arXiv:1910.06251v3 [cs.CV] 9 Dec 2020

Deep Independently Recurrent Neural Network (IndRNN)

Shuai Li, Wanqing Li, *Senior Member, IEEE,* Chris Cook, Yanbo Gao

Abstract—Recurrent neural networks (RNNs) are known to be difficult to train due to the gradient vanishing and exploding problems and thus difficult to learn long-term patterns and construct deep networks. To address these problems, this paper proposes a new type of RNNs with the recurrent connection formulated as Hadamard product, referred to as independently recurrent neural network (IndRNN), where neurons in the same layer are independent of each other and connected across layers. Due to the better behaved gradient backpropagation, IndRNN with regulated recurrent weights effectively addresses the gradient vanishing and exploding problems and thus long-term dependencies can be learned. Moreover, an IndRNN can work with non-saturated activation functions such as ReLU (rectified linear unit) and be still trained robustly. Different deeper IndRNN architectures, including the basic stacked IndRNN, residual IndRNN and densely connected IndRNN, have been investigated, all of which can be much deeper than the existing RNNs. Furthermore, IndRNN reduces the computation at each time step and can be over 10 times faster than the commonly used Long short-term memory (LSTM). Experimental results have shown that the proposed IndRNN is able to process very long sequences and construct very deep networks. Better performance has been achieved on various tasks with IndRNNs compared with the traditional RNN, LSTM and the popular Transformer.

✦

1 INTRODUCTION

LONG-TERM dependency is important for many applications. Especially for applications processing temporal sequences such as action recognition [1], [2] and language processing [3], [4], the past information is important for the recognition of the future events. There are also applications exploring spatial context information such as scene segmentation [5] and spatial pooling [6]. To explore the long-term dependency, recurrent neural networks (RNNs) [7] have been widely used and have achieved impressive results. Compared with the feed-forward networks such as the convolutional neural networks (CNNs), a recurrent connection is added where the hidden state at the previous time step is used as an input to obtain the current state, in order to keep memory of the past information. The update of the hidden states at each time step follows:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{1}$$

where $\mathbf{x}_t \in \mathbb{R}^M$ and $\mathbf{h}_t \in \mathbb{R}^N$ are the input and hidden state at time step t, respectively. $\mathbf{W} \in \mathbb{R}^{N\times M}$, $\mathbf{U} \in \mathbb{R}^{N\times N}$ and $\mathbf{b} \in \mathbb{R}^N$ are the weights for the current input and the recurrent input, and the bias of the neurons, respectively. σ is an element-wise activation function of the neurons, and M and N are the dimension of the input and the number of neurons in this RNN layer, respectively.

Due to the recurrent connections with repeated multiplication of the recurrent weight matrix, training of the RNNs suffers from the gradient vanishing and exploding problem. Despite the efforts in initialization and training techniques [8], [9], [10], [11], it is still very difficult to learn long-term dependency. Several RNN variants such as the long short-term memory (LSTM) [12], [13], [14] and the gated recurrent unit (GRU) [15] have been proposed to address the gradient problems. However, the use of the hyperbolic tangent and the sigmoid functions as the activation function in these variants results in gradient decay over layers. While some researches investigate using deep RNNs with LSTM [16], [17], much research has only focused on using some relative shallow RNNs such as in [1], [18], [19], [20], [21], [22].

On the other hand, the existing RNN models share the same component $\sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b})$ in (1), where the recurrent connection connects all the neurons through time. This makes it hard to interpret and understand the roles of each individual neuron (e.g., what patterns each neuron responds to) without considering the others. Moreover, with the recurrent connections, matrix product is performed at each time step and the computation cannot be easily paralleled, leading to a very time-consuming process when dealing with long sequences.

In this paper, we propose a new type of RNN, referred to as independently recurrent neural network (IndRNN). In the proposed IndRNN, the recurrent inputs are processed with the Hadamard product as $\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{u}\odot\mathbf{h}_{t-1} + \mathbf{b})$. This provides a number of advantages over the traditional RNNs including:

- Able to process longer sequences: the gradient vanishing and exploding problem is effectively addressed by regulating the recurrent weights, and long-term memory can be kept in order to process long sequences. Experiments have demonstrated that an IndRNN can well process sequences over 5000 steps.
- Able to construct deeper networks: multiple layers of IndRNNs can be efficiently stacked, especially with skip-connection and dense connection, to increase the depth of the network. An example of 21-layer

---

- *S. Li and Y. Gao are with Shandong University, Jinan, China. Email: {shuaili, ybgao}@sdu.edu.cn.*
- *W. Li and C. Cook are with University of Wollongong, NSW 2522, Australia. E-mail: {wanqing, ccook}@uow.edu.au*

Manuscript received October 10, 2019.

residual IndRNN and deep densely connected IndRNN are demonstrated in the experiments.

- Able to be robustly trained with non-saturated functions such as ReLU: with the gradient backpropagation through time better behaved, non-saturated function such as ReLU [23] can be used as the activation function and be trained robustly. IndRNN with ReLU is used throughout the experiments.
- Able to interpret the behaviour of IndRNN neurons independently without the effect from the others: since the neurons in one layer are independent of each other, each neuron's behaviour can be interpreted individually. Moreover, the relationship between the range of the memories and the recurrent weights are established through gradient backpropagation, and the memories learned by the task can be understood by visualizing the recurrent weights, as illustrated in experiments.
- Reduced complexity. With the new recurrent connections based on element-wise vector product, which is much more efficient than the matrix product, the complexity of IndRNN is greatly reduced compared with the traditional RNNs (over 10 times faster than the cuDNN LSTM).

