^{-t})(0 < \alpha < 1)$ under mild extra assumptions but also stabilizes the NAG iteration [106]. We call the MomentumRNN with the NAG-style momentum and scheduled restart momentum the NAG-based RNN and the scheduled restart RNN (SRRNN), respectively.

Adam Principled RNNs. Adam [44] leverages the moving average of historical gradients and entry-wise squared gradients to accelerate the stochastic gradient dynamics. We use Adam to accelerate (9) and end up with the following iteration

$$\begin{aligned} \boldsymbol{p}_t &= \mu \boldsymbol{p}_{t-1} + (1-\mu)\boldsymbol{u}_t, \\ \boldsymbol{m}_t &= \beta \boldsymbol{m}_{t-1} + (1-\beta)\boldsymbol{u}_t \odot \boldsymbol{u}_t, \\ \boldsymbol{h}_t &= \phi\left(\boldsymbol{h}_{t-1} - s\frac{\boldsymbol{p}_t}{\sqrt{\boldsymbol{r}_t + \epsilon}}\right), \end{aligned} \tag{16}$$

where $\mu, s, \beta > 0$ are hyperparameters, ϵ is a small constant and chosen to be 10^{-8} by default, and $\odot/\sqrt{\cdot}$ denotes the entry-wise product/square root[3]. Again, let $\boldsymbol{v}_t = -\boldsymbol{U}\boldsymbol{p}_t$, we rewrite (16) as follows

$$\begin{aligned} \boldsymbol{v}_t &= \mu \boldsymbol{v}_{t-1} + (1-\mu)\boldsymbol{W}\boldsymbol{x}_t, \\ \boldsymbol{m}_t &= \beta \boldsymbol{m}_{t-1} + (1-\beta)\boldsymbol{u}_t \odot \boldsymbol{u}_t, \\ \boldsymbol{h}_t &= \sigma\left(\boldsymbol{U}\boldsymbol{h}_{t-1} + s\frac{\boldsymbol{v}_t}{\sqrt{\boldsymbol{m}_t + \epsilon}}\right). \end{aligned}$$

As before, here $\boldsymbol{u}_t := \boldsymbol{U}^{-1}\boldsymbol{W}\boldsymbol{x}_t$. Computing $\mathbf{U}^{-1}$ is expensive. Our experiments suggest that replacing $\boldsymbol{u}_t \odot \boldsymbol{u}_t$ by $\boldsymbol{W}\boldsymbol{x}_t \odot \boldsymbol{W}\boldsymbol{x}_t$ is sufficient and more efficient to compute. In our implementation, we also relax $\boldsymbol{v}_t = \mu\boldsymbol{v}_{t-1} + (1-\mu)\boldsymbol{W}\boldsymbol{x}_t$ to $\boldsymbol{v}_t = \mu\boldsymbol{v}_{t-1} + s\boldsymbol{W}\boldsymbol{x}_t$ that follows the momentum in the MomentumRNN (11) for better performance. Therefore, we propose the *AdamRNN* that is given by

$$\begin{aligned} \boldsymbol{v}_t &= \mu \boldsymbol{v}_{t-1} + s\boldsymbol{W}\boldsymbol{x}_t, \\ \boldsymbol{m}_t &= \beta \boldsymbol{m}_{t-1} + (1-\beta)(\boldsymbol{W}\boldsymbol{x}_t \odot \boldsymbol{W}\boldsymbol{x}_t), \\ \boldsymbol{h}_t &= \sigma\left(\boldsymbol{U}\boldsymbol{h}_{t-1} + \frac{\boldsymbol{v}_t}{\sqrt{\boldsymbol{m}_t + \epsilon}}\right). \end{aligned} \tag{17}$$

In AdamRNN, if μ is set to 0, we achieve another new RNN, which obeys the RMSProp gradient update rule [97]; which we call *RMSPropRNN*.

[3] In contrast to Adam, we do not normalize $\boldsymbol{p}_t$ and $\boldsymbol{m}_t$ since they can be absorbed in the weight matrices.

Remark 4 Both AdamRNN and RMSPropRNN can also be derived by letting $\boldsymbol{v}_t = -\boldsymbol{p}_t$ and $\widehat{\boldsymbol{W}} := \boldsymbol{U}^{-1}\boldsymbol{W}$ as in Remark 1. This parameterization yields the following formulation for AdamRNN

$$\begin{aligned}
\boldsymbol{v}_t &= \mu \boldsymbol{v}_{t-1} + s\widehat{\boldsymbol{W}}\boldsymbol{x}_t, \\
\boldsymbol{m}_t &= \beta \boldsymbol{m}_{t-1} + (1-\beta)(\widehat{\boldsymbol{W}}\boldsymbol{x}_t \odot \widehat{\boldsymbol{W}}\boldsymbol{x}_t), \\
\boldsymbol{h}_t &= \sigma\left(\boldsymbol{U}\boldsymbol{h}_{t-1} + \frac{\boldsymbol{U}\boldsymbol{v}_t}{\sqrt{\boldsymbol{m}_t + \epsilon}}\right).
\end{aligned}$$

Here, we simply need to learn $\widehat{\boldsymbol{W}}$ and $\boldsymbol{U}$ without any relaxation. In contrast, we relaxed $\boldsymbol{U}^{-1}$ to an identity matrix in (17). Our experiments suggest that both parameterizations yield similar results.

2.5 Integrating momentum into LSTMs

Similar to integrating momentum into RNNs, we can integrate momentum into LSTMs by handling the input gate, cell input, and output gate in the same way as outlined in Sects. 2.2 and 2.4.

2.6 Experimental results

In this subsection, we evaluate the effectiveness of our momentum approach in designing RNNs in terms of convergence speed and accuracy. We compare the performance of the MomentumLSTM with the baseline LSTM [37] in the following tasks: (1) the object classification task on pixel-permuted MNIST [47], (2) the speech prediction task on the TIMIT dataset [3,34,35,58,113], (3) the celebrated copying and adding tasks [3,37], and (4) the language modeling task on the Penn TreeBank (PTB) dataset [59]. These four tasks are among standard benchmarks to measure the performance of RNNs and their ability to handle long-term dependencies. Also, these tasks cover different data modalities—image, speech, and text data—as well as a variety of model sizes, ranging from thousands to millions of parameters with one (MNIST and TIMIT tasks) or multiple (PTB task) recurrent cells in concatenation. Our experimental results confirm that MomentumLSTM converges faster and yields better test accuracy than the baseline LSTM across tasks and settings. We also discuss the AdamLSTM, RMSPropLSTM, and scheduled restart LSTM (SRLSTM) and show their advantage over MomentumLSTM in specific tasks. All of our results are averaged over 5 runs with different seeds. For MNIST and TIMIT experiments, we use the baseline codebase provided by [10]. For PTB experiments, we use the baseline codebase provided by [81].

2.6.1 Pixel-by-Pixel MNIST

In this task, we classify image samples of hand-written digits from the MNIST dataset [50] into one of the ten classes. Following the implementation of [47], we flatten the image of original size 28×28 pixels and feed it into the model as a sequence of length 784. In the unpermuted task (MNIST), the sequence of pixels is processed row by row. In the permuted task (PMNIST), a fixed permutation is selected at the beginning of the experiments and then applied to both training and test sequences. We summarize the results in Table 1. Our experiments show that *MomentumLSTM achieves better test accuracy than the baseline LSTM in both MNIST and PMNIST digit classification tasks* using different

Table 1 Best test accuracy at the MNIST and PMNIST tasks (%)

Model	n	# params (K)	MNIST	PMNIST
LSTM	128	≈ 68	98.70[34],97.30 [102]	92.00 [34],92.62 [102]
LSTM	256	≈ 270	98.90 [34], 98.50 [113]	92.29 [34], 92.10 [113]
MomentumLSTM	128	≈ 68	9.04 ± 0.049	3.40 ± 0.259
MomentumLSTM	256	≈ 270	9.08 ± 0.059	4.72 ± 0.169
AdamLSTM	256	≈ 270	99.09 ± 0.03	95.05 ± 0.37
RMSPropLSTM	256	≈ 270	9.15 ± 0.069	5.38 ± 0.199
SRLSTM	256	≈ 270	99.01 ± 0.07	93.82 ± 1.85

We use the baseline results reported in [34,102,113]. Our proposed models outperform the baseline LSTM. Among the models using $N = 256$ hidden units, RMSPropLSTM yields the best results

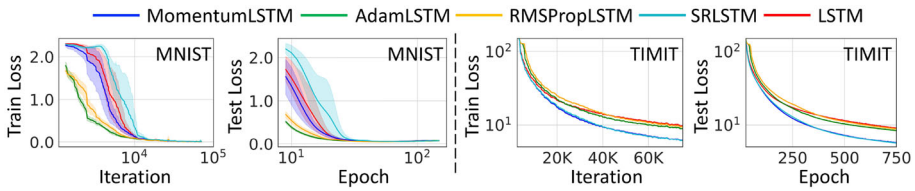

Fig. 3 Train and test loss of MomentumLSTM, AdamLSTM, RMSPropLSTM, SRLSTM, and LSTM for MNIST (left two panels) and TIMIT (right two panels) tasks. MomentumLSTM converges faster than LSTM in both tasks. For MNIST, AdamLSTM and RMSPropLSTM converge fastest. For TIMIT, MomentumLSTM and SRLSTM converge fastest

numbers of hidden units (i.e., $N = 128, 256$). Especially, the improvement is significant on the PMNIST task, which is designed to test the performance of RNNs in the context of long-term memory. Furthermore, we notice that *MomentumLSTM converges faster than LSTM* in all settings. Figure 3 corroborates this observation when using $N = 256$ hidden units.

2.6.2 TIMIT speech dataset

We study how MomentumLSTM performs on audio data with speech prediction experiments on the TIMIT speech dataset [28], which is a collection of real-world speech recordings. As first proposed by [113], the recordings 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. We use the standard train/validation/test separation in [11,52,113], thereby having 3640 utterances for the training set with a validation set of size 192 and a test set of size 400.

The results for this TIMIT speech prediction are shown in Table 2. Results are reported on the test set using the model parameters that yield the best validation loss. Again, we see the advantage of MomentumLSTM over the baseline LSTM. In particular, MomentumLSTM yields much better prediction accuracy and faster convergence speed compared to LSTM. Figure 3 shows the convergence of MomentumLSTM vs. LSTM when using $N = 158$ hidden units.

2.6.3 Copying and adding tasks

Two other important tasks for measuring the ability of a model to learn long-term dependency are the copying and adding tasks [3,37]. In both copying and adding tasks, avoiding vanishing/exploding gradients becomes more relevant when the input sequence length

Table 2 Test and validation MSEs at the end of the epoch with the lowest validation MSE for the TIMIT task

Model	n	# params (K)	Val. MSE	Test MSE
LSTM	84	≈ 83	14.87 ± 0.15	14.94 ± 0.15
LSTM	120	≈ 135	11.77 ± 0.14	11.83 ± 0.12
LSTM	158	≈ 200	9.33 ± 0.14	9.37 ± 0.14
MomentumLSTM	84	≈ 83	0.90 ± 0.191	0.98 ± 0.181
MomentumLSTM	120	≈ 135	.00 ± 0.308	.04 ± 0.308
MomentumLSTM	158	≈ 200	.86 ± 0.145	.87 ± 0.155
AdamLSTM	158	≈ 200	8.66 ± 0.15	8.69 ± 0.14
RMSPropLSTM	158	≈ 200	9.13 ± 0.33	9.17 ± 0.33
SRLSTM	158	≈ 200	.81 ± 0.105	.83 ± 0.105

All of our proposed models outperform the baseline LSTM. Among models using $N = 158$ hidden units, SRLSTM performs the best

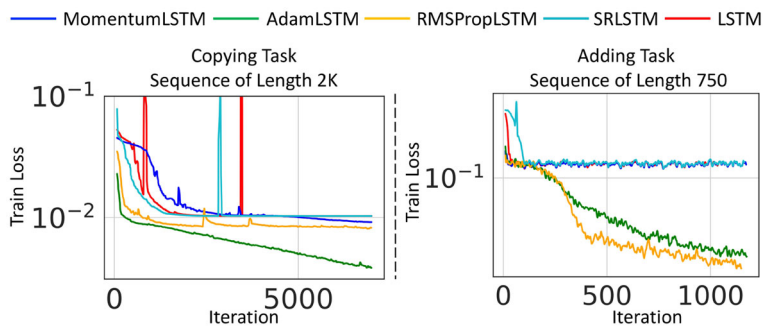

Fig. 4 Train loss vs. iteration for (left) copying task with sequence length 2K and (right) adding task with sequence length 750. AdamLSTM and RMSPropLSTM converge faster and to better final losses than other models

increases. We compare the performance of MomentumLSTM over LSTM on these tasks. We also examine the performance of AdamLSTM, RMSPropLSTM, and SRLSTM on the same tasks. We summarize our results in Fig. 4. In copying task for sequences of length 2K, MomentumLSTM obtains slightly better final training loss than the baseline LSTM (0.009 vs. 0.01). In adding task for sequence of length 750, both models achieve similar training loss of 0.162. However, AdamLSTM and RMSPropLSTM significantly outperform the baseline LSTM.

2.6.4 Word-level Penn TreeBank

To study the advantage of MomentumLSTM over LSTM on text data, we perform language modeling on a preprocessed version of the PTB dataset [59], which has been a standard benchmark for evaluating language models. Unlike the baselines used in the (P)MNIST and TIMIT experiments which contain one LSTM cell, in this PTB experiment, we use a three-layer LSTM model, which contains three concatenated LSTM cells, as the baseline. The size of this model in terms of the number of parameters is also much larger than those in the (P)MNIST and TIMIT experiments. Table 3 shows the test and validation perplexity (PPL) using the model parameters that yield the best validation loss. Again, MomentumLSTM achieves better perplexities and converges faster than the baseline LSTM (see Fig. 5).

