Neural Processing Letters (2022) 54:1195–1209
https://doi.org/10.1007/s11063-021-10676-7

A Dynamically Stabilized Recurrent Neural Network

Samer Saab Jr.[1] · Yiwei Fu[2] · Asok Ray[2,3] · Michael Hauser[2]

Accepted: 15 October 2021 / Published online: 30 October 2021

Abstract
This work proposes a novel recurrent neural network architecture, called the Dynamically Stabilized Recurrent Neural Network (DSRNN). The developed DSRNN includes learnable skip-connections across a specified number of time-steps, which allows for a state-space representation of the network's hidden-state trajectory, and a regularization term is introduced in the loss function in the setting of Lyapunov stability theory. The regularizer enables the placement of eigenvalues of the (linearized) transfer function matrix to desired locations in the complex plane, thereby acting as an internal controller for the hidden-state trajectories. In this way, the DSRNN adjusts the weights of temporal skip-connections to achieve recurrent hidden-state stability, which mitigates the problems of vanishing and exploding gradients. The efficacy of the DSRNN is demonstrated on a forecasting task of a recorded double-pendulum experimental model. The results show that the DSRNN outperforms both the Long Short-Term Memory (LSTM) and vanilla recurrent neural networks, and the relative mean-squared error of the LSTM is reduced by up to $\sim$99.64%. The DSRNN also showed comparable results to the LSTM on a classification task of two Lorenz oscillator systems.

Keywords Recurrent neural networks · Long short-term memory · Lyapunov stability

✉ Samer Saab Jr.
sys5880@psu.edu

Yiwei Fu
yiweifu1@gmail.com

Asok Ray
axr2@psu.edu

Michael Hauser
mikebenh@gmail.com

[1] Department of Electrical Engineering, The Pennsylvania State University, State College, PA 16802, USA

[2] Department of Mechanical Engineering, The Pennsylvania State University, State College, PA 16802, USA

[3] Department of Mathematics, The Pennsylvania State University, State College, PA 16802, USA

1 Introduction

Recurrent neural networks (RNN) have set the state-of-the-art on very large arrays of diverse time-series tasks [1]. Many RNN architectures have been developed to model temporal data, such as Long Short-Term Memory (LSTM) networks [2,3], and have shown great promise in processing time-series data for applications like natural language processing (NLP) [4], natural language generation [5], optimization (e.g., see [6–8]), human activity recognition [9], and medical time series data [10]. Primitively, RNNs act as non-linear dynamical systems intended to model sequential data. However, the stability of the dynamical system defined by RNNs is not guaranteed, as they notoriously fall victim to exploding or vanishing gradients. This suggests that the way in which RNNs are trained are often in a chaotic regime [11], and thus the stability of RNNs are of high interest [12,13].

Methods exist to stabilize the recurrent hidden states, such as layer normalization (LN) proposed by Ba et al. in [14], which fixes the mean and the variance of the summed inputs within each layer of the recurrent network to reduce the "covariate shift" problem in a computationally efficient manner. Other methods utilize direct connections, either spatially or temporally, in RNNs which aid in stabilizing the flow of the hidden state signals [15, 16]. For example, the Highway LSTMs (HLSTMs) developed in [17] create direct links between memory cells which allows information to flow between layers in a controlled fashion. In other words, these direct links stabilize the hidden states of the network by alleviating the vanishing gradients, allowing the HLSTM to train on data with arbitrary lengths. The residual LSTM [18] extends the intuition behind the Highway LSTM by also leveraging shortcut paths temporally by adding direct links between the output gates, while permanently turning on highway paths. However, for both HLSTMs and residual LSTMs, the performance degrades as the number of layers is increased. Furthermore, even though the HLSTM and residual LSTM alleviate vanishing gradients, a problem that remains from leveraging skip-connections in an RNN is that the hidden state trajectory is prone to diverging. This phenomenon can occur due to the hidden states being a function of the summation of all previously connected hidden states, which may result in oscillations. Such a situation prevents the network from learning, since the blown-up values of the hidden states will saturate their respective activation functions.

In view of the above motivation, this paper develops the Dynamically Stabilized Recurrent Neural Network, hereafter called the DSRNN, which is designed with a prespecified number of weighted skip-connections through time, where the weights of skip-connections are learnable parameters. Using this architecture of weighted skip-connections, we propose a novel methodology to stabilize the hidden states' trajectories; this is done by formulating a state-space representation of the hidden states. Thus, the hidden states can be analyzed as a discrete-time dynamical system, which is an efficient way to study the stability properties of the hidden states [19]. We then utilize Lyapunov's linearization method [20] to confine the hidden states' values within an acceptable region. Using the linearized state-space representation of the DSRNN, a novel regularizer is designed to adjust the weights of the skip-connections such that the linearized state-space model is stable, which ensures the non-divergence of hidden state trajectories.

Following Hauser and Ray [21], skip-connections are weighted in such a way that the network can learn up to kth-order smooth perturbation terms, where the intuitions behind learning arbitrary-order perturbation terms are borrowed for implementation in the DSRNN. However, unlike the layer-wise skip-connections in feedforward NNs, the weighted skip-connections in the DSRNN are across time-steps, allowing the DSRNN to learn temporal order perturbation terms. Furthermore, unlike the LN method [14] which is rather restricted by assuming that all channels in a layer make *similar* contributions, the DSRNN learns the weights of the skip-coefficients, which allows the DSRNN to stabilize the hidden state trajectory in a flexible manner.

In the context of control theory, the regularizer in the DSRNN acts as a stabilizer to the system, which is not to be confused with gating functions that designers call a controller cell. For example, in [22], the goal of their neural network is to *learn* a controller during training, whereas we *impose* a controller in the designed DSRNN to stabilize the hidden state trajectory through eigenvalue placement assignment of the linearized system.

The DSRNN is validated on an experimental dataset, obtained from [23], of a recorded double pendulum system, as well as a classification task on two Lorenz oscillators, and is compared against the LSTM and vanilla RNN. These are chaotic systems, making them highly nonlinear and sensitive to initial conditions, and therefor are difficult for RNNs to model due to long-term unpredictability. As pointed out in [24], deep RNNs are likely to either converge prematurely or overfit when modeling a chaotic system. The results show that the DSRNN can predict the future trajectory of the pendulum with errors orders of magnitude less than both LSTM and vanilla RNN. We show that it reduces the mean squared error of the LSTM on a test set by up to ~99.639%. We also show that the DSRNN achieves comparable performance to the LSTM and vanilla RNN on the Lorenz system classification task.