Experiments have demonstrated that IndRNN performs much better than the traditional RNN, LSTM and Transformer models on the tasks of the adding problem, sequential MNIST classification, language modelling and action recognition. With the advantages brought by IndRNN, we are able to further show:

- Better performance can be achieved with deeper IndRNN architectures as verified for the sequential MNIST classification, language modelling and skeleton-based action recognition tasks.
- Better performance can be achieved by learning with longer dependency as verified for the language modelling tasks.

Part of this paper has appeared in the conference paper [24] where IndRNN is introduced and verified on some tasks without further analysing its advantage. Significant extension has been made in this paper. 1) New deep IndRNN architecture, densely connected IndRNN is proposed to enhance the feature reuse in addition to the residual IndRNN architecture. 2) The relationship between memory and recurrent weight is established through gradient backpropagation, and the learned memories are visualized for skeleton-based action recognition as an example. 3) More experiments are added including a new task, i.e., word-level language modelling. 4) Experiments are conducted to verify that IndRNN with longer temporal dependency and deeper architecture can achieve better performance. 5) Comparison has been made to the popular Transformer [25], [26] on sequential MNIST classification, Char- and Word-level language modelling and skeleton based action recognition. 6) A faster implementation with CUDA optimization is provided and made publicly available, which can be over 10 times faster than the cuDNN LSTM.

The rest of this paper is organized as follows. Section 2 describes the related work in the literature. Section 3 presents the proposed IndRNN with its gradient backpropagation through time process. It also describes the relationship between the recurrent weight and memory, and its complexity compared with the existing methods. Section 4 explains different deep IndRNN architectures and Section 5 presents the experimental results. Finally, conclusions are drawn at Section 6

2 RELATED WORK

It is known that a simple RNN suffers from the gradient vanishing and exploding problem due to the repeated multiplication of the recurrent weight matrix, which makes it very difficult to train and capture long dependencies. In order to solve the gradient vanishing and exploding problem, long short-term memory (LSTM) [27] was introduced, with a constant error carousel (CEC) to enforce a constant error flow through time. Multiplicative gates including input gate, output gate and forget gate are employed to control the information flow, resulting in many more parameters than the simple RNN. A well known LSTM variant is the gated recurrent unit (GRU) [15] composed of a reset gate and an update gate, which reduces the number of parameters to some extent. It has been reported in various papers [14] that GRU achieves similar performance as LSTM. There are also some other LSTM variants [12], [14], [28], [29] reported in the literature. However, these architectures [12], [14], [28], [29] generally take a similar form as LSTM and show a similar performance as well, so they are not discussed further. LSTM and its variants use gates on the input and the recurrent input to regulate the information flow through the network. However, the use of gates based on the recurrent input prevents parallel computation and thus increases the computational complexity of the whole network. To reduce the complexity and process the states of the network over time in parallel, QRNN (Quasi-Recurrent Neural Network) [30] and SRU (Simple Recurrent Unit) [31] were proposed where the recurrent connections are fixed to be identity connection and controlled by gates with only input information, thus making most of the computation parallel. While this strategy greatly simplifies the computational complexity, it reduces the capability of their RNNs since the recurrent connections are no longer trainable. By contrast, the proposed IndRNN with regulated recurrent weights addresses the gradient exploding and vanishing problems without losing the power of trainable recurrent connections and without involving gate parameters. Moreover, IndRNN reduces computation and runs much faster than LSTM and even SRU [31].

There are also some simple RNN variants trying to solve the gradient vanishing and exploding problem by altering the recurrent connections. In [32], a skip RNN was proposed where a binary state update gate is added to control the network to skip the processing of one time step. In this way, fewer time steps may be processed and the gradient vanishing and exploding problem can be alleviated to some extent. In [33], [34], a unitary evolution RNN was proposed where the recurrent weights are empirically defined in the form of a unitary matrix. In this case, the norm of the backpropagated gradient can be bounded without exploding. In [35], a Fourier Recurrent Unit (FRU) was proposed where the hidden states over time are summarized with Fourier basis functions and the gradients are bounded. However, such methods usually introduce other transforms on the recurrent weight which complicates the recurrent units, making them hard to use and interpret. On the contrary, the proposed IndRNN further simplifies the recurrent connections and makes all the neurons in one layer independent from others, thus easier to interpret (e.g. understanding the long and short memories kept by the task).

There are also attempts at using the non-saturated activation functions such as ReLU (rectified linear unit) to reduce the gradient decay problem introduced by the activation function. While this reduces the gradient decay problem, it greatly aggravates the effect introduced by the repeated use of the recurrent weight matrix, making the network highly unstable. To alleviate this problem, works on initialization and training techniques, such as initializing the recurrent weights to a proper range or regulating the norm of the gradients over time, were proposed in the literature. In [10], an initialization technique was proposed for an RNN with ReLU activation, termed as IRNN, which initializes the recurrent weight matrix to be the identity matrix and bias to be zero. In [11], the recurrent weight matrix was further suggested to be a positive definite matrix with the highest eigenvalue being unity and all the remaining eigenvalues less than 1. In [8], the geometry of RNNs was investigated and a path-normalized optimization method for training was proposed for RNNs with ReLU activation. In [9], a penalty term on the squared distance between successive hidden states' norms was proposed to prevent the exponential growth of IRNN's activation. Although these methods help ease gradient exploding, they are not able to completely avoid the problem (e.g. the eigenvalues of the recurrent weight matrix may still be larger than 1 in the process of training and lead to gradient exploding). Moreover, the training of an RNN with ReLU is very sensitive to the learning rate. When the learning rate is large, the gradient is likely to explode. The proposed IndRNN addresses the gradient problems by regulating the recurrent weights, which are then mapped to the constraint on the maximum gradient. It can work with ReLU and be trained robustly. As a result, an IndRNN is able to process very long sequences (e.g. over 5000 steps as demonstrated in the experiments).