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 (M)	Val. PPL	Test PPL
lstm	≈ 24	61.96 ± 0.83	59.71 ± 0.99
MomentumLSTM	≈ 24	0.71 ± 0.246	8.62 ± 0.225
SRLSTM	≈ 24	61.12 ± 0.68	58.83 ± 0.62

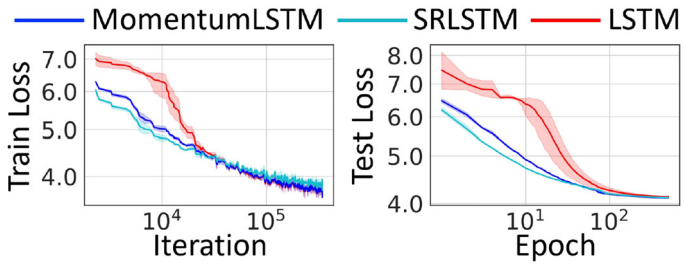

Fig. 5 Train (left) and test loss (right) of MomentumLSTM (blue), SRLSTM (cyan), and LSTM (red) for the Penn Treebank language modeling tasks at word level

2.6.5 NAG- and Adam-principled RNNs

Finally, we evaluate AdamLSTM, RMSPropLSTM, and SRLSTM on all tasks. For (P)MNIST and TIMIT tasks, we summarize the test accuracy of the trained models in Tables 1 and 2 and provide the plots of train and test losses in Fig. 3. We observe that though AdamLSTM and RMSPropLSTM work better than the MomentumLSTM at (P)MNIST task, they yield worse results at the TIMIT task. Interestingly, SRLSTM shows an opposite behavior—better than MomentumLSTM at TIMIT task but worse at (P)MNIST task. For the copying and adding tasks, Fig. 4 shows that AdamLSTM and RMSPropLSTM converge faster and to better final training loss than other models in both tasks. Finally, for the PTB task, both MomentumLSTM and SRLSTM outperform the baseline LSTM (see Fig. 5 and Table 3). However, in this task, AdamLSTM and RMSPropLSTM yield slightly worse performance than the baseline LSTM. In particular, test PPL for AdamLSTM and RMSPropLSTM are 61.11 ± 0.31 and 64.53 ± 0.20, respectively, which are higher than the test PPL for LSTM (59.71 ± 0.99). We observe that there is no model that win in all tasks. This is somewhat expected, given the connection between our model and its analogy to optimization algorithm. An optimizer needs to be chosen for each particular task, and so is for our MomentumRNN. All of our models outperform the baseline LSTM.

2.7 Summary of our contributions and limitations

We present a systematic approach to integrating momentum into RNNs and LSTMs, resulting in more expressive neural network architectures for sequential learning. In particular, the new architecture can mitigate vanishing gradient issues and results in effective learning of long-term dependencies. One particular bottleneck is that the current MomentumRNNs require calibrating momentum-related hyperparameters. One approach to avoid this is to develop adaptive schemes for these hyperparameters; see Sect. 4 for an illustration.

3 Neural ODEs

In this section, we derive the continuous limit of heavy-ball momentum and then present a new class of neural ODEs, named heavy-ball neural ODEs (HBNODEs), which have two properties that imply practical advantages over NODEs: (1) The adjoint state of an HBNODE also satisfies an HBNODE, accelerating both forward and backward ODE solvers, thus significantly reducing the NFEs and improving the utility of trained models. (2) The spectrum of HBNODEs is well structured, enabling effective learning of long-term dependencies from complex sequential data. We verify the advantages of HBNODEs over NODEs on benchmark tasks, including image classification, learning complex dynamics, and sequential modeling. Our method requires remarkably fewer forward and backward NFEs, is more accurate, and learns long-term dependencies more effectively than the other ODE-based neural network models. Part of the results in this section has been accepted for publication at NeurIPS 2021 [114]. We organize this section as follows: In Sect. 3.1, we briefly review the neural ODE models and their bottlenecks and some recent advances in neural ODEs. We present heavy-ball neural ODEs and their adjoint equations in Sect. 3.2, and we show that heavy-ball neural ODEs help to learn long-term dependencies in Sect. 3.3. We contrast the empirical performance of heavy-ball neural ODEs with other benchmark models in Sect. 3.4.

3.1 Recap on neural ODEs

Neural ODEs (NODEs) are a family of continuous-depth machine learning models whose forward and backward propagations rely on solving an ODE and its adjoint equation [14]. NODEs model the dynamics of hidden features $\boldsymbol{h}(t) \in \mathbb{R}^N$ using an ODE, which is parametrized by a neural network $f(\boldsymbol{h}(t), t, \theta) \in \mathbb{R}^N$ with learnable parameters θ, i.e.,

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta). \tag{18}$$

Starting from the input $\boldsymbol{h}(t_0)$, NODEs obtain the output $\boldsymbol{h}(T)$ by solving (18) for $t_0 \le t \le T$ with the initial value $\boldsymbol{h}(t_0)$, using a black-box numerical ODE solver. The NFEs that the black-box ODE solver requires in a single forward pass are an analogue for the continuous-depth models [14] to the depth of networks in ResNets [32]. The loss between $\boldsymbol{h}(T)$ and the ground truth is denoted by $\mathcal{L}(\boldsymbol{h}(T))$; we update parameters θ using the following gradient [72]

$$\frac{d\mathcal{L}(\boldsymbol{h}(T))}{d\theta} = \int_{t_0}^{T} \boldsymbol{a}(t) \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta} dt, \tag{19}$$

where $\boldsymbol{a}(t) := \partial \mathcal{L} / \partial \boldsymbol{h}(t)$ satisfies the following adjoint equation

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t) \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}}. \tag{20}$$

NODEs are flexible in learning from irregularly sampled sequential data and particularly suitable for learning complex dynamical systems [14,24,42,65,80,117], which can be trained by efficient algorithms [20,74,118]. The drawback of NODEs is also prominent. In many ML tasks, NODEs require very high NFEs in both training and inference, especially in high accuracy settings where a lower tolerance is needed. The NFEs increase rapidly with training; high NFEs reduce computational speed and accuracy of NODEs and can

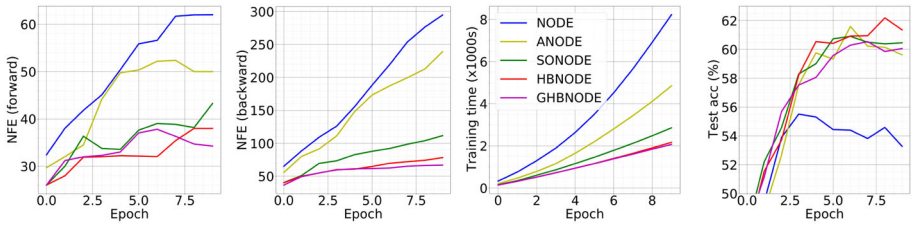

Fig. 6 Contrasting NODE, ANODE, SONODE, HBNODE, and GHBNODE for CIFAR10 classification in NFEs, training time, and test accuracy. (Tolerance: 10^{-5}, see Sect. 3.4.2 for experimental details)

lead to blow-ups in the worst-case scenario [25,30,57,65]. As an illustration, we train NODEs for CIFAR10 classification using the same model and experimental settings as in [25], except using a tolerance of 10^{-5}; Fig. 6 shows both forward and backward NFEs and the training time of different ODE-based models; we see that NFEs and computational times increase very rapidly for NODE, ANODE [25], and SONODE [65]. More results on the large NFE and degrading utility issues for different benchmark experiments are available in Sect. 3.4. Another issue is that NODEs often fail to effectively learn long-term dependencies in sequential data [48], discussed in Sect. 3.3.

3.2 Heavy-ball neural ODEs

3.2.1 Heavy-ball ODE

The continuous limit of the heavy-ball method has been widely studied, see, e.g., [4,111]. For the ease of reading and completeness, we derive the HBODE from the heavy-ball momentum method. For any fixed step size s, let $\boldsymbol{m}^k := (\boldsymbol{x}^{k+1} - \boldsymbol{x}^k)/\sqrt{s}$, and let $\beta := 1 - \gamma\sqrt{s}$, where $\gamma \geq 0$ is another hyperparameter. Then we can rewrite (1) as

$$\boldsymbol{m}^{k+1} = (1 - \gamma\sqrt{s})\boldsymbol{m}^k - \sqrt{s}\nabla F(\boldsymbol{x}^k);\ \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \sqrt{s}\boldsymbol{m}^{k+1}. \tag{21}$$

Let $s \to 0$ in (21); we obtain the following system of first-order ODEs,

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{m}(t);\ \frac{d\boldsymbol{m}(t)}{dt} = -\gamma\boldsymbol{m}(t) - \nabla F(\boldsymbol{x}(t)). \tag{22}$$

This can be further rewritten as a second-order heavy-ball ODE (HBODE), which also models a damped oscillator,

$$\frac{d^2\boldsymbol{x}(t)}{dt^2} + \gamma\frac{d\boldsymbol{x}(t)}{dt} = -\nabla F(\boldsymbol{x}(t)). \tag{23}$$

3.2.2 Heavy-ball neural ODEs

Similar to NODE, we parameterize $-\nabla F$ in (23) using a neural network $f(\boldsymbol{h}(t), t, \theta)$, resulting in the following HBNODE with initial position $\boldsymbol{h}(t_0)$ and momentum $\boldsymbol{m}(t_0) := d\boldsymbol{h}/dt(t_0)$,

$$\frac{d^2\boldsymbol{h}(t)}{dt^2} + \gamma\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \tag{24}$$

where $\gamma \geq 0$ is the damping parameter, which can be set as a tunable or a learnable hyperparameter with positivity constraint. In the trainable case, we use $\gamma = \epsilon \cdot \text{sigmoid}(\omega)$

for a trainable $\omega \in \mathbb{R}$ and a fixed tunable upper bound ϵ (we set $\epsilon = 1$ below). According to (22), HBNODE (24) is equivalent to

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{m}(t); \quad \frac{d\boldsymbol{m}(t)}{dt} = -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta). \tag{25}$$

Equation (24) (or equivalently, the system (25)) defines the forward ODE for the HBNODE, and we can use either the first-order (Prop. 2) or the second-order (Prop. 1) adjoint sensitivity method to update the parameter θ [65].

Proposition 1 (Adjoint equation for HBNODE) *The adjoint state* $\boldsymbol{a}(t) := \partial \mathcal{L}/\partial \boldsymbol{h}(t)$ *for the HBNODE* (24) *satisfies the following HBODE with the same damping parameter* γ *as that in* (24)*,*

$$\frac{d^2\boldsymbol{a}(t)}{dt^2} - \gamma \frac{d\boldsymbol{a}(t)}{dt} = \boldsymbol{a}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta). \tag{26}$$

Proof Consider the following coupled first-order ODE system

$$\frac{\partial}{\partial t}\begin{bmatrix}\boldsymbol{h}\\ \boldsymbol{v}\end{bmatrix} = \begin{bmatrix}\boldsymbol{v}\\ f(\boldsymbol{h}(t), \boldsymbol{v}(t), t, \theta)\end{bmatrix}, \quad \begin{bmatrix}\boldsymbol{h}\\ \boldsymbol{v}\end{bmatrix}(t_0) = \begin{bmatrix}\boldsymbol{h}_{t_0}\\ \boldsymbol{v}_{t_0}\end{bmatrix}. \tag{27}$$

Denote $\boldsymbol{z} = \begin{bmatrix}\boldsymbol{h}\\ \boldsymbol{v}\end{bmatrix}$ and final state as

$$\begin{bmatrix}\boldsymbol{h}(T)\\ \boldsymbol{v}(T)\end{bmatrix} = \begin{bmatrix}\boldsymbol{h}_T\\ \boldsymbol{v}_T\end{bmatrix} = \boldsymbol{z}_T. \tag{28}$$

Then, the adjoint equation is given by

$$\frac{\partial \boldsymbol{A}(t)}{\partial t} = -\boldsymbol{A}(t)\begin{bmatrix}\boldsymbol{0} & \boldsymbol{I}\\ \frac{\partial f}{\partial \boldsymbol{h}} & \frac{\partial f}{\partial \boldsymbol{v}}\end{bmatrix}, \quad \boldsymbol{A}(T) = -\boldsymbol{I}, \quad \boldsymbol{a}(t) = -\frac{d\mathcal{L}}{d\boldsymbol{z}_T}\boldsymbol{A}(t). \tag{29}$$

By rewriting $\boldsymbol{A} = \begin{bmatrix}\boldsymbol{A}_{\boldsymbol{h}} & \boldsymbol{A}_{\boldsymbol{v}}\end{bmatrix}$, we have the following differential equations

$$\frac{\partial \boldsymbol{A}_{\boldsymbol{h}}(t)}{\partial t} = -\boldsymbol{A}_{\boldsymbol{v}}(t)\frac{\partial f}{\partial \boldsymbol{h}}, \quad \frac{\partial \boldsymbol{A}_{\boldsymbol{v}}(t)}{\partial t} = -\boldsymbol{A}_{\boldsymbol{h}}(t) - \boldsymbol{A}_{\boldsymbol{v}}(t)\frac{\partial f}{\partial \boldsymbol{v}}, \tag{30}$$

with initial conditions

$$\boldsymbol{A}_{\boldsymbol{h}}(T) = -\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{0}\end{bmatrix}, \quad \boldsymbol{A}_{\boldsymbol{v}}(T) = -\begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{I}\end{bmatrix}, \tag{31}$$

and adjoint states

$$\boldsymbol{a}_{\boldsymbol{h}}(t) = \frac{d\mathcal{L}}{d\boldsymbol{z}_T}\boldsymbol{A}_{\boldsymbol{h}}(t), \quad \boldsymbol{a}_{\boldsymbol{v}}(t) = \frac{d\mathcal{L}}{d\boldsymbol{z}_T}\boldsymbol{A}_{\boldsymbol{v}}(t). \tag{32}$$

The gradient equations become

$$\frac{d\mathcal{L}}{d\theta} = \int_{t_0}^{T} \boldsymbol{a}\begin{bmatrix}\boldsymbol{0}\\ \frac{\partial f}{\partial \theta}\end{bmatrix} dt = \int_{t_0}^{T} \boldsymbol{a}_{\boldsymbol{v}}\frac{\partial f}{\partial \theta} dt, \quad \frac{d\mathcal{L}}{d\boldsymbol{h}_{t_0}} = \boldsymbol{a}_{\boldsymbol{h}}(t_0), \quad \frac{d\mathcal{L}}{d\boldsymbol{v}_{t_0}} = \boldsymbol{a}_{\boldsymbol{v}}(t_0). \tag{33}$$