Contributions The major contributions of this paper are as follows:

- *Development of learnable coefficients for hidden state connections*: This concept is introduced and validated over a number of previous time-steps.
- *Formulation of a nonlinear state-space representation of the hidden states of the recurrent network*: The hidden states of the network are treated as state variables in a novel state-space representation of the network.
- *Stabilization of hidden state trajectory*: The eigenvalue placement of the linearized system is introduced as a regularization term to the overall cost function, compelling the eigenvalues to converge to their desired positions during training.

The rest of the manuscript is organized as follows. Section 2 presents the proposed approach and architecture of the DSRNN. Furthermore, the state-space representation of the hidden state trajectory is formulated, along with the regularizer used to stabilize the model, and the backpropagation gradients of the DSRNN is also derived. Section 3 presents the experimental study performed to analyze the DSRNN performance. Finally, conclusions and final remarks are drawn in Sect. 4.

2 Problem Formulation and Preliminaries

To improve and control the flow of data through an RNN, we introduce the DSRNN architecture which contains a parameter $k \in \mathbb{N}$ that refers to the number of *weighted* skip connections across time-steps, in the following manner:

$$h_t = \sum_{i=1}^{k} \left\{ \alpha_i h_{t-i} \right\} + \varphi_h(h_{t-1}, x_t) \tag{1}$$

where $\alpha_i \in \mathbb{R}^{n \times n}, \forall i = 1, \ldots, k$ is a diagonal matrix with diagonal entries $\alpha_i^{(j)}, \forall j = 1, \ldots, n$, and $\varphi_h(\cdot)$ is the activation function. The matrices α_i are referred to as the skip-coefficients, $h_t \in \mathbb{R}^n$ are the hidden states of the network, and $x_t \in \mathbb{R}^n$ is the input. Notice that if we set $k = 0$, the system reduces to a vanilla RNN. Whereas for $k = 1$ and $\alpha_1 = \mathbb{1}$, the system reduces to the constant error carousel structure in the original LSTM paper [2]. It should also be noted that the forget gate was added to the LSTM later [3], effective by making $\alpha_1 = f_t$, where f_t is the forget gate value.

The skip-coefficients are set to be learnable parameters to allow the network to stabilize the hidden states of the DSRNN. To ensure such control, we introduce a regularizer that is added to the loss-function to update the skip-coefficients such that the states of the RNN never diverge. Thus, we create a linearized state-space representation of the network and build the regularizer to ensure that the absolute value of the eigenvalues of the linearized state-space model are inside the unit circle. The following subsection will formulate the state-space representation of the hidden states of the DSRNN.

2.1 State-Space Representation of DSRNN Architecture

With the architecture described above, we define the state-vectors as connected hidden states, in the following manner $q_t^i := h_{t-i}, \forall i = 1, \dots, k$. The resulting state vector can therefore be written as $q_t \equiv [q_t^1, \dots, q_t^k]^T$. Thus, the state-space formulation of the the hidden states becomes:

$$\begin{bmatrix} q_{t+1}^1 \\ q_{t+1}^2 \\ q_{t+1}^3 \\ \vdots \\ q_{t+1}^k \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \dots & \alpha_k \\ \mathbb{1} & \mathbb{0} & \mathbb{0} & \dots & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} & \dots & \mathbb{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbb{0} & \mathbb{0} & \dots & \mathbb{1} & \mathbb{0} \end{bmatrix} \begin{bmatrix} q_t^1 \\ q_t^2 \\ q_t^3 \\ \vdots \\ q_t^k \end{bmatrix} + \begin{bmatrix} \varphi_h(q_t^1, x_t) \\ \mathbb{0} \\ \mathbb{0} \\ \vdots \\ \mathbb{0} \end{bmatrix} \tag{2}$$

where $\mathbb{0}$ and $\mathbb{1}$ are the $n \times n$ zero and identity matrices, respectively. For simplicity, the state vectors can be re-written as $q_{t+1} \triangleq f(q_t, x_t)\ \forall t$. The system in Eq. (2) is a nonlinear time-varying dynamical system, which makes it difficult to figure out whether or not the states q_t^i for $i = 1, \dots, k$ will eventually diverge.

In order to study the nonlinear system's states, one may use Lyapunov's Indirect Method to draw conclusions about stability in local regions around an equilibrium pair, (q^e, x^e). A state q_t and input x_t are said to be an equilibrium pair if the state q_t remains constant in the future (i.e. $q^e = f(q^e, x^e)\ \forall t$) provided that there are no exogenous disturbances. Lyapunov's Indirect Method considers the nonlinear system $q_{t+1} = f(q_t)$, where $f(\cdot) : D \to \mathbb{R}^n$ is continuously differentiable and D is a neighborhood of the equilibrium point, q^e, which for simplicity will be assumed to be the origin. Using the mean value theorem, and for some point $z \in D$, where if the line connecting z to the equilibrium point is entirely inside D, then for each element $i \in \{1, ..., n\}$ of f it follows that

$$f_i(q) = \frac{\partial f_i}{\partial q}(z_i)q = \frac{\partial f_i}{\partial q}(0)q + \left[\frac{\partial f_i}{\partial q}(z_i) - \frac{\partial f_i}{\partial q}(0)\right]q$$

where $g_i(q) \triangleq \left[\frac{\partial f_i}{\partial q}(z_i) - \frac{\partial f_i}{\partial q}(0)\right]q$ satisfies

$$|g_i(q)| \le \left\| \frac{\partial f_i}{\partial q}(z_i) - \frac{\partial f_i}{\partial q}(0) \right\| \|q\|$$

Therefore, $f(q) = \bar{A}q + g(q)$, where $\bar{A} = \frac{\partial f_i}{\partial q}(0)$. Since the gradient $\frac{\partial f_i}{\partial q}$ is also continuous, then $\frac{\|g(q)\|}{\|q\|} \to 0$ as $\|q\| \to 0$ [25]. Therefore, inside a neighborhood, or region, around the equilibrium point, the nonlinear system can be approximated by $q_{t+1} = \bar{A}q_t$. This can be summarized in the following theorem:

Theorem 1 [25] *Let $q^e = 0$ be an equilibrium point for the nonlinear system $q_{t+1} = f(q_t)$, where $f(\cdot) : D \to \mathbb{R}^n$ is continuously differentiable and D is a neighborhood of the equilibrium point. Let the Jacobian matrix $\bar{A} = \frac{\partial f_i}{\partial q}(q)\big|_{q=q^e}$. Then, the linearized system $q_{t+1} = \bar{A}q_t$ is asymptotically stable if each eigenvalue of $\bar{A}$ is inside the unit circle, or $|\lambda_i| < 1$ for all i.*

In the case of neural networks, the equilibrium points will vary according to the chosen activation function. The activation function chosen however is assumed to be continuously differentiable and locally Lipschitz. For example, the hyperbolic tangent function, which is both continuously differentiable and locally Lipschitz, has an equilibrium point at the origin. We therefore evaluate the stability of the network using Lyapunov's indirect method above, where the equilibrium point is an equilibrium pair (q^e, x^e). Therefore, the linearized system is found by taking the Jacobian of the functions $f(q_t, x_t)$ with respect to the states and inputs of the system evaluated at the equilibrium points, as such:

$$q_{t+1} = \nabla_q f(q_t, x_t)\Big|_{\substack{q_t=q^e\\x_t=x^e}} q_t + \nabla_x f(q_t, x_t)\Big|_{\substack{q_t=q^e\\x_t=x^e}} x_t + h.o.t. \tag{3}$$

where $h.o.t.$ are the higher order terms of the true nonlinear system. In matrix form, after neglecting the higher order terms, the linearized system is written as:

$$q_{t+1} \approx \underbrace{\begin{bmatrix} \alpha_1 + \frac{\partial\varphi(\cdot)}{\partial q_t^1} & \alpha_2 & \alpha_3 & \dots & \alpha_k \\ \mathbb{1} & \mathbb{0} & \mathbb{0} & \dots & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} & \dots & \mathbb{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbb{0} & \dots & \mathbb{0} & \mathbb{1} & \mathbb{0} \end{bmatrix}}_{A_{(n\cdot k\times n\cdot k)}} q_t + \underbrace{\begin{bmatrix} \frac{\partial\varphi(\cdot)}{\partial x_t} \\ \mathbb{0} \\ \vdots \\ \mathbb{0} \\ \mathbb{0} \end{bmatrix}}_{B_{(n\cdot k\times 1)}} x_t \tag{4}$$

where A and B are evaluated at the equilibrium points. In our experiments, the activation function chosen is the hyperbolic tangent, meaning the equilibrium pair is located at the origin, i.e., $(q^e, x^e) = (0, 0)$, and $\frac{\partial\varphi(\cdot)}{\partial q_t^1}\Big|_{\substack{q^e=0\\x^e=0}} = W_{rec}$, where W_{rec} is the recurrent weight matrix.

From Theorem 1, it is possible to determine whether the states q_t^j will converge or not $\forall j = 1, 2, \dots, k$, by studying the eigenvalues, $\lambda(A)$, of the state-transition matrix A in equation (4) above. Since the system is discrete, to ensure convergence of the states, we require $|\lambda_i^*(A)| < 1, \forall i = 1, \dots, n \cdot k$, where $\lambda_i^*(A)$ is the ith desired eigenvalue of A.

2.2 Regularizer for Desired Eigenvalues

Since the linearized model of the system is updated at every training iteration, the skip-coefficients constantly need to be adjusted in order to maintain all desired eigenvalues, λ_i^* for $i = 1, \dots, n \cdot k$, of the system. For this reason, a regularizer is designed and embedded into the loss function of the DSRNN, which is built to enforce eigenvalue placement. The regularizer is a function of both the skip-coefficients as well as the desired eigenvalues. With this formulation, all parameters are updated such that the eigenvalues $\lambda(A)$ converge towards $\lambda^*(A)$, as well as minimize the objective error of the network.

The eigenvalue regularizer is designed to minimize the distance between λ_i^* for $i = 1, \dots, n \cdot k$ and the eigenvalues of A in (4) as a function of the skip-coefficients, $\lambda(\alpha)$, by using the Euclidean norm $C_{\lambda}(\lambda^{*}, \lambda(\alpha)) = \sqrt{\sum_{i=1}^{n \cdot k} \left(\lambda_i^{*} - \lambda_i(\alpha)\right)^2}$. The cost-function (or loss) of the DSRNN, $C_{DSRNN}(\cdot)$, is therefore extended, and the resulting cost function becomes $C_{DSRNN}(\cdot) = C_{NN}(\cdot) + \beta \cdot C_{\lambda}(\cdot)$, where β is a constant regularizer parameter and can be simply chosen to be 1 as in our experiments, and $C_{NN}(\cdot)$ is a cost function between the output of the DSRNN and the ground-truth, which in our experiments is chosen to be the L_2 norm. We note that the complexity for computing the eigenvalues of a square $m \times m$ matrix increases by the order of $\mathcal{O}(m^2)$.

Remark 1 The skip-coefficients and all parameters of the DSRNN are updated simultaneously, which indicates that the latest skip-coefficients no longer yield the desired eigenvalues of the new linearized model. However, this does not pose an alarming problem if the learning rate is small enough. This is because the parameters will be updated slow enough such that $\lambda(A^{(t+1)})$ are in a region close enough to $\lambda(A^{(t)})$, where the superscript (t) refers to the training iteration. □

Remark 2 The closer the eigenvalues of A are chosen to the origin, the faster the states converge towards the equilibrium points, whereas eigenvalues chosen very close to 1 require a large number of time-steps for h_t to converge. Hence, intuitively one would choose a small desired eigenvalue, $0 < |\lambda^{*}(A)| \ll 1$, to ensure a more stable trajectory of hidden states, which effectively allows the RNN to better learn. □