In addition to solving the gradient vanishing and exploding problem in order to process long sequences and learn long-term dependency, efforts are also put into investigating deeper RNN networks. Compared with the deep CNN architectures which could be over 100 layers such as the residual CNN [36] and the densely connected CNN [37], most of the existing RNN architectures only consist of several layers (2 or 3 for example [10], [18], [20]). This is partly due to the gradient vanishing and exploding problem which already results in the difficulty in training a single-layer RNN. A deeper RNN [38], [39] may further aggravate this problem. For LSTM models, since all the gate functions, input and output modulations employ sigmoid or hyperbolic tangent functions as the activation function, it suffers from the gradient decay over layers when multiple LSTM layers are stacked into a deep model. Currently, a few models were reported that employ residual connections [36] between LSTM layers to make the network deeper [40]. As shown in [41], residual networks behave like ensembles of relatively shallow networks. For the residual connections to work efficiently, each residual block desires a good gradient behaviour. Due to gradient decay problem in each LSTM, the deep LSTM model with the residual connections cannot significantly improve the performance, which is also observed in [42]. In [43], a recurrent highway network (RHN) was proposed where at each time step, multiple layers with highway connections are used to process the recurrent state. Compared with the LSTM, the transition process is greatly deepened. Recently, Transformers [25] have been proposed using self-attention. It does not contain any recurrent connections, hence resembles a CNN more than a RNN. All in all, RNN architectures that can be stacked with multiple layers and efficiently trained are still highly desired. The proposed IndRNN well addresses this issue and can construct very deep models with the use of ReLU, residual or dense connections (e.g. over 20 layers as demonstrated in the experiments).

Other than the above works making RNN process long sequences and construct deep networks, there are also other variants such as the multiplicative integration [44], [45], multidimensional [46], [47] or bidirectional extensions, which is beyond the scope of this paper and thus not further explained. One interesting result reported in [29] shows that the capacities of existing RNNs are very similar and the difference in their performances are driven primarily by differences in training effectiveness. Compared with the existing RNNs, the proposed IndRNN uses a simple formulation, and is easy and robust to train with different kinds of techniques such as the ReLU, batch normalization and dropout from CNN.

3 INDEPENDENTLY RECURRENT NEURAL NETWORK (INDRNN)

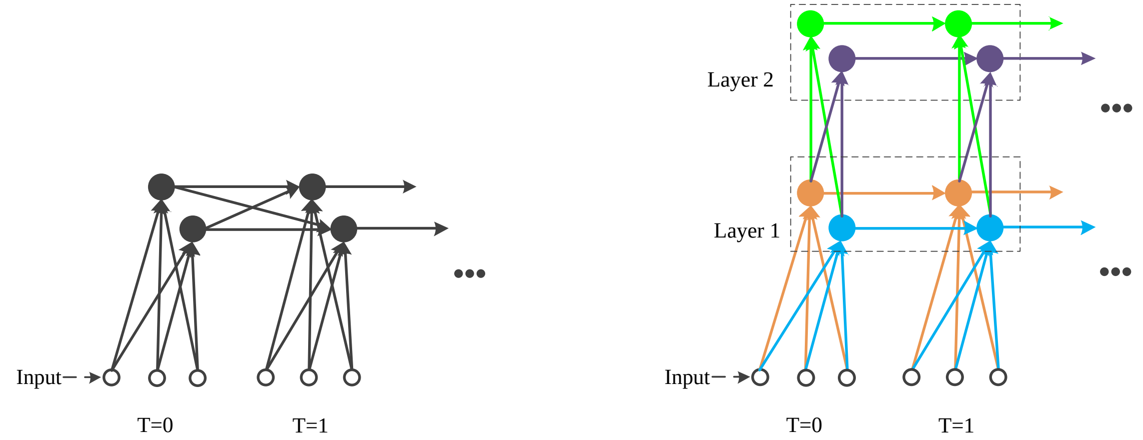
The proposed independently recurrent neural network (IndRNN) follows:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{u} \odot \mathbf{h}_{t-1} + \mathbf{b}) \tag{2}$$

where $\mathbf{x}_t \in \mathbb{R}^M$ and $\mathbf{h}_t \in \mathbb{R}^N$ are the input and hidden state at time step t, respectively. $\mathbf{W} \in \mathbb{R}^{N \times M}$, $\mathbf{u} \in \mathbb{R}^N$ and $\mathbf{b} \in \mathbb{R}^N$ are the weights for the current input and the recurrent input, and the bias of the neurons, respectively. $\odot$ is the Hadamard product, σ is an element-wise activation function of the neurons, and N is the number of neurons in this RNN layer. Compared with the traditional RNN where the recurrent weight $\mathbf{U}$ is a matrix and processes the recurrent input using matrix product, the recurrent weight $\mathbf{u}$ in IndRNN is a vector and processes the recurrent input with element-wise vector product. Each neuron in one layer is independent from others, thus termed as "independently recurrent". For the n-th neuron, the hidden state $h_{n,t}$ can be obtained as

$$h_{n,t} = \sigma(\mathbf{w}_n \mathbf{x}_t + u_n h_{n,t-1} + b_n) \tag{3}$$

where $\mathbf{w}_n$, u_n and b_n are the n-th row of the input weight, recurrent weight and bias, respectively. The proposed IndRNN can be extended to a convolutional IndRNN where, instead of processing input of each time step using a fully



(a) The conventional simple RNN with recurrent connections connecting all neurons over time.

(b) The proposed IndRNN with independent recurrent connections making each neuron in one layer independent while connected over layers.