Note $\boldsymbol{h}_{t_0}$ is fixed, and thus, $\boldsymbol{a}_{\boldsymbol{h}}$ disappears in gradient computation. Therefore, we are only interested in $\boldsymbol{a}_{\boldsymbol{v}}$. Thus, the adjoint $\boldsymbol{A}_{\boldsymbol{v}}$ satisfies the following second-order ODE

$$\frac{\partial^2 \boldsymbol{A}_{\boldsymbol{v}}(t)}{\partial t^2} = \boldsymbol{A}_{\boldsymbol{v}}(t)\frac{\partial f}{\partial \boldsymbol{h}} - \frac{\partial(\boldsymbol{A}_{\boldsymbol{v}}(t)\frac{\partial f}{\partial \boldsymbol{v}})}{\partial t}, \tag{34}$$

and thus

$$\frac{\partial^2 \boldsymbol{a}_{\boldsymbol{v}}(t)}{\partial t^2} = \boldsymbol{a}_{\boldsymbol{v}}(t)\frac{\partial f}{\partial \boldsymbol{h}} - \frac{\partial(\boldsymbol{a}_{\boldsymbol{v}}(t)\frac{\partial f}{\partial \boldsymbol{v}})}{\partial t}, \tag{35}$$

with initial conditions

$$\boldsymbol{a}_{\boldsymbol{v}}(T) = -\frac{d\mathcal{L}}{d\boldsymbol{z}}\boldsymbol{A}_{\boldsymbol{v}}(T) = \frac{d\mathcal{L}}{d\boldsymbol{v}_T}, \quad \frac{\partial \boldsymbol{a}_{\boldsymbol{v}}(T)}{\partial t} = -\frac{d\mathcal{L}}{d\boldsymbol{h}_T} - \boldsymbol{a}_{\boldsymbol{v}}(T)\frac{\partial f}{\partial \boldsymbol{v}}(T). \tag{36}$$

As HBNODE takes the form

$$\frac{d^2\boldsymbol{h}(t)}{dt^2} + \gamma\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \tag{37}$$

which can also be viewed as a SONODE. By applying the adjoint equation (35), we arrive at

$$\frac{\partial^2 \boldsymbol{a}(t)}{\partial t^2} = \boldsymbol{a}(t)\frac{\partial f}{\partial \boldsymbol{h}} + \gamma\frac{\partial \boldsymbol{a}(t)}{\partial t}. \tag{38}$$

As HBNODE only carries its state $\boldsymbol{h}$ to the loss $\mathcal{L}$, we have $\frac{d\mathcal{L}}{d\boldsymbol{v}_T} = 0$, and thus, the initial conditions in equation (36) become

$$\boldsymbol{a}(T) = \boldsymbol{0}, \quad \frac{\partial \boldsymbol{a}(T)}{\partial t} = -\frac{d\mathcal{L}}{d\boldsymbol{h}_T}, \tag{39}$$

which concludes the proof of Proposition 1. □

Remark 5 Note that we solve the adjoint equation (26) from time $t = T$ to $t = t_0$ in the backward propagation. By letting $\tau = T - t$ and $\boldsymbol{b}(\tau) = \boldsymbol{a}(T - \tau)$, we can rewrite (26) as follows,

$$\frac{d^2\boldsymbol{b}(\tau)}{d\tau^2} + \gamma\frac{d\boldsymbol{b}(\tau)}{d\tau} = \boldsymbol{b}(\tau)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(T-\tau), T-\tau, \theta). \tag{40}$$

Therefore, the adjoint of the HBNODE is also a HBNODE and they have the same damping parameter.

Proposition 2 (Adjoint equations for the first-order HBNODE system) *The adjoint states* $\boldsymbol{a}_{\boldsymbol{h}}(t) := \partial\mathcal{L}/\partial\boldsymbol{h}(t)$ *and* $\boldsymbol{a}_{\boldsymbol{m}}(t) := \partial\mathcal{L}/\partial\boldsymbol{m}(t)$ *for the first-order HBNODE system* (25) *satisfy*

$$\frac{d\boldsymbol{a}_{\boldsymbol{h}}(t)}{dt} = -\boldsymbol{a}_{\boldsymbol{m}}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta); \quad \frac{d\boldsymbol{a}_{\boldsymbol{m}}(t)}{dt} = -\boldsymbol{a}_{\boldsymbol{h}}(t) + \gamma\boldsymbol{a}_{\boldsymbol{m}}(t). \tag{41}$$

Proof The coupled form of HBNODE is a coupled first-order ODE system of the form

$$\frac{\partial}{\partial t}\begin{bmatrix}\boldsymbol{h}\\ \boldsymbol{m}\end{bmatrix}=\begin{bmatrix}\boldsymbol{m}\\ -\gamma\boldsymbol{m}+f(\boldsymbol{h}(t),t,\theta)\end{bmatrix},\quad \begin{bmatrix}\boldsymbol{h}\\ \boldsymbol{m}\end{bmatrix}(t_0)=\begin{bmatrix}\boldsymbol{h}_{t_0}\\ \boldsymbol{m}_{t_0}\end{bmatrix}. \tag{42}$$

Denote the final state as

$$\begin{bmatrix}\boldsymbol{h}(T)\\ \boldsymbol{m}(T)\end{bmatrix}=\begin{bmatrix}\boldsymbol{h}_T\\ \boldsymbol{m}_T\end{bmatrix}=z. \tag{43}$$

Using the conclusions from the proof of Proposition 1, we have the adjoint equation

$$\frac{\partial \boldsymbol{A}(t)}{\partial t}=-\boldsymbol{A}(t)\begin{bmatrix}\boldsymbol{0} & \boldsymbol{I}\\ \frac{\partial f}{\partial \boldsymbol{h}} & -\gamma\boldsymbol{I}\end{bmatrix},\quad \boldsymbol{A}(T)=-\boldsymbol{I},\quad \boldsymbol{a}(t)=-\frac{d\mathcal{L}}{d\boldsymbol{z}}\boldsymbol{A}(t). \tag{44}$$

Let $\begin{bmatrix}\boldsymbol{a_h} & \boldsymbol{a_m}\end{bmatrix}=\boldsymbol{a}$, by linearity we have

$$\frac{\partial\begin{bmatrix}\boldsymbol{a_h} & \boldsymbol{a_m}\end{bmatrix}}{\partial t}=-\begin{bmatrix}\boldsymbol{a_h} & \boldsymbol{a_m}\end{bmatrix}\begin{bmatrix}\boldsymbol{0} & \boldsymbol{I}\\ \frac{\partial f}{\partial \boldsymbol{h}} & -\gamma\boldsymbol{I}\end{bmatrix},\quad \begin{bmatrix}\boldsymbol{a_h}(T) & \boldsymbol{a_m}(T)\end{bmatrix}=\begin{bmatrix}\frac{d\mathcal{L}}{d\boldsymbol{h}_T} & \frac{d\mathcal{L}}{d\boldsymbol{m}_T}\end{bmatrix}, \tag{45}$$

which gives us the initial conditions at $t=T$, and the simplified first-order ODE system

$$\frac{\partial \boldsymbol{a_h}}{\partial t}=-\boldsymbol{a_m}\frac{\partial f}{\partial \boldsymbol{h}},\quad \frac{\partial \boldsymbol{a_m}}{\partial t}=-\boldsymbol{a_h}+\gamma\boldsymbol{a_m}, \tag{46}$$

concluding the proof of Proposition 2. □

Remark 6 Let $\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)=d\boldsymbol{a_m}(t)/dt$, then $\boldsymbol{a_m}(t)$ and $\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)$ satisfies the following first-order heavy-ball ODE system

$$\frac{d\boldsymbol{a_m}(t)}{dt}=\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t);\quad \frac{d\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)}{dt}=\boldsymbol{a_m}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t),t,\theta)+\gamma\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t). \tag{47}$$

Note that we solve this system backward in time in back-propagation. Moreover, we have $\boldsymbol{a_h}(t)=\gamma\boldsymbol{a_m}(t)-\tilde{\boldsymbol{a}}_{\boldsymbol{m}}(t)$.

Similar to [65], we use the coupled first-order HBNODE system (25) and its adjoint first-order HBNODE system (41) for practical implementation, since the entangled representation permits faster computation [65] of the gradients of the coupled ODE systems.

3.2.3 Generalized heavy-ball neural ODEs

In this part, we propose a generalized version of HBNODE (GHBNODE), see (48), to mitigate the potential blow-up issue in training ODE-based models. We observe that $\boldsymbol{h}(t)$ of ANODEs [25], SONODEs [65], and HBNODEs (25) usually grows much faster than that of NODEs. The fast growth of $\boldsymbol{h}(t)$ can lead to finite-time blow-up. As an illustration, we compare the performance of NODE, ANODE, SONODE, HBNODE, and GHBNODE on the Silverbox task as in [65]. The goal of the task is to learn the voltage of an electronic circuit that resembles a Duffing oscillator, where the input voltage $V_1(t)$ is used to predict the output $V_2(t)$. Similar to the setting in [65], we first augment ANODE by 1 dimension with 0-augmentation and augment SONODE, HBNODE, and GHBNODE with a dense

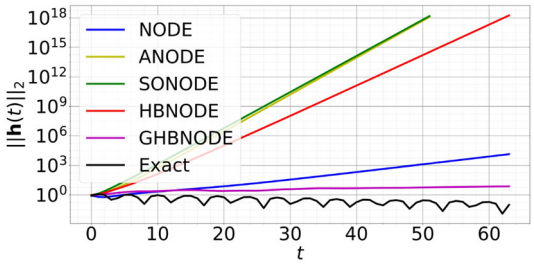

Fig. 7 Contrasting $\boldsymbol{h}(t)$ for different models. $\boldsymbol{h}(t)$ in ANODE, SONODE, and HBNODE grows much faster than that in NODE. GHBNODE controls the growth of $\boldsymbol{h}(t)$ effectively when t is large

network. We use a simple dense layer to parameterize f for all five models, with an extra input term for $V_1(t)$[4]. For both HBNODE and GHBNODE, we set the damping parameter γ to be sigmoid(-3). For GHBNODE (48) below, we set $\sigma(\cdot)$ to be the `hardtanh` function with bound $[-5, 5]$ and $\xi = \ln(2)$. As shown in Fig. 7, compared to the vanilla NODE, the ℓ_2 norm of $\boldsymbol{h}(t)$ grows much faster when a higher-order NODE is used, which leads to blow-up during training. Similar issues arise in the time-series experiments (see Sect. 3.4.4), where SONODE blows up during long-term integration in time, and HBNODE suffers from the same issue with some initialization.

To alleviate the problem above, we propose the following GHBNODE

$$\begin{aligned} \frac{d\boldsymbol{h}(t)}{dt} &= \sigma(\boldsymbol{m}(t)), \\ \frac{d\boldsymbol{m}(t)}{dt} &= -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta) - \xi \boldsymbol{h}(t), \end{aligned} \tag{48}$$

where $\sigma(\cdot)$ is a nonlinear activation, which is set as tanh in our experiments. The positive hyperparameters $\gamma, \xi > 0$ are tunable or learnable. In the trainable case, we let $\gamma = \epsilon \cdot \text{sigmoid}(\omega)$ as in HBNODE, and $\xi = \text{softplus}(\chi)$ to ensure that $\gamma, \xi \geq 0$. Here, we integrate two main ideas into the design of GHBNODE: (i) We incorporate the gating mechanism used in LSTM [37] and GRU [16], which can suppress the aggregation of $\boldsymbol{m}(t)$; (ii) following the idea of skip connection [33], we add the term $\xi \boldsymbol{h}(t)$ into the governing equation of $\boldsymbol{m}(t)$, which benefits training and generalization of GHBNODEs. Figure 7 shows that GHBNODE can indeed control the growth of $\boldsymbol{h}(t)$ effectively.

Proposition 3 (Adjoint equations for GHBNODEs) *The adjoint states* $\boldsymbol{a}_{\boldsymbol{h}}(t) := \partial \mathcal{L}/\partial \boldsymbol{h}(t)$, $\boldsymbol{a}_{\boldsymbol{m}}(t) := \partial \mathcal{L}/\partial \boldsymbol{m}(t)$ *for the GHBNODE* (48) *satisfy the following first-order ODE system*

$$\begin{aligned} \frac{\partial \boldsymbol{a}_{\boldsymbol{h}}(t)}{\partial t} &= -\boldsymbol{a}_{\boldsymbol{m}}(t)\Big(\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta) - \xi \boldsymbol{I}\Big), \\ \frac{\partial \boldsymbol{a}_{\boldsymbol{m}}(t)}{\partial t} &= -\boldsymbol{a}_{\boldsymbol{h}}(t)\sigma'(\boldsymbol{m}(t)) + \gamma \boldsymbol{a}_{\boldsymbol{m}}(t). \end{aligned} \tag{49}$$

Proof GHBNODE can be written as the following first-order ODE system

$$\frac{\partial}{\partial t}\begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{m} \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{m}) \\ -\gamma \boldsymbol{m} + f(\boldsymbol{h}(t), t, \theta) - \xi \boldsymbol{h}(t) \end{bmatrix}, \quad \begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{m} \end{bmatrix}(t_0) = \begin{bmatrix} \boldsymbol{h}_{t_0} \\ \boldsymbol{m}_{t_0} \end{bmatrix}. \tag{50}$$

[4]Here, we exclude an $\boldsymbol{h}^3$ term that appeared in the original Duffing oscillator model because including it would result in finite-time explosion.