Remark 3 Although there is no straightforward way of knowing whether the values of the hidden states are within that local region D around the equilibrium pair, it is likely that the initial training iterations return hidden state values that are close enough to the equilibrium pair. This assumes that the equilibrium point is located at the origin (as in the case of the hyperbolic tangent function), initializing all the parameters of the network using a normal distribution, and scaling down or normalizing the input data in order to return small hidden states values in the initial training iterations, and consequently within reasonable proximity of D. Assumptions can also be made about the step-size chosen for the optimizer, or the value of the regularizer parameter, to ensure that the weights are modified to reduce the hidden state values till they enter D. Therefore, without loss of generality, we assume that the operation is inside the region, D. □

2.3 Backpropagation Through Time with DSRNN

Back-propagation through time (BPTT) is well understood and established [26], especially for a standard RNN with no skip connections. However, when skip-connections are introduced into the architecture, as in the DSRNN, the derivation of the BPTT becomes more complex to expand as a function of time. This is because the chain-rule adapted in the standard BPTT needs to be reapplied to all k previously connected hidden states. In equation form, by letting $\Delta h_i^j \equiv \frac{\partial h_{t-j}}{\partial h_{t-i}}$, the back-propagation expands to include multiple nested summation terms which have the following structure:

$$\frac{\partial \mathcal{E}_t}{\partial h_{t-T}} = \frac{\partial \mathcal{E}_t}{\partial h_t}\left(\sum_{i_1=1}^{k} \Delta h_{i_1}^{0}\left(\sum_{i_2=1}^{k} \Delta h_{i_1+i_2}^{i_1}\right)\right) \tag{5}$$

$$= \frac{\partial \mathcal{E}_t}{\partial h_t}\left(\sum_{i_1=1}^{k} \Delta h_{i_1}^{0}\left(\dots\left(\sum_{i_T=1}^{k} \Delta h_{i_1+\ldots+i_T}^{i_1+\ldots+i_{T-1}}\right)\right)\right) \tag{6}$$

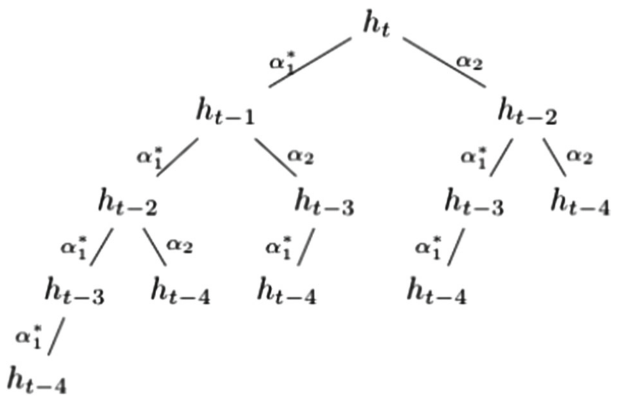

Fig. 1 Tree diagram of to expand the BPTT for DSRNN with $k = 2$ and $T = 4$

where $T \geq k$ and $\mathcal{E}_t$ is the error of the RNN at time-step t. Once the equation is expanded to the Tth time-step, the following substitutions are recommended: $\frac{\partial h_t}{\partial h_{t-1}} = \alpha_1 + W_{rec}^T diag\big(\varphi^{'}(h_{t-1})\big)$ and $\frac{\partial h_t}{\partial h_{t-i}} = \alpha_i$ for $1 < i \leq k$.

We present two examples to illustrate BPTT for the DSRNN for the cases where $k = 1$ and 2. For the $k = 1$ case, we show the effect of the BPTT on the gradient by borrowing notations from the proof in [27], as follows:

We know that the Jacobian matrix $\frac{\partial h_t}{\partial h_{t-1}} = \alpha_1 + W_{rec}^T diag\big(\varphi^{'}(h_{t-1})\big)$. Let $\Lambda_w = \frac{1}{\gamma}$ be the largest singular value of the recurrent weight matrix W_{rec}, and Λ_α be the largest singular value of the skip-coefficient α_1. Then $\forall t$:

$$\left\| \frac{\partial h_{t+1}}{\partial h_t} \right\| \leq \|\alpha_1\| + \left\| W_{rec}^T \right\| \left\| diag\big(\varphi^{'}(h_t)\big) \right\| \leq \Lambda_\alpha + \frac{1}{\gamma}\gamma \tag{7}$$

Let $\eta \in \mathbb{R}$ be such that $\left\| W_{rec}^T \right\| \left\| diag\big(\varphi^{'}(h_t)\big) \right\| = \eta$. Then it can be shown that

$$\left\| \frac{\partial \mathcal{E}_t}{\partial h_t} \left(\prod_{i=T}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} \right) \right\| \leq (\Lambda_\alpha + \eta)^{t-T} \left\| \frac{\partial \mathcal{E}_t}{\partial h_t} \right\| \tag{8}$$

For the $k = 2$ case, we give an example of BPTT for $T = 6$. Using equation (6) and the recommended substitutions, and defining $\alpha_1 + W_{rec}^T diag\big(\varphi^{'}(h_{t-1})\big) \triangleq \alpha_1^*$, the example unfolds as follows:

$$\frac{\partial \mathcal{E}_t}{\partial h_{t-6}} = \frac{\partial \mathcal{E}_t}{\partial h_t}\Big(\Delta h_1^0 + \Delta h_2^0\Big) \tag{9}$$

$$= \frac{\partial \mathcal{E}_t}{\partial h_t}\Big(\Delta h_1^0\Big(\Delta h_2^1 + \Delta h_3^1\Big) + \Delta h_2^0\Big(\Delta h_3^2 + \Delta h_4^2\Big)\Big) \tag{10}$$

$$= (\alpha_1^*)^6 + 5(\alpha_1^*)^4(\alpha_2)^1 + 6(\alpha_1^*)^2(\alpha_2)^2 + (\alpha_2)^3 \tag{11}$$

We present methods using combinatorics to simplify the calculations relating to the expansion of the BPTT. Due to space constraints, we will show the example of $k = 2$ and $T = 4$ here; this can be easily generalized to multiple time-steps and different k values. For this case, the DSRNN has the following structure: $h_t = \alpha_1 h_{t-1} + \alpha_2 h_{t-2} + \varphi_h(h_{t-1}, x_t)$. We can draw the tree shown in Fig. 1.