Fig. 1: Illustration of a conventional simple RNN and the proposed IndRNN unfolded in time. Each solid dot represents a neuron in a layer and each line represents a connection.

connected weight ($\mathbf{W}\mathbf{x}_t$), it is processed with convolutional operation ($\mathbf{W} * \mathbf{x}_t$, where $*$ denotes the convolution operator).

Fig. 1b illustrates the proposed IndRNN unfolded in time. Each solid dot represents a neuron in a layer and each line represents a connection. It can be seen that the processing of each neuron in one layer is independent from each other as noted by different colors. Each neuron only receives information from the input and its own hidden state at the previous time step. That is, each neuron in an IndRNN deals with one type of spatial-temporal pattern independently. On the contrary, neurons in the simple RNN are connected together over time by the recurrent weight connection as shown in Fig. 1a. Fig. 1b also shows that one more IndRNN layer can explore the correlation (cross-channel information) among the neurons in one layer as the neuron in the following layer takes output of all the neurons in the previous layer as input. The relationship between IndRNN and the conventional RNN is illustrated in the Appendix A, where we have shown that under certain circumstances (e.g. linear activation), a traditional RNN is a special case of a two-layer IndRNN.

3.1 Backpropagation Through Time for An IndRNN

Since the neurons in each layer are independent of each other, the gradient backpropagation through time for neurons in one layer can also be performed independently. For the n-th neuron $h_{n,t} = \sigma(\mathbf{w}_n\mathbf{x}_t + u_n h_{n,t-1})$ where the bias is ignored, suppose the objective trying to minimize at time step T is J_n. Then the gradient back propagated to the time step t is

$$\begin{aligned}\frac{\partial J_n}{\partial h_{n,t}} &= \frac{\partial J_n}{\partial h_{n,T}}\frac{\partial h_{n,T}}{\partial h_{n,t}} = \frac{\partial J_n}{\partial h_{n,T}}\prod_{k=t}^{T-1}\frac{\partial h_{n,k+1}}{\partial h_{n,k}} \\ &= \frac{\partial J_n}{\partial h_{n,T}}\prod_{k=t}^{T-1}\sigma'_{n,k+1}u_n = \frac{\partial J_n}{\partial h_{n,T}}u_n^{T-t}\prod_{k=t}^{T-1}\sigma'_{n,k+1} \qquad (4)\end{aligned}$$

where $\sigma'_{n,k+1}$ is the derivative of the element-wise activation function. It can be seen that the gradient only involves the exponential term of a scalar value u_n which can be easily regulated, and the gradient of the activation function which is often bounded in a certain range. Compared with the gradients of an RNN ($\frac{\partial J}{\partial h_T}\prod_{k=t}^{T-1} diag(\sigma'(h_{k+1}))\mathbf{U}^T$ where $diag(\sigma'(h_{k+1}))$ is the Jacobian matrix of the element-wise activation function), the gradient of an IndRNN directly depends on the value of the recurrent weight (which is changed by a small magnitude according to the learning rate) instead of matrix product (which is mainly determined by its eigenvalues and can be changed significantly even though the change to each matrix entries is small [48]). Thus the training of an IndRNN is more robust than a traditional RNN.

To address the gradient exploding and vanishing problem over time, we need to regulate the exponential term "$u_n^{T-t}\prod_{k=t}^{T-1}\sigma'_{n,k+1}$" to an appropriate range (ignoring the gradient backpropagated from the objective to the hidden state at time step T, $\frac{\partial J_n}{\partial h_{n,T}}$). Note that for some activation functions such as ReLU, when not activated, its derivative is 0 and the neuron cannot learn anything. Here we only discuss the gradient when neurons are activated ($\prod_{k=t}^{T-1}\sigma'_{n,k+1} \neq 0$). For gradient exploding, denoting the magnitude of the largest gradient value without exploding by γ, the constraint "$|u_n^{T-t}\prod_{k=t}^{T-1}\sigma'_{n,k+1}| \leq \gamma$" avoids the gradient exploding problem. Accordingly, the recurrent weight $|u_n|$ satisfies $|u_n| \leq \sqrt[(T-t)]{\frac{\gamma}{|\prod_{k=t}^{T-1}\sigma'_{n,k+1}|}}$. For the commonly used activation functions such as ReLU and tanh, their derivatives are no larger than 1, i.e., $|\sigma'_{n,k+1}| \leq 1$. Especially for ReLU, its gradient σ' is either 0 or 1. Therefore, this constraint can be simplified to $|u_n| \leq \sqrt[(T-t)]{\gamma}$. For especially long sequences, it can be further simplified to $|u_n| \leq 1$.

On the other hand, for the gradient vanishing problem, the magnitude of the gradient needs to be larger than 0, which can be met by $|u_n| > 0$. In the literature and practice, when gradient vanishing is referred, usually we expect the gradient after many time steps (i.e. $T-t$) to be larger than a small value (ϵ) in order for the gradient to effectively make

TABLE 1: Relationship between recurrent weight and memory through gradient.