Denote the final state as $\boldsymbol{z}_T := [\boldsymbol{h}_T \boldsymbol{m}_T]$. We have the adjoint equation

$$\frac{\partial \boldsymbol{A}(t)}{\partial t} = -\boldsymbol{A}(t)\begin{bmatrix} \boldsymbol{0} & \sigma'(\boldsymbol{m}) \\ \frac{\partial f}{\partial \boldsymbol{h}} - \xi \boldsymbol{I} & -\gamma \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{A}(T) = -\boldsymbol{I}, \quad \boldsymbol{a}(t) = -\frac{d\mathcal{L}}{d\boldsymbol{z}_T}\boldsymbol{A}(t). \tag{51}$$

Let $\begin{bmatrix} \boldsymbol{a}_h & \boldsymbol{a}_m \end{bmatrix} = \boldsymbol{a}$, by linearity we have

$$\begin{aligned} &\frac{\partial \begin{bmatrix} \boldsymbol{a}_h & \boldsymbol{a}_m \end{bmatrix}}{\partial t} = -\begin{bmatrix} \boldsymbol{a}_h & \boldsymbol{a}_m \end{bmatrix}\begin{bmatrix} \boldsymbol{0} & \sigma'(\boldsymbol{m}) \\ \frac{\partial f}{\partial \boldsymbol{h}} - \xi \boldsymbol{I} & -\gamma \boldsymbol{I} \end{bmatrix}, \\ &\begin{bmatrix} \boldsymbol{a}_h(T) & \boldsymbol{a}_m(T) \end{bmatrix} = \begin{bmatrix} \frac{d\mathcal{L}}{d\boldsymbol{h}_T} & \frac{d\mathcal{L}}{d\boldsymbol{m}_T} \end{bmatrix}, \end{aligned} \tag{52}$$

which gives us the initial conditions at $t = T$, and the simplified first-order ODE system

$$\frac{\partial \boldsymbol{a}_h}{\partial t} = -\boldsymbol{a}_m\Big(\frac{\partial f}{\partial \boldsymbol{h}} - \xi \boldsymbol{I}\Big), \quad \frac{\partial \boldsymbol{a}_m}{\partial t} = -\boldsymbol{a}_h \sigma'(\boldsymbol{m}) + \gamma \boldsymbol{a}_m, \tag{53}$$

concluding the proof of Proposition 3. □

Though the adjoint state of the GHBNODE (49) does not satisfy the exact heavy-ball ODE, based on our empirical study, it also significantly reduces the backward NFEs.

3.3 Learning long-term dependencies: vanishing gradient

As mentioned in Sect. 2, the vanishing gradient is the main bottleneck for training RNNs with long-term dependencies. As the continuous analogue of RNN, NODEs as well as their hybrid ODE-RNN models, may also suffer from vanishing in the adjoint state $\boldsymbol{a}(t) := \partial \mathcal{L}/\partial \boldsymbol{h}(t)$ [48]. When the vanishing gradient issue happens, $\boldsymbol{a}(t)$ goes to $\boldsymbol{0}$ quickly as $T - t$ increases, then $d\mathcal{L}/d\theta$ in (19) will be independent of these $\boldsymbol{a}(t)$. We have the following expressions for the adjoint states of the NODE and HBNODE:

– For NODE, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T}\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T}\exp\Big\{-\int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)ds\Big\}. \tag{54}$$

– For GHBNODE[5], from (41) we can derive

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t}\frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_t} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T}\frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T} \end{bmatrix}\begin{bmatrix} \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{m}_t} \\ \frac{\partial \boldsymbol{m}_T}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{m}_T}{\partial \boldsymbol{m}_t} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T} \end{bmatrix}\exp\Big\{\underbrace{-\int_T^t \begin{bmatrix} \boldsymbol{0} & \frac{\partial \sigma}{\partial \boldsymbol{m}} \\ \Big(\frac{\partial f}{\partial \boldsymbol{h}} - \xi \boldsymbol{I}\Big) & -\gamma \boldsymbol{I} \end{bmatrix} ds}_{:=\boldsymbol{M}}\Big\}. \tag{55}$$

Note that the matrix exponential is directly related to its eigenvalues. By Schur decomposition, there exists an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{U}$, where the diagonal entries of $\boldsymbol{U}$ are eigenvalues of $\boldsymbol{Q}$ ordered by their real parts, such that

$$-\boldsymbol{M} = \boldsymbol{Q}\boldsymbol{U}\boldsymbol{Q}^\top \Longrightarrow \exp\{-\boldsymbol{M}\} = \boldsymbol{Q}\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top. \tag{56}$$

[5] HBNODE can be seen as a special GHBNODE with $\xi = 0$ and σ be the identity map.

Let $\boldsymbol{v}^\top := \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\right] \boldsymbol{Q}$, then (55) can be rewritten as

$$\begin{aligned} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_t}\right] &= \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\right] \exp\{-\boldsymbol{M}\} \\ &= \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\right] \boldsymbol{Q} \exp\{\boldsymbol{U}\} \boldsymbol{Q}^\top = \boldsymbol{v}^\top \exp\{\boldsymbol{U}\} \boldsymbol{Q}^\top. \end{aligned} \tag{57}$$

Taking the ℓ_2 norm in (57) and dividing both sides by $\left\| \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\right] \right\|_2$, we have

$$\frac{\left\| \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_t}\right] \right\|_2}{\left\| \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\right] \right\|_2} = \frac{\left\| \boldsymbol{v}^\top \exp\{\boldsymbol{U}\} \boldsymbol{Q}^\top \right\|_2}{\left\| \boldsymbol{v}^\top \boldsymbol{Q}^\top \right\|_2} = \frac{\left\| \boldsymbol{v}^\top \exp\{\boldsymbol{U}\} \right\|_2}{\|\boldsymbol{v}\|_2} = \left\| \boldsymbol{e}^\top \exp\{\boldsymbol{U}\} \right\|_2, \tag{58}$$

i.e., $\left\| \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_t}\right] \right\|_2 = \left\| \boldsymbol{e}^\top \exp\{\boldsymbol{U}\} \right\|_2 \left\| \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\right] \right\|_2$ where $\boldsymbol{e} = \boldsymbol{v}/\|\boldsymbol{v}\|_2$.

Proposition 4 *The eigenvalues of $-\boldsymbol{M}$ can be paired so that the sum of each pair equals $(t - T)\gamma$.*

Proof Let $\boldsymbol{F} = \frac{1}{t-T} \int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta) ds - \xi \boldsymbol{I}$, $\boldsymbol{J} = \frac{1}{t-T} \int_T^t \frac{\partial \sigma}{\partial \boldsymbol{m}}(\boldsymbol{m}(s)) ds$, and $\boldsymbol{H} = \frac{1}{t-T} \boldsymbol{M}$; then, we have the following equation

$$\boldsymbol{H} = \frac{1}{t - T} \boldsymbol{M} = \begin{bmatrix} 0 & \boldsymbol{J} \\ \boldsymbol{F} & -\gamma \boldsymbol{I} \end{bmatrix}. \tag{59}$$

As $(\lambda + \gamma)\boldsymbol{I}$ commutes with any matrix $\boldsymbol{F}$, the characteristics polynomials of $\boldsymbol{H}$ and $\boldsymbol{JF}$ satisfy the relation

$$\begin{aligned} ch_{\boldsymbol{H}}(\lambda) &= \det(\lambda \boldsymbol{I} - \boldsymbol{H}) = \det \begin{bmatrix} \lambda \boldsymbol{I} & -\boldsymbol{J} \\ -\boldsymbol{F} & (\lambda + \gamma)\boldsymbol{I} \end{bmatrix} \\ &= det(\lambda(\lambda + \gamma)\boldsymbol{I} - \boldsymbol{JF}) = -ch_{\boldsymbol{JF}}(\lambda(\lambda + \gamma)). \end{aligned} \tag{60}$$

Since the characteristics polynomial of $\boldsymbol{JF}$ splits in the field $\mathbb{C}$ of complex numbers, i.e., $ch_{\boldsymbol{JF}}(x) = \prod_{i=1}^n (x - \lambda_{\boldsymbol{JF},i})$, we have

$$ch_{\boldsymbol{H}}(\lambda) = -ch_{\boldsymbol{JF}}(\lambda(\lambda + \gamma)) = -\prod_{i=1}^{n} (\lambda(\lambda + \gamma) - \lambda_{\boldsymbol{JF},i}). \tag{61}$$

Therefore, the eigenvalues of $\boldsymbol{H}$ appear in n pairs with each pair satisfying the quadratic equation

$$\lambda(\lambda + \gamma) - \lambda_{\boldsymbol{JF},i} = 0. \tag{62}$$

By Vieta's formulas, the sum of these pairs are all $-\gamma$. Therefore, the eigenvalues of $\boldsymbol{M}$ come in n pairs and the sum of each pair is $-(t - T)\gamma$, which finishes the proof of Proposition 4. □

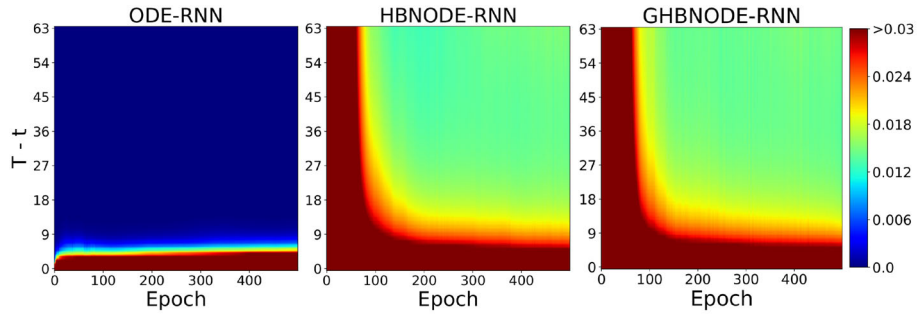

Fig. 8 Plot of the ℓ_2-norm of the adjoint states for ODE-RNN and (G)HBNODE-RNN back-propagated from the last time stamp. The adjoint state of ODE-RNN vanishes quickly when the gap between the final time T and intermediate time t becomes larger, while the adjoint states of (G)HBNODE-RNN decay much more slowly. This implies that (G)HBNODE-RNN is more effective in learning long-term dependency than ODE-RNN

For a given constant $a > 0$, we can group the upper triangular matrix $\exp\{\boldsymbol{U}\}$ as follows

$$\exp\{\boldsymbol{U}\} := \begin{bmatrix} \exp\{\boldsymbol{U}_L\} & \boldsymbol{P} \\ \boldsymbol{0} & \exp\{\boldsymbol{U}_V\} \end{bmatrix}, \tag{63}$$

where the diagonal of $\boldsymbol{U}_L$ ($\boldsymbol{U}_V$) contains eigenvalues of $-\boldsymbol{M}$ that are no less (greater) than $(t-T)a$. Then, we have $\|\boldsymbol{e}^\top \exp\{\boldsymbol{U}\}\|_2 \geq \|\boldsymbol{e}_L^\top \exp\{\boldsymbol{U}_L\}\|_2$ where the vector $\boldsymbol{e}_L$ denotes the first m columns of $\boldsymbol{e}$ with m be the number of columns of $\boldsymbol{U}_L$. By choosing $0 \leq \gamma \leq 2a$, for every pair of eigenvalues of $-\boldsymbol{M}$ there is at least one eigenvalue whose real part is no less than $(t-T)a$. Therefore, $\exp\{\boldsymbol{U}_L\}$ decays at a rate at most $(t-T)a$, and the dimension of $\boldsymbol{U}_L$ is at least $N \times N$. We avoid exploding gradients by clipping the ℓ_2 norm of the adjoint states similar to that used for training RNNs.

In contrast, all eigenvalues of the matrix $\int_T^t \partial f/\partial \boldsymbol{h} ds$ in (54) for NODE can be very positive or negative, resulting in exploding or vanishing gradients. As an illustration, we consider the benchmark Walker2D kinematic simulation task that requires learning long-term dependencies effectively [8,48]. We train ODE-RNN [80] and (G)HBNODE-RNN on this benchmark dataset, and the detailed experimental settings are provided in Sect. 3.4.4. Figure 8 plots $\|\partial \mathcal{L}/\partial \boldsymbol{h}_t\|_2$ for ODE-RNN and $\|[\partial \mathcal{L}/\partial \boldsymbol{h}_t \ \partial \mathcal{L}/\partial \boldsymbol{m}_t]\|_2$ for (G)HBNODE-RNN, showing that the adjoint state of ODE-RNN vanishes quickly, while that of (G)HBNODE-RNN does not vanish even when the gap between T and t is very large.

3.4 Experimental results

In this section, we compare the performance of the proposed HBNODE and GHBNODE with existing ODE-based models, including NODE [14], ANODE [25], and SONODE [65] on the benchmark point cloud separation, image classification, learning dynamical systems, and kinematic simulation. For all the experiments, we use Adam [44] as the benchmark optimization solver (the learning rate and batch size for each experiment are listed in Table 4). For HBNODE and GHBNODE, we set $\gamma = \text{sigmoid}(\theta)$, where θ is a trainable weight initialized as $\theta = -3$.

Table 4 The batch size and learning rate for different datasets

Dataset	Point Cloud	MNIST	CIFAR10	Plane Vibration	Walker2D
Batch Size	50	64	64	64	256
Learning Rate	0.01	0.001	0.001	0.0001	0.003

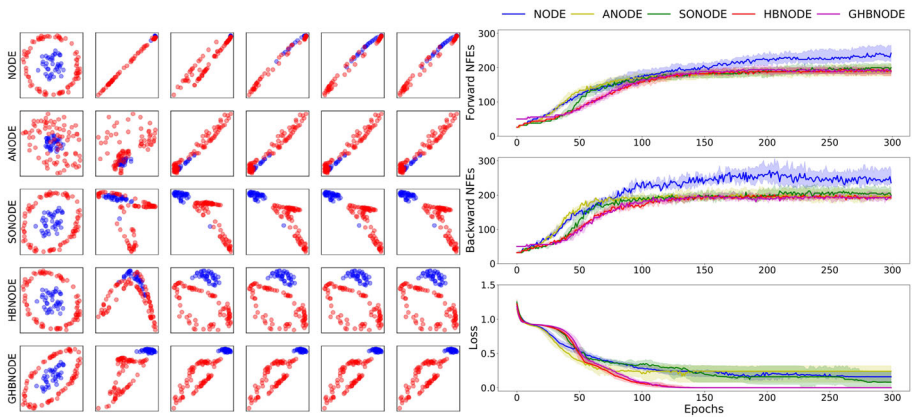

Fig. 9 Comparison between NODE, ANODE, SONODE, HBNODE, and GHBNODE for two-dimensional point cloud separation. HBNODE and GHBNODE converge better and require less NFEs in both forward and backward propagation than the other benchmark models

3.4.1 Point cloud separation

In this subsection, we consider the two-dimensional point cloud separation benchmark. A total of 120 points are sampled, in which 40 points are drawn uniformly from the circle $\|\boldsymbol{r}\| < 0.5$, and 80 points are drawn uniformly from the annulus $0.85 < \|\boldsymbol{r}\| < 1.0$. This experiment aims to learn effective features to classify these two point clouds. Following [25], we use a three-layer neural network to parameterize the right-hand side of each ODE-based model, integrate the ODE-based model from $t_0 = 0$ to $T = 1$, and pass the integration results to a dense layer to generate the classification results. We set the size of hidden layers so that the models have similar sizes, and the number of parameters of NODE, ANODE, SONODE, HBNODE, and GHBNODE is 525, 567, 528, 568, and 568, respectively. To avoid the effects of numerical error of the black-box ODE solver, we set tolerance of ODE solver to be 10^{-7}. Figure 9 plots a randomly selected evolution of the point cloud separation for each model; we also compare the forward and backward NFEs and the training loss of these models (100 independent runs). HBNODE and GHBNODE improve training as the training loss consistently goes to zero over different runs, while ANODE and SONODE often get stuck at local minima, and NODE cannot separate the point cloud since it preserves the topology [25].