Starting at h_t, where $h_t = f(h_{t-1}, h_{t-2})$, the error being backpropagated will split to both h_{t-1} (left child) and h_{t-2} (right child), with the edge of the tree denoting the respective partial derivatives. The error can be traced back through several branches until h_{t-4} is reached. By summing up all the branches we get the total error propagated to $T = 4$ time-steps back. $\frac{\partial \mathcal{E}_t}{\partial h_{t-4}} = (\alpha_1^*)^4 + 3(\alpha_1^*)^2(\alpha_2)^1 + (\alpha_2)^2$. Note, the value of k dictates the number of branches each edge splits into, meaning for $k = 1$ trees or $k = 0$ trees (vanilla RNN), they will only have the left child because the current hidden state only depends on its immediate previous hidden state, i.e. $h_t = f(h_{t-1})$.

For the case ($k = 2$ and $T = 4$), we can alternatively treat this problem as attempting to go back 4 time-steps in total, going back 1 step ($\partial h_t / \partial h_{t-1}$) or 2 steps ($\partial h_t / \partial h_{t-2}$) at a time. This becomes a problem of finding the pair of natural numbers (i, j) such that

$$T = i \times 1 + j \times 2 \Big|_{i,j \in \mathbb{N}} \tag{12}$$

If $T = 4$, then we have $4 = 4 \times 1 = 2 \times 1 + 1 \times 2 = 2 \times 2$, or $(i, j) \in ((4, 0), (2, 1), (0, 2))$. Looking back at the tree graph, since going left is 1 step and going right is 2 steps, we can take 4 left steps, or 2 left steps and 1 right step, or 2 right steps to reach h_{t-4}. Once we've settled for each pair of (i, j) values, the associated term is given by

$$\binom{i+j}{i} \left(\frac{\partial h_t}{\partial h_{t-1}} \right)^i \cdot \left(\frac{\partial h_t}{\partial h_{t-2}} \right)^j \tag{13}$$

For example, when $(i, j) = (2, 1)$, by using combinatorics we can easily determine that coefficient for the term $(\alpha_1^*)^2(\alpha_2)^1$ is $\binom{2+1}{2} = 3$ (basically we are taking 2 left children and 1 right child from the tree, and we need to choose the locations of the two left children among all three children).

Then for the derivation of $\frac{\partial \mathcal{E}_t}{\partial h_{t-6}}$ for a DSRNN of parameter $k = 2$, we can write the BPTT as follows, without having to expand every term

$$\begin{aligned} \frac{\partial \mathcal{E}_t}{\partial h_{t-6}} &= \binom{6}{6} \left(\frac{\partial h_t}{\partial h_{t-1}} \right)^6 + \binom{5}{4} \left(\frac{\partial h_t}{\partial h_{t-1}} \right)^4 \left(\frac{\partial h_t}{\partial h_{t-2}} \right)^1 \\ &\quad + \binom{4}{2} \left(\frac{\partial h_t}{\partial h_{t-1}} \right)^2 \left(\frac{\partial h_t}{\partial h_{t-2}} \right)^2 + \binom{3}{3} \left(\frac{\partial h_t}{\partial h_{t-2}} \right)^3 \\ &= (\alpha_1^*)^6 + 5(\alpha_1^*)^4(\alpha_2)^1 + 6(\alpha_1^*)^2(\alpha_2)^2 + (\alpha_2)^3 \end{aligned} \tag{14}$$

This can also be generalized to larger values of k.

3 Simulations and Experimental Study

In this section, we test the performance of the DSRNN on a forecasting and a classification task. We compare the results of the DSRNN with a vanilla RNN and an LSTM. We choose to use two chaotic systems for our experiments, a double pendulum for the forecasting task and a Lorenz oscillator for the classification task. Since chaotic systems are extremely sensitive to initial conditions and are unpredictable in the long-term [28], the networks need to maintain stable hidden state trajectories to model such long-term unpredictability. The double pendulum dataset is obtained from [23], where the networks must forecast the positions of the two dynamic joints of the double pendulum system. The Lorenz systems are simulated, and the objective is to map short segments of the state signals to their corresponding oscillator system that produced them. The final loss and the mean and standard deviations of the test errors are used to compare the networks against one another.

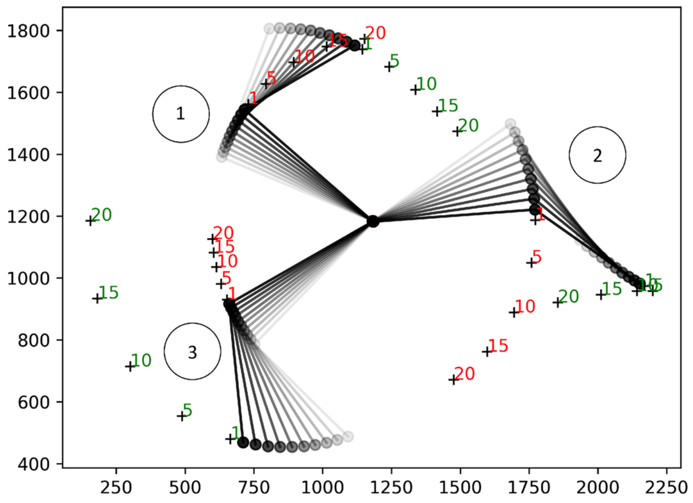

Fig. 2 A visualization of the double pendulum joints' trajectories at three random instances, each with a time-series length of 10 steps. Each joint is represented by a black circle, where the direction of travel is illustrated as travelling from the lightly shaded joints to the darker ones. The five black cross-markers, shown after each moving joint, represent the true positions of the joints $T \in \{1, 5, 10, 15, 20\}$ time-steps ahead. The value of each T is marked, where the red numbers correspond to the second (middle) joint, and the green numbers correspond to the third (last) joint

Model Parameters: The networks are chosen after a basic hyperparameter search, where the batch size, learning rate, and number of LSTM blocks are varied. Batch sizes of [100, 250, 500, 1000], learning rates of [0.1, 0.01, 0.001, 0.0001], and LSTMs with 1 and 2 stacked layers are all considered. All networks contain 128 hidden units and the weight matrices of the DSRNN models are initialized with Glorot initializers [29]. All networks use the Adam optimizer [30] and their gradients clipped to 5 [27]. All models are implemented in Tensorflow [31]. Our open-source implementation is publicly available at: https://github.com/samersaabjr/DSRNN.