Memory	Gradient	Recurrent weight
Long-term	effective after $T-t$ steps: $\frac{\partial J_{n,T}}{\partial h_{n,t}} > \epsilon$	$\lvert u_n\rvert \in (\sqrt[(T-t)]{\frac{\epsilon}{\lvert\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}\rvert}}, +\infty)$
Short-term	reduced after $T-t$ steps: $\frac{\partial J_{n,T}}{\partial h_{n,t}} \leq \epsilon$	$\lvert u_n\rvert \in [0, \ \sqrt[(T-t)]{\frac{\epsilon}{\lvert\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}\rvert}}]$
	not exploding: $\frac{\partial J_{n,T}}{\partial h_{n,t}} \leq \gamma$	$\lvert u_n\rvert \in [0, \ \sqrt[(T-t)]{\frac{\gamma}{\lvert\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}\rvert}}]$

a change. Therefore, we further strength the above gradient constraint to $|u_n^{T-t}\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}| > \epsilon$. Accordingly, we can obtain the range for the recurrent weights to effectively tackle the gradient vanishing as $|u_n| > \sqrt[(T-t)]{\frac{\epsilon}{|\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}|}}$. One advantage of IndRNN is that it can work robustly with ReLU as activation function. For ReLU, when the neuron is activated, its gradient is fixed to be 1 without decay, greatly alleviating the gradient vanishing problem in the recurrent processing. Therefore, the constraint can be simplified to $|u_n| > \sqrt[(T-t)]{\epsilon}$. In other words, with the recurrent weight regulated in this range $|u_n| > \sqrt[(T-t)]{\epsilon}$, IndRNN with ReLU can avoid gradient vanishing. This is the constraint for one neuron, and for IndRNN with many neurons, this constraint can be further relaxed. For a multiple layers RNN with each layer containing multiple neurons, different neurons may learn different features including features from different ranges of memories. For IndRNN, neurons in one layer are independent from each other, the vanishing gradient of one neuron does not affect the learning of other neurons. For example, some IndRNN neurons in the first few layers may only learn the spatial information at its own time step projecting the input into a feature space, then the following layers can process such features in the temporal domain. Such neurons in the first few layers do not concern any memory and the gradient backpropagated from time is zero (gradient vanishing from the perspective of temporal gradient backpropagation for this specific neuron), but the gradients backpropagated from the deep layers at the current step still exist. Moreover, with the temporal gradient backpropagation of neurons independent in one layer, the above neurons do not affect the learning of other neurons in the same layer. Therefore, the condition of avoiding gradient vanishing shown above ($|u_n| > \sqrt[(T-t)]{\epsilon}$) can be further relaxed to $|u_n| > \sqrt[(T-t)]{\epsilon}, \exists u_n \in \mathbf{u}$, which can be easily met. Although this condition can be forcibly applied in the implementation, in practice, we found that there is no need for even such a relaxed constraint. The recurrent weights generally distribute in the whole range of memories even under no constraints as shown in the following Subsection 5.3.3 Fig. 6, and highly unlikely that all the recurrent weights are trained to be close to zero with the gradient of the whole network close to zero.

Therefore, for robust training of IndRNN without gradient exploding and vanishing, only the constraint $|u_n| \leq \sqrt[(T-t)]{\gamma}$ is needed and thus used in the experiments. Note that the regulation on the recurrent weight u is different from the gradient or gradient norm clipping technique. For the gradient clipping or gradient norm clipping [49], the calculated gradient has already exploded and is forced back to a predefined range. The gradients for the following steps may keep exploding. In this case, the gradient of the other layers relying on this neuron may not be accurate. On the contrary, the regulation proposed here essentially maintains the gradient in an appropriate range without affecting the gradient backpropagated through this neuron.

3.2 Recurrent Weight and Memory

In this Subsection, the relationship between the recurrent weight and the memory keeping mechanism in IndRNN is analyzed to explicitly show the range of the recurrent weights for keeping different ranges of memories.

One of the key roles of RNNs is to keep memories of the past. That is to say, the current state at time step T can be affected by the past state at time step t. In other words, a change of the past state (h_t) changes the current state (h_T). From the perspective of gradient backpropagation, the gradient from h_T propagated to h_t could still be effectively updating the weights. Assume the minimum effective gradient is ϵ, and the gradient follows $|\frac{\partial J_{n,T}}{\partial h_{n,t}}| > \epsilon$. Accordingly, a range for the recurrent weight to keep memory of $T-t$ time steps can be obtained, which is $|u_n| \in (\sqrt[(T-t)]{\frac{\epsilon}{|\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}|}}, +\infty)$ according to (4) (ignoring the gradient backpropagated from the objective to the hidden state at time step T, $\frac{\partial J_n}{\partial h_{n,T}}$). On the contrary, when only short-term memory is required, the gradient from h_T propagated to h_t should not be able to effectively update the weights, and the gradient follows $|\frac{\partial J_{n,T}}{\partial h_{n,t}}| \leq \epsilon$. The range of the recurrent weight can be obtained accordingly as $|u_n| \in [0, \sqrt[(T-t)]{\frac{\epsilon}{|\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}|}}]$. In an extreme case, the short-term memory is no memory of the past at all, where the recurrent weight is 0 and the gradient backpropagated from the future time steps is zero (the gradient backpropagated from the deeper layers still exists). The short-term memories can be important for the performance of the network as well. Especially for a multiple layers RNN, the short-term memories from the earlier layers may be accumulated in the following layer with long-term memory, which is also the reason that the gradient constraint on the vanishing gradient in the above subsection can be alleviated. Together with the basic requirement of avoiding the gradient exploding described above, the relationships among recurrent weight and memory through the gradient can be depicted as Table 1. With this relationship between the recurrent weight and memory, the range of memories learned by the task can be visualized through the learned recurrent weights, which is illustrated in the experiments.

Without any prior knowledge, the short-term memories and long-term memories are assumed to be both important for sequence problems. Therefore, the recurrent weight can be initialized in the range $[0, \sqrt[(T-t)]{\frac{\gamma}{|\prod_{k=t}^{T-1}\sigma'{}_{n,k+1}|}}]$, where different neurons can keep memories of different lengths.