3.4.2 Image classification

We compare the performance of HBNODE and GHBNODE with the existing ODE-based models on MNIST and CIFAR10 classification tasks using the same setting as in [25]. We parameterize $f(\boldsymbol{h}(t), t, \theta)$ using a 3-layer convolutional network for each ODE-based model, and the total number of parameters for each model is listed in Table 5. For a given input image of the size $c \times h \times w$, we first augment the number of channel from c to

Table 5 The number of parameters for each models for image classification

Model	NODE	ANODE	SONODE	HBNODE	GHBNODE
#Params (MNIST)	85,315	85,462	86,179	85,931	85,235
#Params (CIFAR10)	173,611	172,452	171,635	172,916	172,916

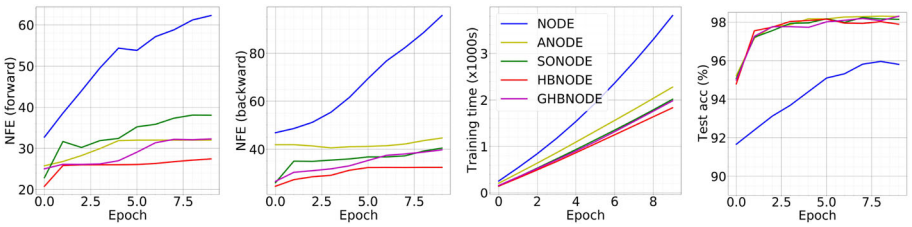

Fig. 10 Contrasting NODE [14], ANODE [25], SONODE [65], HBNODE, and GHBNODE for MNIST classification in NFE, training time, and test accuracy. (Tolerance: 10^{-5})

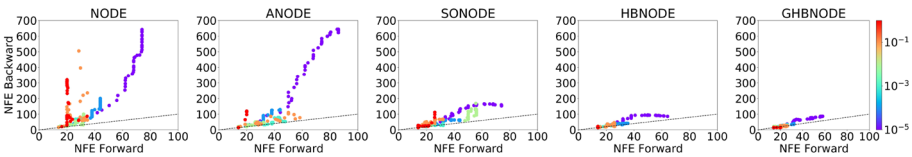

Fig. 11 NFE vs. tolerance (shown in the colorbar) for training ODE-based models for CIFAR10 classification. Both forward and backward NFEs of HBNODE and GHBNODE grow much more slowly than that of NODE, ANODE, and SONODE; especially the backward NFEs. As the tolerance decreases, the advantage of HBNODE and GHBNODE in reducing NFEs becomes more significant

$c + p$ with the augmentation dimension p dependent on each method[6]. Moreover, for SONODE, HBNODE and GHBNODE, we further include velocity or momentum with the same shape as the augmented state.

NFEs. As shown in Figs. 6 and 10, the NFEs grow rapidly with training of the NODE, resulting in an increasingly complex model with reduced performance and the possibility of blow-up. Input augmentation has been verified to effectively reduce the NFEs, as both ANODE and SONODE require fewer forward NFEs than NODE for the MNIST and CIFAR10 classification. However, input augmentation is less effective in controlling their backward NFEs. HBNODE and GHBNODE require much fewer NFEs than the existing benchmarks, especially for backward NFEs. In practice, reducing NFEs implies reducing both training and inference time, as shown in Figs. 6 and 10.

Accuracy. We also compare the accuracy of different ODE-based models for MNIST and CIFAR10 classification. As shown in Figs. 6 and 10, HBNODE and GHBNODE have slightly better classification accuracy than the other three models; this resonates with the fact that less NFEs lead to simpler models which generalize better [25, 65].

NFEs vs. tolerance. We further study the NFEs for different ODE-based models under different tolerances of the ODE solver using the same approach as in [14]. Figure 11 depicts the forward and backward NFEs for different models under different tolerances. We see that (i) both forward and backward NFEs grow quickly when tolerance is decreased,

[6] We set $p = 0, 5, 4, 4, 5/0, 10, 9, 9, 9$ on MNIST/CIFAR10 for NODE, ANODE, SONODE, HBNODE, and GHBNODE, respectively.

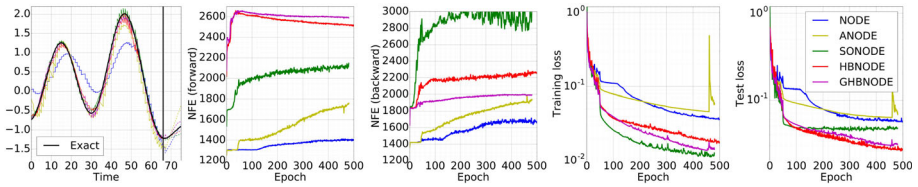

Fig. 12 Contrasting ODE-RNN, ANODE-RNN, SONODE-RNN, HBNODE-RNN, and GHBNODE-RNN for learning a vibrational dynamical system. Left most: The learned curves of each model vs. the ground truth (Time: <66 for training, 66–75 for testing)

and HBNODE and GHBNODE require much fewer NFEs than other models; (ii) under different tolerances, the backward NFEs of NODE, ANODE, and SONODE are much larger than the forward NFEs, and the difference becomes larger when the tolerance decreases. In contrast, the forward and backward NFEs of HBNODE and GHBNODE scale almost linearly with each other. This reflects that the advantage in NFEs of (G)HBNODE over the benchmarks become more significant when a smaller tolerance is used.

3.4.3 Learning dynamical systems from irregularly sampled time series

In this subsection, we learn dynamical systems from experimental measurements. In particular, we use the ODE-RNN framework [14,80], with the recognition model being set to different ODE-based models, to study the vibration of an airplane dataset [64]. The dataset was acquired, from time 0 to 73627, by attaching a shaker underneath the right wing to provide input signals, and 5 attributes are recorded per time stamp; these attributes include voltage of input signal, force applied to aircraft, and acceleration at 3 different spots of the airplane. We randomly take out 10% of the data to make the time series irregularly sampled. We use the first 50% of data as our train set, the next 25% as validation set, and the rest as test set. We divide each set into non-overlapping segments of consecutive 65 time stamps of the irregularly sampled time series, with each input instance consisting of 64 time stamps of the irregularly sampled time series, and we aim to forecast 8 consecutive time stamps starting from the last time stamp of the segment. The input is fed through the hybrid methods in a recurrent fashion; by changing the time duration of the last step of the ODE integration, we can forecast the output in the different time stamps. The output of the hybrid method is passed to a single dense layer to generate the output time series. In our experiments, we compare different ODE-based models hybrid with RNNs. The ODE of each model is parametrized by a 3-layer network, whereas the RNN is parametrized by a simple dense network; the total number of parameters for ODE-RNN, ANODE-RNN, SONODE-RNN, HBNODE-RNN, and GHBNODE-RNN with 16, 22, 14, 15, 15 augmented dimensions are 15,986, 16,730, 16,649, 16,127, and 16,127, respectively. To avoid potential error due to the ODE solver, we use a tolerance of 10^{-7}.

In training those hybrid models, we regularize the models by penalizing the L2 distance between the RNN output and the values of the next time stamp. Due to the second-order natural of the underlying dynamics [65], ODE-RNN and ANODE-RNN learn the dynamics very poorly with much larger training and test losses than the other models even they take smaller NFEs. HBNODE-RNN and GHBNODE-RNN give better prediction than SONODE-RNN using less backward NFEs (Fig. 12).

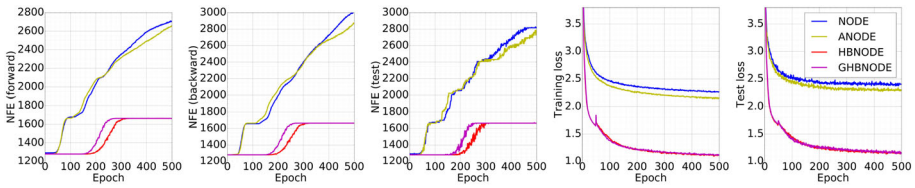

Fig. 13 Contrasting ODE-RNN, ANODE-RNN, SONODE-RNN, HBNODE-RNN, and GHBNODE-RNN for the Walker-2D kinematic simulation

3.4.4 Walker2D kinematic simulation

We evaluate the performance of HBNODE-RNN and GHBNODE-RNN on the Walker2D kinematic simulation task, which requires learning long-term dependency effectively [48]. The dataset [8] consists of a dynamical system from kinematic simulation of a person walking from a pre-trained policy, aiming to learn the kinematic simulation of the MuJoCo physics engine [98]. The dataset is irregularly sampled with 10% of the data removed from the simulation. Each input consists of 64 time stamps fed though the hybrid methods in a recurrent fashion, and the output is passed to a single dense layer to generate the output time series. The goal is to provide an auto-regressive forecast so that the output time series is as close as the input sequence shifted one time stamp to the right. We compare ODE-RNN (with 7 augmentation), ANODE-RNN (with 7 ANODE style augmentation), HBNODE-RNN (with 7 augmentation), and GHBNODE-RNN (with 7 augmentation). The RNN is parametrized by a 3-layer network, whereas the ODE is parametrized by a simple dense network. The number of parameters of the above four models is 8,729, 8,815, 8,899, and 8,899, respectively. In Fig. 13, we compare the performance of the above four models on the Walker2D benchmark; HBNODE-RNN and GHBNODE-RNN not only require significantly less NFEs in both training (forward and backward) and in testing than ODE-RNN and ANODE-RNN, but also have much smaller training and test losses.

3.5 Summary of our contributions and limitations

In this section, we present HBNODEs, a new class of continuous-depth neural networks motivated by the continuous limit of the classical momentum method. HBNODE enjoys computational efficiency since its adjoint equation is also an HBNODE. Also, HBNODE can learn long-term dependencies effectively as the adjoint state of HBNODE does not vanish. There are two open problems: (1) Are HBNODEs robust to noisy information, and whether HBNODEs are more robust than NODEs? (2) Can we extend HBNODEs to accelerate learning continuous normalizing flows [30]? One particular challenge is that HBNODEs are not learning one-to-one maps.

4 Transformers

We further show that momentum can be integrated into transformers, which can significantly reduce the computational and memory costs of the standard transformer [101] and enhance the performance of linear transformers [40].

The self-attention mechanism is a fundamental building block of transformers [43, 101]. Given an input sequence $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times D_x}$ of N feature vectors, the self-attention transformers it into another sequence $\hat{\boldsymbol{V}} = [\hat{\boldsymbol{v}}_1, \hat{\boldsymbol{v}}_2, \cdots, \hat{\boldsymbol{v}}_N]^\top \in \mathbb{R}^{N\times D_v}$ as follows

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \text{softmax}\Big(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\Big)\boldsymbol{v}_j, \quad \text{for } i = 1, \cdots, N, \tag{64}$$

where the scalar $\text{softmax}((\boldsymbol{q}_i^\top \boldsymbol{k}_j)/\sqrt{D})$ can be understood as the attention $\hat{\boldsymbol{v}}_i$ pays to the input feature $\boldsymbol{x}_j$. The vectors $\boldsymbol{q}_i, \boldsymbol{k}_j$, and $\boldsymbol{v}_j$ are called the query, key, and value vectors, respectively; these vectors are computed as follows:

$$\begin{aligned} [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_N]^\top &:= \boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q^\top \in \mathbb{R}^{N\times D}, \\ [\boldsymbol{k}_1, \boldsymbol{k}_2, \cdots, \boldsymbol{k}_N]^\top &:= \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K^\top \in \mathbb{R}^{N\times D}, \\ [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_N]^\top &:= \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V^\top \in \mathbb{R}^{N\times D_v}, \end{aligned} \tag{65}$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K \in \mathbb{R}^{D\times D_x}$, and $\boldsymbol{W}_V \in \mathbb{R}^{D_v\times D_x}$ are the weight matrices. We can further write (64) into the following compact form

$$\hat{\boldsymbol{V}} = \text{softmax}\Big(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\Big)\boldsymbol{V}, \tag{66}$$

where the softmax function is applied to each row of $(\boldsymbol{Q}\boldsymbol{K}^\top)/\sqrt{D}$. Equation (66) is also called the "scaled dot-product attention" or "softmax attention". Each transformer layer $T_\ell(\cdot)$ is defined via the following residual connection,

$$T_\ell(\boldsymbol{X}) = f_\ell(\hat{\boldsymbol{V}} + \boldsymbol{X}), \tag{67}$$

where f_ℓ is a function that transforms each feature vector independently and usually chosen to be a feedforward network. We call a transformer built with softmax attention standard transformer or transformer. It is easy to see that both memory and computational complexity of (66) are $\mathcal{O}(N^2)$ with N being the length of the input sequence. We can further introduce causal masking into (66) for autoregressive applications [101].

Transformers have become the state-of-the-art model for solving many challenging problems in natural language processing [2,9,19,22,38,76,101,112] and computer vision [21,23,88,99]. Nevertheless, the quadratic memory and computational cost of computing the softmax attention (66) are a major bottleneck for applying transformers to large-scale applications that involve very long sequences, such as those in [39,55,68]. Thus, much recent research on transformers has been focusing on developing efficient transformers, aiming to reduce the memory and computational complexities of the model [1,5,7,15, 17,36,40,45,68,70,73,78,79,84,86,89,93,94,103,110,110,115,116]. A thorough survey of recent advances in efficient transformers is available at [96]. These efficient transformers have better memory and/or computational efficiency at the cost of a significant reduction in accuracy.

In this section, we first present our motivation for integrating momentum into transformers in Sect. 4.1. In Sect. 4.2, we review the idea of improving the efficiency of transformers using low-rank approximation and formulating the low-rank version of transformers into RNNs. In Sect. 4.3, we present our strategy for integrating momentum into transformers. We show the advantages of momentum-integrated transformers over the vanilla ones in Sect. 4.4.