3.1 Forecasting the Double Pendulum

The double pendulum (DP) is a simple yet dramatically chaotic dynamical system, consisting of one pendulum attached to the end of another. The dataset used in our experiments can be found in [23], where 21 videos of a double pendulum are recorded, and positions of the three joints are extracted and provided for all frames. In order to test the networks' ability to model the DP, the training set is constructed using the first 20 videos of the dataset, and the test set contains data extracted from the last video, to ensure that the testing data has different initial conditions from the training data. The input into the network are trajectories of the x and y positions of the three joints of the DP with a time-series length of 10 time-steps. The outputs fed into the network are the x and y positions of the two dynamic joints $T \in \{1, 5, 10, 15, 20\}$ time-steps into the future from the last time-step of the input trajectory. Figure 2 shows an example of three distinct input trajectories, along with their corresponding outputs to the RNN models.

For each network, 5 independent runs are computed, where the average results over the 5 trained models are presented. All the networks use a batch size of 250. The DSRNN models

Table 1 Final test loss results for double pendulum forecasting

T	$k=1$	$k=2$	$k=3$	LSTM	RNN
1	$1.03e5$	$7.88e4$	**7.16e4**	$1.98e7$	$2.75e8$
5	$1.18e6$	**8.52e5**	$9.14e5$	$3.57e7$	$3.00e8$
10	$5.89e6$	$5.22e6$	**5.15e6**	$1.07e8$	$2.76e8$
15	$1.62e7$	$1.53e7$	**1.52e7**	$7.43e7$	$3.00e8$
20	**2.94e8**	$3.29e7$	$3.29e7$	$1.06e8$	$2.98e8$

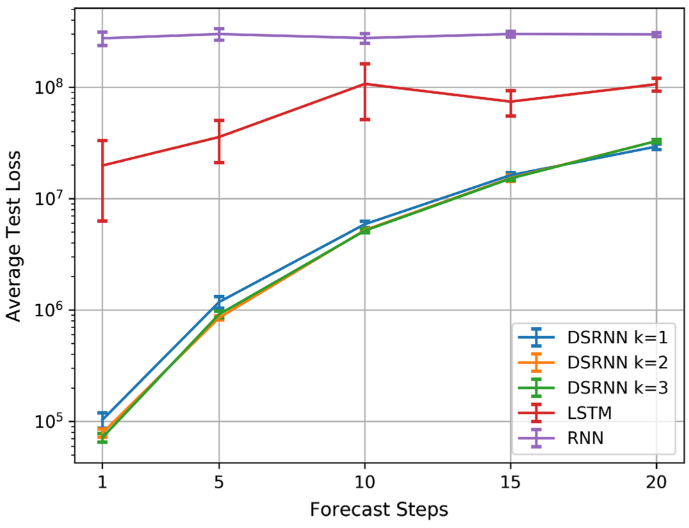

Fig. 3 Average test errors over the 5 independent runs returned by each network model as the number of steps to forecast is increased, with the standard deviations across each 5 network models indicated by the error bars

have a learning rate of 0.0001. The LSTM and vanilla RNN use a learning rate of 0.001, and the LSTM utilizes 2 stacked layers.

The final test losses of the recurrent models are summarized in Table 1 and Fig. 3. The lowest test losses are shown in bold text. The test losses are calculated using half the L_2-norm (without the square-root). It is clear that the DSRNN's outperform the LSTM and Vanilla RNN, where the DSRNN with $k = 3$ achieves the most dominant performance. The $k = 3$ DSRNN reduces the test error of the LSTM by up to 99.639% for forecasting $T = 1$ time-step ahead, and as low as 69.107% reduction in error for the $T = 20$ case. As for the RNN, the relative test error is reduced by up to 99.974% for $T = 1$, and as low as 88.961% reduction in error for $T = 20$.

The p-value is calculated in order to show whether or not there are significant statistical differences between the DSRNN models and both the LSTM and vanilla RNN, where every model was run over 5 random seeds. Since all three DSRNN models performed within close proximity of one another, the LSTM and vanilla RNN are compared with the DSRNN where $k = 3$. The t-test was used to compare the means of the networks, as shown in Table 2.

Figure 4 shows a visual spot of the likely regions that each recurrent network model would predict. The regions are centered around the true prediction plus the average error around each joint at $T \in \{1, 5, 10, 15, 20\}$ time-step prediction. The height and width of the circular regions are proportional in magnitude to 1 standard deviation of the x and y error predictions.

Table 2 Statistical comparisons between the DSRNN $k = 3$ and both the LSTM and vanilla RNN via p-value using the t-test

Model	T=1	T=5	T=10	T=15	T=20
LSTM	0.0193	0.0014	0.0065	0.0003	5.81e−6
RNN	4.75e−7	1.50e−7	4.62e−8	8.07e−10	7.20e−11

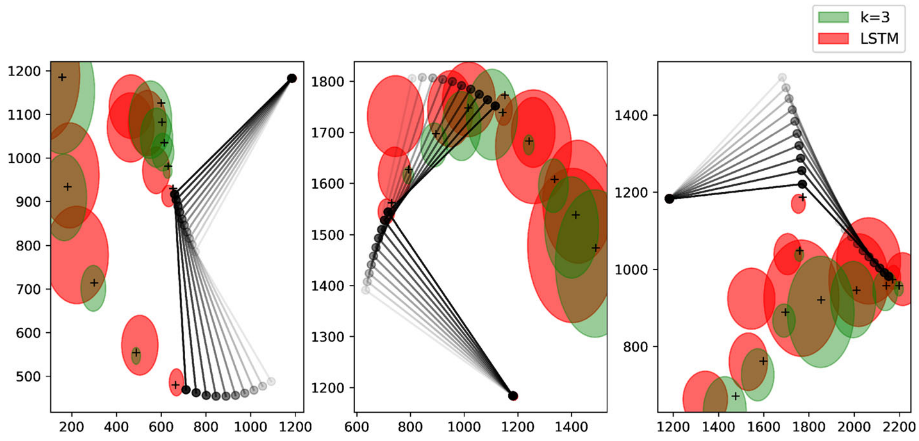

Fig. 4 Visualization of the predicted locations of the dynamic joints of the double pendulum at three different instances by the DSRNN with $k = 3$ and the LSTM. The highlighted regions corresponding to each recurrent network architecture are centered around the average prediction for each joint, corresponding to every $T \in \{1, 5, 10, 15, 20\}$ time-step predictions. The height and width of the circular regions are proportional in magnitude to 1 standard deviation of the x and y error predictions

Clearly, the vanilla RNN performs the worst in this task. The DSRNN however performs the best in all cases, with the $k = 3$ architecture performing the best.