For ReLU, it can be simplified to $|u_n| \in [0, \sqrt[(T-t)]{\gamma}]$. The training of the network can further refine the network to learn required memories for the task by updating the recurrent weights. On the other hand, for tasks such as classification after reading a sequence, for the last layer, only the hidden state at the last time step is used for classification. In order to make the hidden state at the last time step rich of information as much as possible, it is better to keep the long-term memory to aggregate useful information from the beginning to the last time step. Therefore, the recurrent weights can be initialized in the range $[\sqrt[(T-t)]{\frac{\epsilon}{|\prod_{k=t}^{T-1}\sigma'_{n,k+1}|}}, \sqrt[(T-t)]{\frac{\gamma}{|\prod_{k=t}^{T-1}\sigma'_{n,k+1}|}}]$ (which is $[\sqrt[(T-t)]{\epsilon}, \sqrt[(T-t)]{\gamma}]$ for ReLU) to only keep long-term memory, which is also observed to improve performance in the experiments.

3.3 Complexity in terms of the Number of Parameters and Computation

Regarding the number of parameters, for a N-neuron RNN network with input of dimension M, the number of parameters in a traditional RNN is $M \times N + N \times N + N$, while the number of parameters for LSTM is $4*(M \times N + N \times N + N)$, which is much larger than the simple RNN. By contrast, IndRNN uses a simple formulation with less parameters, $M \times N + 2 \times N$. Even for a two-layer IndRNN where both layers consist of N neurons, the number of parameters is $M \times N + N \times N + 4 \times N$, which is of a similar order to the traditional RNN. Usually the number of parameters used in a network is large (e.g. much larger than $3N$), therefore, the number of parameters introduced by the recurrent weight vectors and extra bias vector ($3 \times N$) is negligible compared with the recurrent weight matrix ($N \times N$) used in the traditional RNN. And obviously, IndRNN uses much less parameters than LSTM.

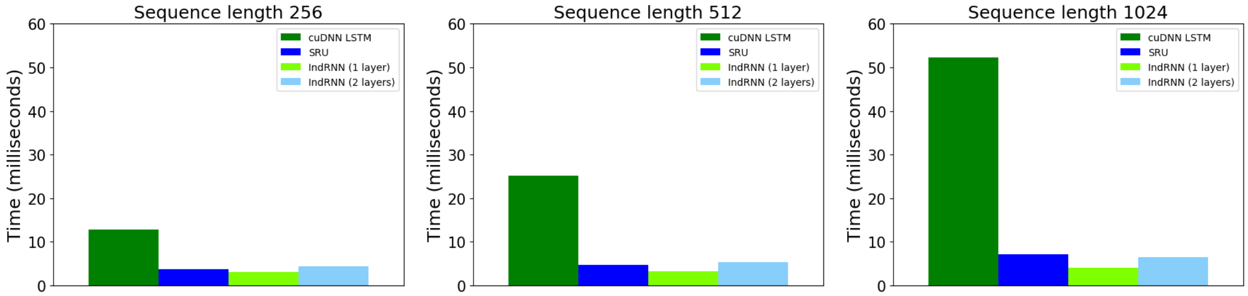
From the perspective of computation, the processing of the input ($\mathbf{W}\mathbf{x}_t + \mathbf{b}$) is independent at different timesteps and can be implemented in parallel, which is the same for the conventional RNNs. For the processing of the recurrent input (which is usually the most time-consuming part in RNNs), IndRNN only applies one element-wise vector product operation, one adding operation and one activation calculation, involving less computation than the traditional RNNs with matrix product. Moreover, IndRNN works efficiently with ReLU, which is more efficient than other saturated functions such as the tanh and sigmoid used in the traditional RNNs. To further accelerate the computation on GPU, similarly as SRU [31] we implemented a fast CUDA optimized version based on PyTorch which is made publicly available. It combines the operations of element-wise vector product, adding and activation calculation to avoid the latency in calling each function separately.

Fig. 2 illustrates the computation complexity comparison in terms of computation time on different lengths of sequences including 256, 512 and 1024. The program is tested on a GPU P100 and the training time for each batch (averaged over 100 batches) including both the forward and backward time (milliseconds) are shown in the figure. In our experiments, we found that this is more accurate than using the forward and backward time separately for each batch. The one-layer IndRNN and two-layer IndRNN are evaluated and the cuDNN LSTM and SRU [31] (both one layer) are used for comparison. The detailed setup is shown in Subsection 5.2.1. It can be seen that IndRNN (both 1-layer and 2-layer) takes much less time than LSTM. To be specific, for sequences of length 256, 512 and 1024, a 1-layer IndRNN is 4.3, 7.6 and 12.9 times faster than LSTM, respectively. Even a 2-layer IndRNN is 2.9, 4.8 and 8.0 times faster than a 1-layer LSTM for sequences of length 256, 512 and 1024, respectively. It is worth noting that while the time for processing longer sequences using LSTM is dramatically increasing, the time using IndRNN only increases a very small amount. This indicates that compared with the matrix product (in the weight processing of the input), the element-wise product and addition is much faster, making IndRNN highly efficient with a comparable computation as a feedforward network. Moreover, it can also be seen that the one-layer IndRNN is faster than SRU [31] and even two-layer IndRNN is faster than SRU [31] for processing sequence of length 1000, which should be the case since SRU uses more computations at each time step. The memory consumptions of different RNNs also follow closely to the number of parameters and the computation, which will not be further detailed.