4.1 Motivation

In [40], the authors have established a connection between transformers and RNNs through the kernel trick. They proposed the linear transformer, which can be considered a rank-one approximation of the softmax transformer. Linear transformers have computational advantages in training, test, and inference: the RNN formulation (see (71)) enjoys fast inference, especially for autoregressive tasks, and the unrolled RNN formulation (see (69)) is efficient for fast training. See Sect. 4.2 for a detailed review of the linear transformer and its advantages. [62] proposes integrating momentum into RNNs to accelerate training RNNs and improve learning long-term dependencies. We notice that MomentumRNN also enjoys a closed unrolling form, which is quite unique among existing techniques for improving RNNs, enabling fast training, test, and inference; see Sect. 4.3 for details. As such, in this section we study *how momentum improves linear transformers?*

4.2 Linear transformer

Transformers learn long-term dependencies in sequences effectively and concurrently through the self-attention mechanism. Note we can write (64) as $\hat{\boldsymbol{v}}_i = (\sum_{j=1}^N k(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j)/(\sum_{j=1}^N k(\boldsymbol{q}_i, \boldsymbol{k}_j))$, where $k(\boldsymbol{q}_i, \boldsymbol{k}_j) := \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j/\sqrt{D})$. In linear transformers [17,40,86,110], the feature map $k(\boldsymbol{q}_i, \boldsymbol{k}_j)$ is linearized as the product of feature maps $\phi(\cdot)$ on the vectors $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$, i.e., $k(\boldsymbol{q}_i, \boldsymbol{k}_j) = \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)$. The associative property of matrix multiplication is then utilized to derive the following efficient computation of the attention map

$$\hat{\boldsymbol{v}}_i = \frac{\sum_{j=1}^N k(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^N k(\boldsymbol{q}_i, \boldsymbol{k}_j)} = \frac{\sum_{j=1}^N \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^N \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)} = \frac{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^N \phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top}{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^N \phi(\boldsymbol{k}_j)}. \tag{68}$$

In the matrix-product form, we can further write (68) as follows

$$\hat{\boldsymbol{V}} = \frac{\phi(\boldsymbol{Q})(\phi(\boldsymbol{K})^\top \boldsymbol{V})}{\phi(\boldsymbol{Q})\phi(\boldsymbol{K})^\top}. \tag{69}$$

Replacing $(\phi(\boldsymbol{Q})\phi(\boldsymbol{K}^\top))\boldsymbol{V}$ with $\phi(\boldsymbol{Q})(\phi(\boldsymbol{K}^\top)\boldsymbol{V})$ reduces the memory and computational cost of computing the attention map from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$, making linear transformers scalable to very long sequences.

Causal masking can be easily implemented in the linearized attention by truncating the summation term in the last equation of (68), resulting in

$$\hat{\boldsymbol{v}}_i = \frac{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^i \phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top}{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^i \phi(\boldsymbol{k}_j)} := \frac{\phi(\boldsymbol{q}_i)^\top \boldsymbol{s}_i}{\phi(\boldsymbol{q}_i)^\top \boldsymbol{z}_i}, \tag{70}$$

where $\boldsymbol{s}_i = \sum_{j=1}^i \phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top$ and $\boldsymbol{z}_i = \sum_{j=1}^i \phi(\boldsymbol{k}_j)$. The states $\boldsymbol{s}_i$ and $\boldsymbol{z}_i$ can be computed recurrently.

Efficient inference via the RNN formulation. Self-attention processes tokens of a sequence concurrently, enabling fast training of transformers. However, during inference, the output for timestep i is the input for timestep $i+1$. As a result, the inference in standard transformers cannot be parallelized and is thus inefficient. Linear transformers provide an elegant approach to fixing this issue by leveraging their RNN formulation. In particular, we can further write the linear attention with causal masking in (70) into the following

RNN form[7]

$$\begin{aligned} \boldsymbol{s}_i &= \boldsymbol{s}_{i-1} + \phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top; \\ \boldsymbol{z}_i &= \boldsymbol{z}_{i-1} + \phi(\boldsymbol{k}_i); \\ \hat{\boldsymbol{v}}_i &= \frac{\phi(\boldsymbol{q}_i)^\top \boldsymbol{s}_i}{\phi(\boldsymbol{q}_i)^\top \boldsymbol{z}_i}, \end{aligned} \tag{71}$$

where $\boldsymbol{s}_0 = \boldsymbol{0}$ and $\boldsymbol{z}_0 = \boldsymbol{0}$. Note that this RNN formulation of linear transformers with causal masking contains two memory states $\boldsymbol{s}_i$ and $\boldsymbol{z}_i$.

4.3 Momentum transformer

In this section, we present the *momentum transformer*. We start by integrating the heavy-ball momentum into the RNN formulation of causal linear attention in (71), resulting in the causal momentum attention. Next, we generalize the causal momentum attention to momentum attention that can efficiently train the model. Moreover, we propose the *momentum connection* to replace residual connections between the attention $\hat{\boldsymbol{V}}$ and the input $\boldsymbol{X}$ in (67) to boost the model's performance. Finally, we derive the adaptive momentum attention from the theory of optimal choice of momentum for the heavy-ball method.

4.3.1 Momentum transformer

Integrating momentum into causal linear attention. Now we consider integrating momentum into causal linear attention. We integrate momentum into the state $\boldsymbol{s}_i$ in (71) only since the denominator in causal linear attention is simply a normalizing scalar. If we regard $-\phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top$ as the gradient vector in (3), then we can add momentum into the state $\boldsymbol{s}_i$ by following the heavy-ball method in (2), resulting in the following RNN formulation of causal momentum attention,

$$\begin{aligned} \boldsymbol{m}_i &= \beta \boldsymbol{m}_{i-1} - \phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top; \\ \boldsymbol{s}_i &= \boldsymbol{s}_{i-1} - \gamma \boldsymbol{m}_i; \\ \boldsymbol{z}_i &= \boldsymbol{z}_{i-1} + \phi(\boldsymbol{k}_i); \\ \hat{\boldsymbol{v}}_i &= \frac{\phi(\boldsymbol{q}_i)^\top \boldsymbol{s}_i}{\phi(\boldsymbol{q}_i)^\top \boldsymbol{z}_i}, \end{aligned} \tag{72}$$

where $\boldsymbol{m}_0 = \boldsymbol{0}$, and $\gamma > 0$ and $0 \le \beta < 1$ are two hyperparameters. The RNN formulation of causal momentum attention in (72) is efficient for autoregressive inference. For training, we need to rewrite (72) into a form that is similar to (70). To this end, we need to eliminate $\boldsymbol{m}_i, \boldsymbol{s}_i$, and $\boldsymbol{z}_i$ from (72). Note that

$$\boldsymbol{s}_i = \boldsymbol{s}_{i-1} - \underbrace{\gamma \boldsymbol{m}_i}_{:=\boldsymbol{p}_i} = \underbrace{\boldsymbol{s}_0}_{=\boldsymbol{0}} - \Big(\boldsymbol{p}_i + \boldsymbol{p}_{i-1} + \cdots + \boldsymbol{p}_1\Big),$$

[7] We omit the nonlinearity (a two-layer feedforward network) compared to [40].

since $\boldsymbol{m}_i = \beta \boldsymbol{m}_{i-1} - \phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top$, we have $\boldsymbol{p}_i = \beta \boldsymbol{p}_{i-1} - \gamma\phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top$. Therefore,

$$\begin{aligned}
\boldsymbol{s}_i &= -(\boldsymbol{p}_i + \boldsymbol{p}_{i-1} + \cdots + \boldsymbol{p}_1) = \gamma\phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top - \Big((1+\beta)\boldsymbol{p}_{i-1} + \boldsymbol{p}_{i-2} + \cdots + \boldsymbol{p}_1\Big) \\
&= \gamma\phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top + \gamma(1+\beta)\phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top - \Big((1+\beta)^2\boldsymbol{p}_{i-2} + \cdots + \boldsymbol{p}_1\Big) \\
&= \cdots \\
&= \gamma \sum_{j=1}^{i} \frac{1-\beta^{i-j+1}}{1-\beta}\phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top \text{ for } i \geq 1.
\end{aligned}$$

We can then formulate the causal momentum attention as follows

$$\hat{\boldsymbol{v}}_i = \frac{\gamma\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^{i} \left(\frac{1-\beta^{i-j+1}}{1-\beta}\phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top\right)}{\phi(\boldsymbol{q}_i)^\top \boldsymbol{z}_i}. \tag{73}$$

Note that (73) is mathematically equivalent to (72), but it can be trained much more efficiently in a concurrent fashion via layer-wise parallelism.

Remark 7 Comparing (73) with (70) , we see that momentum plays a role in reweighting the terms $\{\phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top\}_{j=1}^i$. It is interesting to note that this reweighting is opposite to that used for reweighting the local attention [19]. It has also been noticed that low-rank attention can complement local attention, resulting in improved performance [63].

Integrating momentum into linear attention. To obtain momentum attention without causal masking, we can simply take the sum from 1 to N instead of summing from 1 to i. Therefore, we obtain the following momentum attention

$$\hat{\boldsymbol{v}}_i = \frac{\gamma\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^{N} \left(\frac{1-\beta^{N-j+1}}{1-\beta}\phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top\right)}{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^{N} \phi(\boldsymbol{k}_j)}. \tag{74}$$

Memory and computational complexity. Training momentum transformers have the same memory and computational complexities of $\mathcal{O}(N)$ as the training of linear transformers. For test and inference, momentum transformers also have the same memory and computational complexities as linear transformers. However, in the RNN form, momentum transformers require slightly more memory than linear transformers to store the extra momentum state $\boldsymbol{m}_i$.

4.3.2 Momentum connection

Each transformer layer has a residual connection between the self-attention output and the input as shown in (67). We further integrate momentum into (67) and derive the momentum connection as follows:

$$T_\ell(\boldsymbol{X}) = f_\ell\big(\hat{\boldsymbol{V}} + \boldsymbol{X} + \tilde{\beta}(\boldsymbol{X} - T_{\ell-1}(\boldsymbol{X}))\big), \quad 0 \leq \tilde{\beta} < 1. \tag{75}$$

Adaptive momentum. Our momentum transformer introduces additional hyperparameters γ and β, as well as $\tilde{\beta}$, compared to the linear transformer. Often γ can be simply set to 1. However, tuning β and $\tilde{\beta}$ can introduce extra computational cost for training

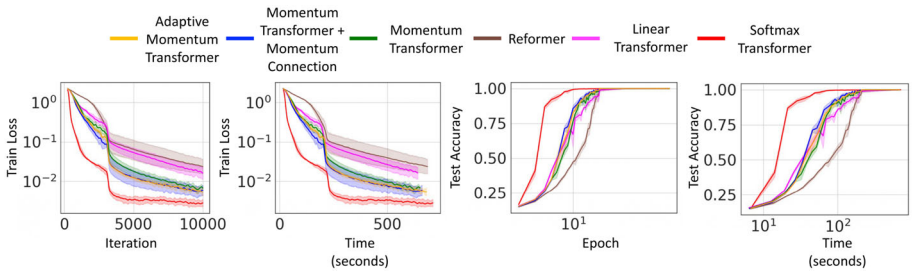

Fig. 14 Convergence comparison of adaptive momentum, momentum, reformer, linear, and softmax transformer on the sequence copy task. Momentum and adaptive momentum transformers converge faster and achieve better training loss than both linear transformer and reformer. Softmax transformer converges the fastest but suffers from quadratic memory and computational complexity. Adaptive momentum transformer performs as well as momentum transformer without intensively searching for momentum values

transformers. Moreover, using a constant momentum may not give us optimal performance. In this part, we will introduce an adaptive momentum formula for computing the momentum hyperparameter in momentum connection and thus eliminating the computational overhead for tuning $\tilde{\beta}$. Here, the adaptive momentum does not apply to β since it will break the closed unrolling form in (73). Adaptive momentum has been used in optimization, see, e.g., [91,108]; here, we use the later one for its simplicity.

In practice, for a given step size γ, we restrict the adaptive momentum to be in the range $[0, 1-\delta]$ with δ being the threshold parameter, and we choose it to be 10^{-3} in this work. Hence, we have the following adaptive momentum

$$\mathbf{proj}_{[0,1-\delta]}\left(1-\sqrt{\gamma\frac{\|\nabla f(\boldsymbol{x}^k)-\nabla f(\boldsymbol{x}^{k-1})\|}{\|\boldsymbol{x}^k-\boldsymbol{x}^{k-1}\|}}\right)^2, \tag{76}$$

where $\mathbf{proj}_{[0,1-\delta]}(\cdot) := \max(0, \min(1-\delta, \cdot))$. To simplify our computation, we apply the gradient descent update to approximate $\boldsymbol{x}^k - \boldsymbol{x}^{k-1}$, i.e., we approximate $\boldsymbol{x}^k - \boldsymbol{x}^{k-1}$ by $\gamma\nabla f(\boldsymbol{x}^{k-1})$, and we end up with

$$\beta_k := \mathbf{proj}_{[0,1-\delta]}\left(1-\sqrt{\frac{\|\nabla f(\boldsymbol{x}^k)-\nabla f(\boldsymbol{x}^{k-1})\|}{\|\nabla f(\boldsymbol{x}^{k-1})\|}}\right)^2. \tag{77}$$

Remark 8 The adaptive momentum in (77) can also be used for the MomentumRNNs but not for heavy-ball neural ODEs since such a choice for momentum hyperparameter will destroy the nice properties of the adjoint equation of heavy-ball neural ODEs.

4.4 Experimental results

We evaluate the benefits of our momentum transformers in terms of convergence speed, efficiency, and accuracy. We compare the performance of momentum and adaptive momentum transformers with the baseline standard softmax transformer and several other efficient transformers in the following tasks: (1) the synthetic copy task, (2) the MNIST and CIFAR image generation task, (3) Long-Range Arena [95], and (4) the non-autoregressive machine translation task. These tasks are among standard benchmarks for

measuring the performance of transformers and their efficiency. The tasks we choose also cover different data modalities—text and image—and a variety of model sizes. Our experimental results confirm that momentum and adaptive momentum transformers outperform many existing efficient transformers, including linear transformers and reformers, in accuracy and converge faster. Furthermore, adaptive momentum transformer improves over momentum transformer without the need of searching for momentum hyperparameter.