The results from forecasting the double pendulum suggests that the LSTM, although extremely powerful when modeling systems with an internal memory, has significant difficulty in modeling this chaotic system, as its structure does not lend itself well to this task. Conversely, the DSRNN is capable of modeling such a system with long-term unpredictability.

3.2 Lorenz Oscillator Classification

The Lorenz system is a chaotic dynamical system governed by the following three coupled differential equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z. \end{aligned} \tag{15}$$

Table 3 The average test accuracy results obtained over the last 10 epochs for all networks

T	DSRNN $k = 1$ (%)	DSRNN $k = 2$ (%)	DSRNN $k = 3$ (%)	LSTM (%)	Vanilla RNN (%)
5	57.49	57.41	56.93	68.15	67.73
	±0.004	±0.002	±0.002	±0.007	±0.006
10	69.04	69.06	67.59	69.44	66.95
	±0.007	±0.007	±0.005	±0.003	±0.005
15	**71.23**	70.22	68.99	69.43	69.08
	±**0.007**	±0.005	±0.006	±0.006	±0.004
20	69.12	69.20	67.95	69.41	68.85
	±0.007	±0.005	±0.008	±0.006	±0.005

For this experiment, we created two Lorenz systems with parameters (a) $\sigma = 10, \rho = 28, \beta = 8/3$ and (b) $\sigma = 11, \rho = 29, \beta = 3$ for classification. The samples from both classes were generated with the same initial conditions of $x = 1, y = 1, z = 1$ and integration step of $\Delta t = 0.01$. The lengths of the time-steps T consist of these following values: $T \in \{10, 20, 50, 100, 200, 500\}$. Starting at the initial conditions, T time-steps of $\{x, y, z\}$ values are generated as one sample in the dataset. Subsequently, we repeat 20, 000 times for each T value to get 10,000 training data (randomly chosen from all samples) and testing data.

For each network, 10 independent runs are computed, where the average results over the 10 trained models are presented. All the DSRNN models use a batch size of 500, and the LSTM and vanilla RNN use a batch size of 250. The DSRNN models have a learning rate of 0.0001. The LSTM and vanilla RNN use a learning rate of 0.001, and the LSTM utilizes 2 stacked layers.

Table 3 lists the test accuracies for different values of k parameter in the DSRNN as well as for the LSTM and vanilla RNN. The results are also visualized in Fig. 5. For this task, the LSTM seems to perform the best overall, however for the $T = 15$ case, we find that the DSRNN with both, $k = 1$ and $k = 2$, outperforms the LSTM and vanilla RNN. The best performance achieved for this task is $71.23\% \pm 0.007\%$ when using the DSRNN with $k = 1$, and is highlighted in bold in the table. Otherwise, the performance of the DSRNN is comparable to the well-tuned LSTM network, which is understandable since, as previously discussed in Sect. 2, an LSTM is a DSRNN with $k = 1$ network with additional gating functions. It is also interesting to note that with smaller sequence length inputs, the DSRNN performs significantly worse, but as the sequence length grows, it performs as well, if not better, than the other networks. Since the performance does not deteriorate, this could be an indication that the DSRNN is robust to long inputs that could possess long-term dependencies. We note that the LSTM converges faster than the DSRNN, as shown in Fig. 5, however the final steady-state performance is very similar. This experiment shows that the DSRNN is well suited for modeling nonlinear systems.

Taking a closer look at just the DSRNN results, we find that the DSRNN with $k = 1$ yields the best overall performance for this classification task. It is consistent with the Lorenz System equations (15) since the current state of the Lorenz states only depends on the one time-step previous states. In other words, the Lorenz system is a first-order dynamical equation, and whenever k is chosen such that it matches the order of the system being learned, one can expect better results. Enforcing additional structures on the hidden states by adding hidden states further back in history would lead to worse results.

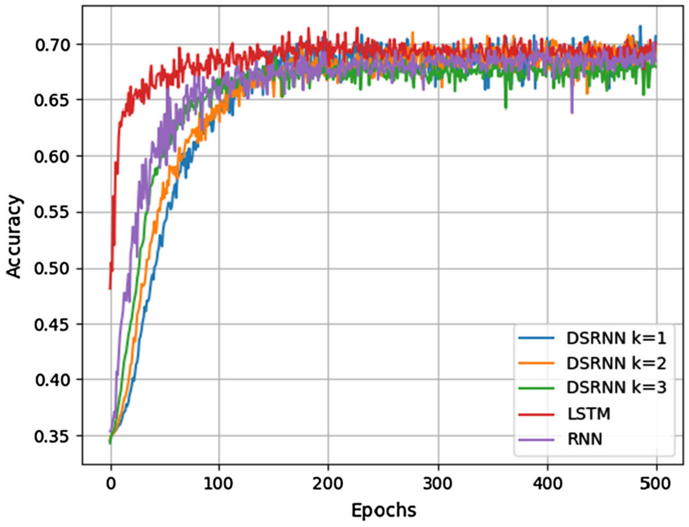

Fig. 5 The average test accuracy over 10 random initializations of the network parameters at every epoch for the DSRNN models, LSTM, and vanilla RNN for the $T = 10$ time-steps case