4 DEEP INDEPENDENTLY RECURRENT NEURAL NETWORK

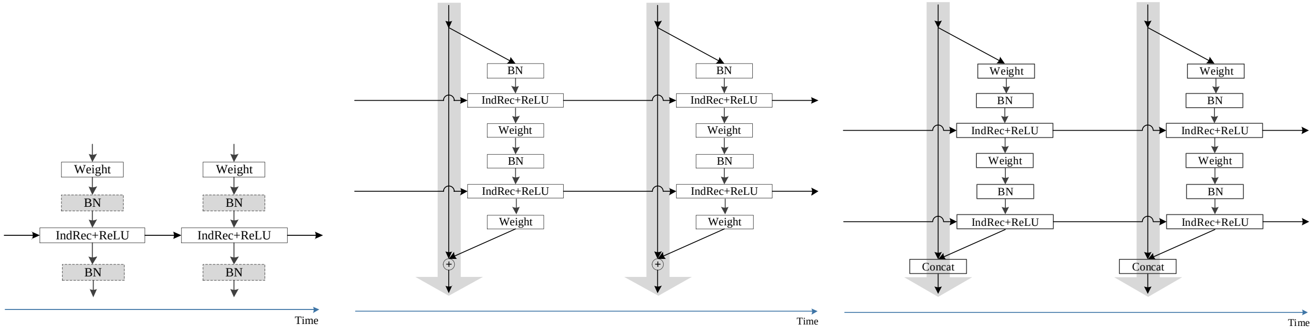
As described in the above Section, IndRNN can be stacked to construct multiple-layer IndRNN networks to better explore the cross-channel information. One important advantage of IndRNN is that it can work robustly with non-saturated activation functions such as ReLU, since the gradient behaviour of IndRNN can be well regulated. The non-saturated functions provide stable gradient propagation when activated (feature values larger than 0), which can greatly facilitate the development of deep networks. While in the current CNNs, non-saturated activation functions such as ReLU have been widely used and become the default setting, it is still not prevailing in the construction of RNNs due to the gradient exploding over time. Although there are some attempts on using ReLU as activation function such as IRNN [10], it is very unstable and even carefully trained with a very small learning rate (10^{-5}), the model may still explode. Most of the RNN models adopt gates with saturated functions to address the gradient exploding problem. To the best of our knowledge, IndRNN is the only one that stably and completely works with non-saturated activation functions such as ReLU without any gate function including saturated activation. Thus IndRNN can efficiently construct deep networks.

Several deep IndRNN architectures are proposed in this paper, including the basic stacked IndRNN, residual IndRNN (res-IndRNN) and densely connected IndRNN (dense-IndRNN). First, the building block for the basic IndRNN architecture is shown in Fig. 3a, where “Weight” and “IndRec+ReLU” denote the processing of input and the recurrent process at each step with ReLU as the activation function. In addition, batch normalization, denoted as “BN”, can also be employed in the IndRNN network before or after the activation function as shown in Fig. 3a. By stacking this block, a deep basic IndRNN network can be constructed.



(a) Sequence length 256 (b) Sequence length 512 (c) Sequence length 1024

Fig. 2: Complexity comparison in terms of time training one batch (milliseconds) for processing sequences of different lengths. Note that in our experiments the time for one forward or backward process is not very accurate due to the precision of the system clock since the time taken by the forward process is very short. Therefore, the total time of training one batch (including fetching data, forward pass, loss calculation and backward pass) is averaged over 100 batches.



(a) Basic IndRNN architecture (b) Residual IndRNN architecture (c) Densely connected IndRNN architecture

Fig. 3: Illustration of different deep IndRNN architectures.

Since the weight layer ($\mathbf{W}\mathbf{x}_t + \mathbf{b}$) is used to process the input, it is natural to extend it to multiple layers to deepen the processing, making the architecture flexible with different depths to process input features.

The second type of deep IndRNN architecture is the residual IndRNN shown in Fig. 3b, which adds an identity shortcut (skip-connection) [36] to bypass the non-linear transformation of the input features. Denoting the non-linear transformation at layer l and time step t by $\mathcal{F}_{l,t}$, it follows $x_{l,t} = x_{l-1,t} + \mathcal{F}_{l,t}(x_{l-1,t})$, where $x_{l,t}$ is the output feature at layer l. Since the non-linear transformations containing IndRNN are shared over time, $\mathcal{F}_{l,t}$ can be simplified to $\mathcal{F}_l$ and accordingly, $x_{l,t} = x_{l-1,t} + \mathcal{F}_l(x_{l-1,t})$. It can be seen that the skip-connection does not affect the processing in the time dimension, but makes the deeper feature a summation of the shallower feature and a residual feature processed with $\mathcal{F}_l$. In this case, the gradient can be directly propagated to the previous layers through the skip-connection, greatly alleviating the gradient decay problem across layers. Therefore, substantially deeper residual IndRNN can be constructed. Fig. 3b shows the non-linear transformation $\mathcal{F}_{l,t}$ of type "pre-activation" similarly as in [50], which uses the processing of a composite function of three consecutive operations: batch normalization (BN), the recurrent processing and activation calculation (IndRec+ReLU) over time, and the weight processing (Weight), denoted by "BN-IndRec+ReLU-Weight" for simplicity. Also in each residual block, different numbers of "BN-IndRec+ReLU-Weight" operation can be used, and two operations are shown in the figure.

The third type of deep IndRNN architecture is the densely connected IndRNN shown in Fig. 3c, which, instead of using a skip-connection, uses a concatenation operation to combine all the features of the previous layers [37]. It follows $x_{l,t} = \mathcal{C}(x_{l-1,t}, \mathcal{F}_l(x_{l-1,t}))$, where $\mathcal{C}$ is concatenation operation. By substituting $x_{l-1,t}$ recursively, it can be easily seen that $x_{l,t}$ is a combination of all the features in the previous layers and the non-linear transformation of the features in the previous layer. This also forms an identity function among deeper layers and shallower layers, thus allowing gradients to be properly propagated. Compared with the residual IndRNN, it explicitly processes all the features from all the previous layers, encouraging the feature reuse and thus reducing the number of parameters. Different composite functions can be used as in the residual IndRNN, and Fig. 3c shows an example with the composite function "Weight-BN-IndRec+ReLU". As the number of layers concatenated together increases, the dimension of features also increases, leading to an increased number of parameters. Therefore, transition layers as in [37] are used to separate dense layers into dense blocks to reduce the dimension of features. More details on the densely connected IndRNN architecture settings are shown in the following Subsection 5.1 and Appendix B.