4.4.1 Copy task

We train momentum transformers and baseline models on a synthetic copy task to analyze their convergence speed. In this task, the model has to duplicate a sequence of symbols. Each training and test sample has the form $0w0w$ where w is a sequence of symbols collected from the set $\{1, \ldots, N\}$.

In our experiments, we follow the same experimental setting as that used in [40]. In particular, we use a sequence of maximum length 128 with 10 different symbols separated by a separator symbol. The baseline architecture for all methods is a 4-layer transformer with 8 attention heads and $D = 32$. The models are trained with the RAdam optimizer using a batch size of 64 and a learning rate of 10^{-3} which is reduced to 10^{-4} after 3000 iterations. Figure 14 shows the training loss and the test accuracy over epochs and over GPU time. Both the momentum and the adaptive momentum transformers converge much faster and achieve better training loss than the linear transformer. Notice that while the standard transformer converges the fastest, it has quadratic complexity. Adaptive momentum transformer has similar performance as the momentum transformer without the need of tuning for the momentum value.

4.4.2 Image generation

Transformers have shown great promise in autoregressive generation applications [15,75], such as autoregressive image generation [77]. However, the training and sampling procedure using transformers are quite slow for these tasks due to the quadratic computational time complexity and the memory scaling with respect to the sequence length. In this section, we train our momentum-based transformers and the baselines with causal masking to predict images pixel by pixel and compare their performance. In particular, we demonstrate that, like linear transformers, both momentum and adaptive momentum transformers are able to generate images much faster than the standard softmax transformer. Furthermore, we show that momentum-based transformers converge much faster than linear transformers while achieving better bits per dimension (bits/dim). Momentum and adaptive momentum transformers also generate images with constant memory per image like linear transformers.

MNIST. We first examine our momentum-based transformers on the MNIST image generation task. For all methods, we train a 8-layer transformer with 8 attention heads and the embedding size of 256, which corresponds to 32 dimensions per head. The feedforward dimensions are 4 times larger than the embedding size. A mixture of 10 logistics is used to model the output as in [82]. For training, we use the RAdam optimizer with a learning rate of 10^{-4} and train all models for 250 epochs except for the adaptive momentum transformer.

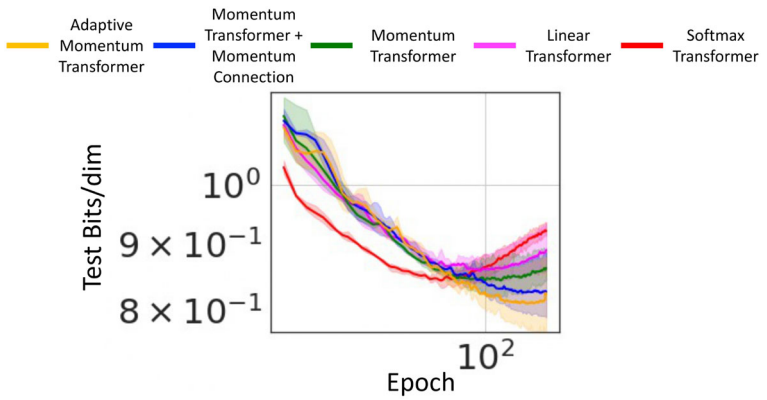

Fig. 15 Momentum transformers outperform linear transformers on the MNIST image generation task. Adaptive momentum transformer achieves the best test bits/dim

Table 6 Momentum transformers achieve better test bits/dim than both softmax and linear transformers on MNIST generation

Method	Bits/dim	Images/s	
Standard softmax transformer	0.84	0.45	(1×)
Linear transformer	0.85	142.8	(317×)
Momentum transformer	0.84	139.7	(310×)
Momentum transformer + momentum connection	0.82	135.5	(301×)
Adaptive momentum transformer	0.80	134.9	(300×)

We report the bits/dim and image generation throughput in Table 6. Compared to the linear transformer, all momentum-based transformers not only attain better bits/dim but also have comparable image generation throughput, justifying the linear complexity of our models. In addition, we demonstrate that the adaptive momentum transformer converges much faster than the baseline models in Fig. 15. Momentum-based transformers even outperform softmax transformers in this task (Table 7).

CIFAR10. Next, we investigate the advantages of our momentum-based transformers when the sequence length and the number of layers in the model increase. We consider the CIFAR-10 image generation task, in which we train 16-layer transformers to generate CIFAR-10 images. The configuration for each layer is the same as in the MNIST experiment. For the linear transformer and our momentum-based transformer, we use a batch size of 4 while using a batch size of 1 for the standard softmax transformer due to the memory limit of the largest GPU available to us, i.e., NVIDIA V100. This is similar to the setting in [40]. Like in the MNIST image generation task, our momentum-based transformers outperform the linear transformer in terms of bits/dim while maintaining comparable image generation throughput. This is a very expensive task, limiting us to perform a thorough hyperparameter search; we believe better results can be obtained with a more thorough hyperparameter search.

4.4.3 Long-Range Arena

In this experiment, we evaluate our model on tasks that involve longer sequence lengths in the long-range Arena (LRA) benchmark [95]. We show that the momentum-based transformer outperforms the baseline linear transformer and standard softmax trans-

Table 7 Momentum-based transformers achieve better test bits/dim than linear transformer on CIFAR10 image generation task

Method	Bits/dim	Images/s	
Standard softmax transformer	3.20	0.004	(1×)
Linear transformer	3.44	17.85	(4462×)
Momentum transformer	3.43	17.52	(4380×)
Momentum transformer + momentum connection	3.41	17.11	(4277×)
Adaptive momentum transformer	3.38	17.07	(4267×)

Table 8 Results on the LRA tasks

Model	ListOps (2K)	Text (4K)	Retrieval (4K)	Image (1K)	Pathfinder (1K)	Avg
Softmax [101]	37.10 (37.10)	64.17 (65.02)	80.71 (79.35)	39.06 (38.20)	72.48 (74.16)	**58.70 (58.77)**
Linear [40]	18.30	64.22	81.37	38.29	71.17	54.67
Performer [17]	18.80	63.81	78.62	37.07	69.87	53.63
Reformer [45]	19.05	64.88	78.64	43.29	69.36	55.04
Linformer [110]	**37.25**	55.91	79.37	37.84	67.60	55.59
Momentum transformer	19.56	64.35	81.95	39.40	73.12	55.68
Adaptive momentum transformer	20.16	**64.45**	**82.07**	**39.53**	**74.00**	56.04

We report the test classification accuracy for each task and average accuracy across all tasks. The momentum-based transformers, in particular, the adaptive momentum transformer, outperforms all other transformers except on the ListOps. The numbers in the parenthesis are from the paper [115]. Unit: %

Table 9 BLEU scores and tokens per second from machine translation models trained on IWSLT show the advantages of our momentum-based transformers

Method	BLEU Score	Speed (tokens/s)
Standard softmax transformer	24.34	5104
Linear transformer	21.37	1382
Momentum transformer	22.11	1398
Momentum transformer + momentum connection	22.14	1403
Adaptive momentum transformer	22.20	1410

The number of trainable parameters is almost the same for all models, up to the small difference introduced by the momentum mechanism in our models. Momentum-based transformers outperform the linear transformer in generation quality in terms of BLEU score and obtain comparable generation efficiency in terms of tokens per second

former [101], justifying the advantage of our momentum-based transformers in capturing long-term dependency.

Datasets and metrics. We consider all five tasks in the LRA benchmark [95], including ListOps, byte-level IMDb reviews text classification, byte-level document retrieval, CIFAR-10 classification on sequences of pixels, and Pathfinder. These tasks involve long sequences of length $2K$, $4K$, $4K$, $1K$, and $1K$, respectively. We follow the setup/evaluation protocol in [95] and report test accuracy for each task and the average result across all tasks.

Models and training. All models have 2 layers, 64 embedding dimension, 128 hidden dimension, 2 attention heads. Mean pooling is applied in all models. Also, we use the nonlinear activation $elu(x) + 1$ for the linear transformer. Our implementation uses the public code in [115] as a starting point, and we follow their training procedures. The training setting and additional baseline model details are provided in the configuration file used in [115].

Results. We summarize our results in Table 8. Both momentum-based transformers outperform linear transformers in all tasks and yield better accuracy than the standard soft-

max transformer in most tasks except the ListOps. The adaptive momentum transformer performs the best on every task except the LipsOps, far behind the softmax transformer and Linformer.

4.4.4 Non-autoregressive machine translation

All of the above experiments are for auto-regressive tasks. In this last experiment, we demonstrate that the benefits of our momentum-based transformers also hold for a non-autoregressive task. We consider a machine translation task on the popular IWSLT' 16 En-De dataset. We follow the setting in [51]. In particular, we tokenize each sentence using a script from Moses [46] and segment each word into subword units using BPE [85]. We also use $40K$ tokens from both source and target. Our baseline model is the small transformer-based network in [51]. This model has 5 layers, and each layer has 2 attention heads. We replace the softmax attention in this network with the linear and momentum-based attention to obtain the linear transformer baseline and the momentum-based transformer models, respectively.

Table 9 reports the results in terms of generation quality, measured by the BLEU score [67], and generation efficiency, measured by the number of generated tokens per second. Consistent with other experiments above, our momentum-based transformers obtain better BLEU scores than the linear transformer in this non-autoregressive setting. Furthermore, in terms of generation efficiency, momentum-based models are comparable with the linear transformer and much more efficient than the standard softmax transformer.

4.5 Summary of our contributions and limitations

We develop a new class of efficient transformers, i.e., momentum transformers, which have the same memory and computational complexity as the recently developed linear transformer. We develop momentum transformers based on an analogy between the RNN formulation of causal linear attention and gradient descent. Then, we integrate the momentum into causal linear attention following the heavy ball method. There are numerous avenues for future work: (1) Can we develop momentum transformers based on other popular optimization algorithms beyond the heavy ball method, e.g., Adam? (2) Can we design better weighting schemes to improve the performance of transformers?

5 Conclusion and future work

In this paper, we reviewed how to integrate momentum into neural networks to enhance their theoretical and practical performances. In particular, we showed that momentum improves learning long-term dependencies of RNNs and neural ODEs and significantly reduces their computational costs. Moreover, we showed that momentum can also be used to improve the efficiency and accuracy of transformers. There are numerous directions for future work: (1) Can we leverage the momentum-augmented neural network component to aid the neural architecture search? (2) Can we further improve the momentum-integrated architectures by using the numerical ODE insights [31]? (3) Momentum has also been used in designing CNNs [53,83]; it is also worth further studying the benefits of momentum for CNNs.

Acknowledgements

This material is based on research sponsored by NSF Grants DMS-1924935, DMS-1952339, DMS-2110145, DMS-2152762, DMS-2208361, DOE Grant DE-SC0021142, and ONR grant N00014-18-1-2527 and the ONR MURI Grant N00014-20-1-2787.

Data and Code Availability
All related code and data have been made available on Github.

Author details
[1]Department of Mathematics, Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA, [2]Department of Mathematics, UCLA, Los Angeles, CA, USA.

Declarations

Conflict of interest
There is no conflict of interest.

Received: 13 October 2021 Accepted: 31 July 2022
Published online: 26 August 2022

References

1. Ainslie, J., Ontanon, S., Alberti, C., Cvicek, V., Fisher, Z., Pham, P., Ravula, A., Sanghai, S., Wang, Q., Yang, L.: ETC: Encoding long and structured inputs in transformers. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pp. 268–284 (2020)
2. Al-Rfou, R., Choe, D.K., Constant, N., Guo, M., Jones, L.: Character-level language modeling with deeper self-attention. In: Thirty-Third AAAI Conference on Artificial Intelligence (2019)
3. Arjovsky, M., Shah, A., Bengio, Y.: Unitary evolution recurrent neural networks. In: International Conference on Machine Learning, pp. 1120–1128 (2016)
4. Attouch, H., Goudou, X., Redont, P.: The heavy ball with friction method, I. The continuous dynamical system: global exploration of the local minima of a real-valued function by asymptotic analysis of a dissipative dynamical system. Commun. Contemp. Math. **2**(01), 1–34 (2000)
5. Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer. (2020). arXiv preprint arXiv:2004.05150
6. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
7. Blanc, G., Rendle, S.: Adaptive sampled softmax with kernel based sampling. In: Dy, J., Krause, A. (eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 590–599. PMLR (2018)
8. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym (2016). cite arxiv:1606.01540
9. Brown, T., et al.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds), Advances in Neural Information Processing Systems, Vol. 33, pp. 1877–1901 (2020)
10. Casado, M.L.: Optimization with orthogonal constraints and on general manifolds. (2019). https://github.com/Lezcano/expRNN
11. Casado, M.L.: Trivializations for gradient-based optimization on manifolds. In: Advances in Neural Information Processing Systems, pp. 9154–9164 (2019)
12. Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D.: Adversarial attacks and defences: a survey. (2018). arXiv preprint arXiv:1810.00069
13. Chandar, S., Sankar, C., Vorontsov, E., Kahou, S.E., Bengio, Y.: Towards non-saturating recurrent units for modelling long-term dependencies. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, pp. 3280–3287 (2019)
14. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 6572–6583 (2018)
15. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. (2019). arXiv preprint arXiv:1904.10509
16. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. (2014). arXiv preprint arXiv:1406.1078
17. Choromanski, K.M., et al.: Rethinking attention with performers. In: International Conference on Learning Representations (2021)
18. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Gated feedback recurrent neural networks. In: International Conference on Machine Learning, pp. 2067–2075 (2015)
19. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q.V., Salakhutdinov, R.: Transformer-xl: Attentive language models beyond a fixed-length context. (2019). arXiv preprint arXiv:1901.02860
20. Daulbaev, T., Katrutsa, A., Markeeva, L., Gusak, J., Cichocki, A., Oseledets, I.: Interpolation technique to speed up gradients propagation in neural odes. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds), Advances in Neural Information Processing Systems, volume 33, pp. 16689–16700. Curran Associates, Inc. (2020)
21. Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., Kaiser, L.: Universal transformers. (2018). arXiv preprint arXiv:1807.03819
22. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. (2018). arXiv preprint arXiv:1810.04805
23. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. (2020). arXiv preprint arXiv:2010.11929