4 Summary, Conclusions, and Future Work

This paper has developed a new RNN architecture, called the Dynamically Stabilized Recurrent Neural Network (DSRNN), which is designed to stabilize the recurrent hidden-state trajectory. The DSRNN is a generalization of the vanilla RNN, where weighted skip connections across k previous hidden states are utilized. From the perspectives of dynamical systems and control theory, the notion of Lyapunov stability is introduced as an eigenvalue regularization term in the overall loss function, which imposes a controller for the hidden state trajectory. Experimental results show that the DSRNN outperforms both the LSTM and vanilla RNN in forecasting the chaotic dynamical double pendulum system, by a substantial margin. Specifically, the DSRNN reduces the relative mean squared errors of both the LSTM and vanilla RNN by up to 99.639% and 99.974%, respectively. The performance of the DSRNN is also validated on a classification task of two Lorenz oscillators, where it was shown that the DSRNN performed comparably well to the LSTM. In fact, the DSRNN outperforms the LSTM in the case where the input sequence length was $T = 15$ time-steps. The DSRNN greatly outperforms the LSTM in the double pendulum. The DSRNN also achieves the best test accuracy for the Lorenz system for input sequence lengths of $T = 15$, but is slightly outperformed by the LSTM for the remaining input sequence lengths. This makes sense when dissecting the nature of the two experiments. In the double pendulum case, the data is obtained using various initial conditions, which result in widely different trajectories. Whereas in the classification task, the initial conditions of the two systems are fixed, and only snippets of the trajectory are used as input. Thus, the Lorenz classification problem can more easily be solved using memory than the double pendulum forecasting task, since the sampled trajectories appear very random in comparison to each other, and so a memory unit will not serve much help. Therefore, although the LSTM lends itself better to memory-based tasks, the memory structure in the LSTM may be its own Achilles' heel when modeling chaotic dynamical systems, because trying to remember trajectories without learning the dynamical structure is ineffective. The DSRNN however does not possess a memory structure, but the learnable skip-coefficients could be adjusted to better learn temporal dependencies with k time delays. The latter is due to the fact that the weight matrices are shared through time.

Future extensions of this work envision capitalizing on layer-wise smoothness by adding (spatial) skip-connections across the hidden layers of the recurrent network. Such an architecture may allow the network to distinguish temporal dynamics from spatial ones, giving the network flexibility in modeling many complex nonlinear dynamical systems.

References

1. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
2. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
3. Gers FA, Schmidhuber J, Cummins F (1999) Learning to forget: continual prediction with lstm
4. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2016) Lstm: a search space odyssey. IEEE Trans Neural Netw Learn Syst 28(10):2222–2232
5. Lippi M, Montemurro MA, Degli Esposti M, Cristadoro G (2019) Natural language statistical features of lstm-generated texts. IEEE Trans Neural Netw Learn Syst 30(11):3326–3337
6. Yu X, Wu L, Xu C, Hu Y, Ma C (2019) A novel neural network for solving nonsmooth nonconvex optimization problems. IEEE Trans Neural Netw Learn Syst
7. Qin S, Xue X (2014) A two-layer recurrent neural network for nonsmooth convex optimization problems. IEEE Trans Neural Netw Learn Syst 26(6):1149–1160
8. Che H, Wang J (2018) A two-timescale duplex neurodynamic approach to biconvex optimization. IEEE Trans Neural Netw Learn Syst 30(8):2503–2514
9. Chen K, Yao L, Zhang D, Wang X, Chang X, Nie F (2019) A semisupervised recurrent convolutional attention model for human activity recognition. IEEE Trans Neural Netw Learn Syst
10. Lipton ZC, Kale D, Wetzel R (2016) Directly modeling missing data in sequences with rnns: improved classification of clinical time series. In: Machine learning for healthcare conference, pp 253–270
11. Miller J, Hardt M (2018) Stable recurrent models. *arXiv preprint*, arXiv:1805.10369
12. Bao G, Peng Y, Zhou X, Gong S (2020) Region stability and stabilization of recurrent neural network with parameter disturbances. Neural Process Lett 52(3):2175–2188
13. Chandran R, Balasubramaniam P (2013) Delay dependent exponential stability for fuzzy recurrent neural networks with interval time-varying delay. Neural Process Lett 37(2):147–161
14. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv preprint, arXiv:1607.06450
15. Bengio Y, Boulanger-Lewandowski N, Pascanu R (2013) Advances in optimizing recurrent networks. In: 2013 IEEE international conference on acoustics, speech and signal processing, pp 8624–8628, IEEE
16. Jaeger H, Lukoševičius M, Popovici D, Siewert U (2007) Optimization and applications of echo state networks with leaky-integrator neurons. Neural Netw 20(3):335–352
17. Zhang Y, Chen G, Yu D, Yaco K, Khudanpur S, Glass J (2016) Highway long short-term memory rnns for distant speech recognition. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp 5755–5759, IEEE
18. Kim J, El-Khamy M, Lee J (2017) Residual lstm: design of a deep recurrent architecture for distant speech recognition. arXiv preprint, arXiv:1701.03360
19. Haviv D, Rivkind A, Barak O (2019) Understanding and controlling memory in recurrent neural networks. In: International conference on machine learning, pp 2663–2671, PMLR
20. Kalman RE, Bertram JE (1960) Control system analysis and design via the “second method" of lyapunov: I—continuous-time systems. J Basic Eng 82(2):371–393
21. Hauser M, Gunn S, Saab S Jr, Ray A (2019) State-space representations of deep neural networks. Neural Comput 31(3):538–554
22. Tay Y, Luu AT, Hui SC (2018) Recurrently controlled recurrent networks. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds) Advances in neural information processing systems 31, pp 4736–4748, Curran Associates, Inc
23. Asseman A, Kornuta T, Ozcan A (2018) Learning beyond simulated physics
24. de Jesús Serrano-Pérez J, Fernández-Anaya G, Carrillo-Moreno S, Yu W (2021) New results for prediction of chaotic systems using deep recurrent neural networks. Neural Process Lett, pp 1–18
25. Bof N, Carli R, Schenato L (2018) Lyapunov theory for discrete time systems. arXiv preprint, arXiv:1809.05289
26. Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560

27. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: International conference on machine learning, pp 1310–1318
28. Ouyang X, Luo Y, Liu J, Liu Y, Bi J, Qiu S (2018) Period analysis of chaotic systems under finite precisions. In: 2018 26th International conference on systems engineering (ICSEng), pp 1–5, IEEE
29. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp 249–256
30. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980
31. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2016) Tensorflow: a system for large-scale machine learning. OSDI 16:265–283