24. Du, J., Joseph, D.-V., Finale: Model-based reinforcement learning for semi-markov decision processes with neural odes. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.), Advances in Neural Information Processing Systems, Vol. 33, pp. 19805–19816. Curran Associates, Inc. (2020)
25. Dupont, E., Doucet, A., Teh, Y.W.: Augmented neural odes. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.), Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc. (2019)
26. Elman, J.L.: Finding structure in time. Cogn. Sci. **14**(2), 179–211 (1990)
27. Finlay, C., Jacobsen, J.-H., Nurbekyan, L., Oberman, A.: How to train your neural ODE: the world of Jacobian and kinetic regularization. In: Hal Daumé, III., Singh, A. (eds.), Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pp. 3154–3164. PMLR, 13–18 (2020)
28. Garofolo, J.S.: Timit acoustic phonetic continuous speech corpus. Linguistic Data Consortium (1993)
29. Ghosh, A., Behl, H., Dupont, E., Torr, P., Namboodiri, V.: Steer : Simple temporal regularization for neural ode. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds), Advances in Neural Information Processing Systems, volume 33, pp. 14831–14843. Curran Associates, Inc. (2020)
30. Grathwohl, W., Chen, R.T.Q., Bettencourt, J., Duvenaud, D.: Scalable reversible generative models with free-form continuous dynamics. In: International Conference on Learning Representations (2019)
31. Haber, E., Ruthotto, L.: Stable architectures for deep neural networks. Inverse Prob. **34**(1), 014004 (2017)
32. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
33. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European Conference on Computer Vision, pp. 630–645. Springer (2016)
34. Helfrich, K., Willmott, D., Ye, Q.: Orthogonal recurrent neural networks with scaled Cayley transform. In: Dy, J., Krause, A., (eds), Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 1969–1978, Stockholmsmässan, Stockholm Sweden, 10–15 (2018). PMLR
35. Henaff, M., Szlam, A., LeCun, Y.: Recurrent orthogonal networks and long-memory tasks. In: Balcan, M.F., Weinberger, K.Q. (eds), Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pp. 2034–2042. New York, New York, USA 20–22 (2016). PMLR
36. Ho, J., Kalchbrenner, N., Weissenborn, D., Salimans, T.: Axial attention in multidimensional transformers. (2019). arXiv preprint arXiv:1912.12180
37. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
38. Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 328–339. Melbourne, Australia (2018). Association for Computational Linguistics
39. Huang, C.-Z.A., Vaswani, A., Uszkoreit, J., Simon, I., Hawthorne, C., Shazeer, N., Dai, A.M., Hoffman, M.D., Dinculescu, M, Eck, D.: Music transformer: Generating music with long-term structure. In: International Conference on Learning Representations (2018)
40. Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are rnns: Fast autoregressive transformers with linear attention. In: International Conference on Machine Learning, pp. 5156–5165. PMLR (2020)
41. Kelly, J., Bettencourt, J., Johnson, M.J., Duvenaud, D.K.: Learning differential equations that are easy to solve. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.), Advances in Neural Information Processing Systems, volume 33, pp. 4370–4380. Curran Associates, Inc. (2020)
42. Kidger, P., Morrill, J., Foster, J., Lyons, T.J.: Neural controlled differential equations for irregular time series. In: NeurIPS (2020)
43. Kim, Y., Denton, C., Hoang, L., Rush, A.M.: Structured attention networks. (2017). arXiv preprint arXiv:1702.00887
44. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. (2014). arXiv preprint arXiv:1412.6980
45. Kitaev, N., Kaiser, L., Levskaya, A.: Reformer: The efficient transformer. (2020). arXiv preprint arXiv:2001.04451
46. Koehn, P., et al.: Moses: Open source toolkit for statistical machine translation. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions, pp. 177–180 (2007)
47. Le, Q.V., Jaitly, N., Hinton, G.E: A simple way to initialize recurrent networks of rectified linear units. (2015). arXiv preprint arXiv:1504.00941
48. Lechner, M., Hasani, R.: Learning long-term dependencies in irregularly-sampled time series (2020)
49. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
50. LeCun, Y., Cortes, C., Burges, C.J.: MNIST handwritten digit database. 2, (2010). ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist
51. Lee, J., Mansimov, E., Cho, K.: Deterministic non-autoregressive neural sequence modeling by iterative refinement. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 1173–1182 (2018)
52. Lezcano-Casado, M., Martínez-Rubio, D.: Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In: International Conference on Machine Learning (ICML), pp. 3794–3803 (2019)
53. Li, H., Yang, Y., Chen, D., Lin, Z.: Optimization algorithm inspired deep neural network structure design. In: Asian Conference on Machine Learning, pp. 614–629. PMLR (2018)
54. Liu, P.J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., Shazeer, N.: Generating wikipedia by summarizing long sequences. In: International Conference on Learning Representations (2018)
55. Liu, P.J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., Shazeer, N.: Generating wikipedia by summarizing long sequences. (2018). arXiv preprint arXiv:1801.10198
56. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018)
57. Massaroli, S., Poli, M., Park, J., Yamashita, A., Asma, H.: Dissecting neural odes. In: 34th Conference on Neural Information Processing Systems, NeurIPS 2020. The Neural Information Processing Systems (2020)
58. Mhammedi, Z., Hellicar, A., Rahman, A., Bailey, J.: Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 2401–2409. JMLR. org (2017)

59. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh Annual Conference of the International Speech Communication Association (2010)
60. Nesterov, Y.: Introductory lectures on convex programming volume I: Basic course. (1998)
61. Nesterov, Y.E.: A method for solving the convex programming problem with convergence rate o ($1/k^2$). Dokl. Akad. Nauk SSSR **269**, 543–547 (1983)
62. Nguyen, T., Baraniuk, R., Bertozzi, A., Osher, S., Wang, B.: MomentumRNN: integrating momentum into recurrent neural networks. In: Advances in Neural Information Processing Systems (NeurIPS 2020), (2020)
63. Nguyen, T.M., Suliafu, V., Osher, S.J., Chen, L., Wang, B.: Fmmformer: Efficient and flexible transformer via decomposed near-field and far-field attention. (2021). arXiv preprint arXiv:2108.02347
64. Noël, J.-P., Schoukens, M.: F-16 aircraft benchmark based on ground vibration test data. In: 2017 Workshop on Nonlinear System Identification Benchmarks, pp. 19–23 (2017)
65. Norcliffe, A., Bodnar, C., Day, B., Simidjievski, N., Liò, P.: On second order behaviour in augmented neural odes. In: Advances in Neural Information Processing Systems (2020)
66. Pal, A., Ma, Y., Shah, V., Rackauckas, C.V.: Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In: Meila, M., Zhang, T., (eds.), Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pp. 8325–8335. PMLR (2021)
67. Papineni, K., Roukos, S., Ward, T., Zhu, W.-J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02, pp. 311–318. USA (2002). Association for Computational Linguistics
68. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., Tran, D.: Image transformer. In: Dy, Jennifer, Krause, Andreas, (eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 4055–4064. PMLR, 10–15 (Jul 2018)
69. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International Conference on Machine Learning, pp. 1310–1318 (2013)
70. Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N., Kong, L.: Random feature attention. In: International Conference on Learning Representations (2021)
71. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. **4**(5), 1–17 (1964)
72. Pontryagin, L.S.: Mathematical Theory of Optimal Processes. Routledge, New York (2018)
73. Qiu, J., Ma, H., Levy, O., Yih, S.W., Wang, S., Tang, J.: Blockwise self-attention for long document understanding. (2019). arXiv preprint arXiv:1911.02972
74. Quaglino, A., Gallieri, M., Masci, J., Koutník, J.: Snode: Spectral discretization of neural odes for system identification. In: International Conference on Learning Representations (2020)
75. Radford, A., Jeffrey, W., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
76. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: SQuAD: 100,000+ questions for machine comprehension of text. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 2383–2392, Austin, Texas, (2016). Association for Computational Linguistics
77. Ramesh, A., Pavlov, M., Goh, G., Gray, S., Chen, M., Child, R., Misra, V., Mishkin, P., Krueger, G., Agarwal, S., Sutskever, I.: Dalle: Creating images from text. OpenAI blog (2020)
78. Rawat, A.S., Chen, J., Yu, F.X.X., Suresh, A.T., Kumar, S.: Sampled softmax with random fourier features. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., (eds), Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc. (2019)
79. Roy, A., Saffar, M., Vaswani, A., Grangier, D.: Efficient content-based sparse attention with routing transformers. Trans. Assoc. Comput. Linguist. **9**, 53–68 (2021)
80. Rubanova, Y., Chen, R.T.Q., Duvenaud, D.K.: Latent ordinary differential equations for irregularly-sampled time series. In: Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., Garnett, R. (eds.), Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc. (2019)
81. Salesforce. Lstm and qrnn language model toolkit for pytorch. (2017). https://github.com/salesforce/awd-lstm-lm
82. Salimans, T., Karpathy, A., Chen, X., Kingma, D.: Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In: International Conference on Learning Representations (2017)
83. Sander, M.E., Ablin, P., Blondel, M., Peyré, G.: Momentum residual neural networks. (2021). arXiv preprint arXiv:2102.07870
84. Schlag, I., Irie, K., Schmidhuber, J.: Linear transformers are secretly fast weight memory systems. CoRR, abs/2102.11174 (2021)
85. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1715–1725 (2016)
86. Shen, Z., Zhang, M., Zhao, H., Yi, S., Li, H.: Efficient attention: Attention with linear complexities. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 3531–3539 (2021)
87. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. nature **550**(7676), 354–359 (2017)
88. So, D.R., Liang, C., Le, Q.V.: The evolved transformer. (2019). arXiv preprint arXiv:1901.11117
89. Song, K., Jung, Y., Kim, D., Moon, I.-C.: Implicit kernel attention. (2021). arXiv preprint arXiv:2006.06147
90. Su, W., Boyd, S., Candes, E.: A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In: Advances in Neural Information Processing Systems, pp. 2510–2518 (2014)
91. Sun, T., Ling, H., Shi, Z., Li, D., Wang, B.: Training deep neural networks with adaptive momentum inspired by the quadratic optimization. (2021). arXiv preprint arXiv:2110.09057
92. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. (2013). arXiv preprint arXiv:1312.6199

93. Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., Zheng, C.: Synthesizer: Rethinking self-attention in transformer models. (2020). arXiv preprint arXiv:2005.00743
94. Tay, Y., Bahri, D., Yang, L., Metzler, D., Juan, D.-C.: Sparse Sinkhorn attention. In: Hal Daumé, I.I.I., Singh, Aarti (eds.), Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pp. 9438–9447. PMLR, 13–18 (2020)
95. Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., Metzler, D.: Long range arena : A benchmark for efficient transformers. In: International Conference on Learning Representations (2021)
96. Tay, Y., Dehghani, M., Bahri, D., Metzler, D.: Efficient transformers: A survey. (2020). arXiv preprint arXiv:2009.06732
97. Tieleman, T., Hinton, G.: Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning (2012)
98. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033 (2012)
99. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. (2020). arXiv preprint arXiv:2012.12877
100. Van Der Westhuizen, J., Lasenby, J.: The unreasonable effectiveness of the forget gate. (2018). arXiv preprint arXiv:1804.04849
101. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
102. Vorontsov, E., Trabelsi, C., Kadoury, S., Pal, C.: On orthogonality and learning recurrent networks with long term dependencies. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 3570–3578. JMLR. org (2017)
103. Vyas, A., Katharopoulos, A., Fleuret, F.: Fast transformers with clustered attention. Adv. Neural Inf. Process. Syst. **33**, (2020)
104. Wang, B., Lin, A., Yin, P., Zhu, W., Bertozzi, A.L., Osher, S.J.: Adversarial defense via the data-dependent activation, total variation minimization, and adversarial training. Inverse Problems Imaging **15**(1), 129–145 (2021)
105. Wang, B., Luo, X., Li, Z., Zhu, W., Shi, Z., Osher, S.: Deep neural nets with interpolating function as output activation. Advances in Neural Information Processing Systems (2018)
106. Wang, B., Nguyen, T.M., Bertozzi, A.L., Baraniuk, R.G., Osher, S.J.: Scheduled restart momentum for accelerated stochastic gradient descent. (2020). arXiv preprint arXiv:2002.10583
107. Wang, B., Osher, S.J.: Graph interpolating activation improves both natural and robust accuracies in data-efficient deep learning. Eur. J. Appl. Math. **32**(3), 540–569 (2021)
108. Wang, B., Ye, Q.: Stochastic gradient descent with nonlinear conjugate gradient-style adaptive momentum. (2020). arXiv preprint arXiv:2012.02188
109. Wang, B., Yuan, B., Shi, Z., Osher, S.: Resnets ensemble via the Feynman-Kac formalism to improve natural and robust accuracies. Adv. Neural Inf. Process. Syst. (2019)
110. Wang, S., Li, B., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. (2020). arXiv preprint arXiv:2006.04768
111. Wibisono, A., Wilson, A.C., Jordan, M.I.: A variational perspective on accelerated methods in optimization. Proc. Natl. Acad. Sci. **113**(47), E7351–E7358 (2016)
112. Williams, A., Nangia, N., Bowman, S.: A broad-coverage challenge corpus for sentence understanding through inference. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 1112–1122. (2018)
113. Wisdom, S., Powers, T., Hershey, J., Le Roux, J., Atlas, L.: Full-capacity unitary recurrent neural networks. In: Advances in Neural Information Processing Systems, pp. 4880–4888 (2016)
114. Xia, H., Suliafu, V., Ji, H., Nguyen, T.M., Bertozzi, A.L., Osher, S.J., Wang, B.: Heavy ball neural ordinary differential equations. (2021). arXiv preprint arXiv:2010.04840
115. Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., Singh, V.: Nyströmformer: A nyström-based algorithm for approximating self-attention. In: Proceedings of the AAAI Conference on Artificial Intelligence (2021)
116. Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., Ahmed, A.: Big bird: Transformers for longer sequences. (2021). arXiv preprint arXiv:2007.14062
117. Zhang, T., Yao, Z., Gholami, A., Gonzalez, J.E., Keutzer, K., Mahoney, M.W., Biros, G.: ANODEV2: A Coupled Neural ODE Framework. In: Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., Garnett, R. (eds.), Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., (2019)
118. Zhuang, J., Dvornek, N.C., Tatikonda, S., Duncan, J.: MALI: A memory efficient and reverse accurate integrator for neural odes. In: International Conference on Learning Representations (2021)

Publisher's Note