We would like to note that while the residual connection and dense connection can also be applied to the other RNN models, they may behave better on IndRNN. As reported in [41], residual networks behave like ensembles of relatively shallow networks. The performance of residual networks is highly dependent on the depth of the model (without residual connections) that can be constructed and effectively trained. In other words, it still depends on the gradient of the model without the residual connection. The dense connection can also be analysed similarly. Therefore, combined with the residual or dense connections, IndRNN is expected to work better than other RNN models with better gradient behaviour over layers. In all, IndRNN can construct deep RNN models and even deeper models with residual and dense connections.

5 EXPERIMENTS

In this Section, first, the capability of IndRNN in processing long sequences and constructing deep networks is verified. Then the performance of the proposed IndRNN on various tasks are evaluated.

5.1 Training Setup

ReLU was used for activation throughout the experiments. For tasks with output at every timestep such as the language modeling problem in Subsection 5.2.3, the recurrent weights of all layers were initialized uniformly in the range $[0,\ \sqrt[(T-t)]{\gamma}]$, while for tasks such as classification where only the output at the last time step in the last layer is used, the recurrent weights of the last layer were initialized uniformly in the range $[\ \sqrt[(T-t)]{\epsilon},\ \sqrt[(T-t)]{\gamma}]$ as described in Subsection 3.2. The recurrent weights are constrained to be in the range of $|u_n| \leq \sqrt[(T-t)]{\gamma}$ for IndRNN with ReLU as analysed in Subsection 3.1. This is especially important in processing long sequences where a small change of recurrent weights may significantly change the gradients. In our experiments, γ is simply set to 1.0 for long sequences since $\sqrt[(T-t)]{\gamma}$ is already close to 1 when T is very large. For short sequences such as 20 steps, the constraint can also be removed as the gradient exploding problem is not very severe in such cases. We have also conducted an ablation study on γ, which also verifies that the performance is not very sensitive to the setting of γ as long as it is in a reasonable range (values in $[1, 10]$ for example).

To accelerate training, batch normalization was used except in the simple adding problem. Moreover, for classification tasks where the whole sequence is processed for output at the final time step, the statistics used in the batch normalization layer were obtained based on the samples at all time steps, while for other tasks such as language modeling which cannot access information from future time steps, the statistics are obtained based on all the samples in each time step. When dropout is applied, the dropout mask is shared over time to avoid the clipping of long memory. Weight decay of 10^{-4} is used for the weight parameters (without applying to the recurrent weight and bias). All the networks were trained using the Adam optimization method [51] with initial learning rate 2×10^{-4}. The learning rate is reduced by a factor of 5 when the accuracy (or loss) on the validation data no longer improves (drops) (with patience set to 100).

For the densely connected IndRNN, the network shape was simply following the conventional denseCNN [37]. Each dense layer (the non-linear transformation function) consists of two composite functions as shown in Fig. 3c and produces k feature maps, termed as the growth rate. The first composite function is called the bottleneck layer and the number of neurons is set to four times (4*k) the growth rate. For each transition layer, the number of the neurons is set to be the half (50%) of the input feature maps. The dense block configuration is set to (8,6,4), where in the first, second and third dense block, 8, 6 and 4 dense layers are used, respectively. This keeps a relatively similar number of neurons in each dense block. Note that it is different from the denseCNN [37] because the tasks in the following experiments do not concern pooling (which reduces the size of the features). For the whole network, one IndRNN layer with six times (6*k) the growth rate are used to process the input first before going through the following dense layers. In the following, the residual IndRNN and the densely connected IndRNN are noted as res-IndRNN and dense-IndRNN, respectively.

5.2 Verification of Processing Long Sequences and Constructing Deep Networks

5.2.1 *Adding Problem*

The adding problem [27], [33] is commonly used to evaluate the performance of RNN models. Two sequences of length T are taken as input. The first sequence is uniformly sampled in the range $(0, 1)$ while the second sequence consists of two entries being 1 and the rest being 0. The output is the sum of the two entries in the first sequence indicated by the two entries of 1 in the second sequence. Three different lengths of sequences, $T = 100$, 500 and 1000, were used for the experiments to show whether the tested models have the ability to model long-term memory.

The RNN models included in the experiments for comparison are the traditional RNN with tanh, LSTM, IRNN (RNN with ReLU). The proposed IndRNN was evaluated with ReLU activation function. Since GRU achieved similar performance as LSTM [14], it is not discussed here. RNN, LSTM, and IRNN are all one layer while the IndRNN model is two layers. 128 hidden units were used for all the models, and the number of parameters for RNN, LSTM, and two-layer IndRNN are $16K$, $67K$ and $17K$, respectively. It can be seen that the two-layer IndRNN has a comparable number of parameters to that of the one-layer RNN, while many more parameters are needed for LSTM. As discussed in Subsection 3.1, the recurrent weight is constrained in the range of $|u_n| \in (0,\ \sqrt[T]{2})$ for the IndRNN.

Mean squared error (MSE) was used as the objective function and the Adam optimization method [51] was used for training. The baseline performance (predicting 1 as the output regardless of the input sequence) is mean squared error of 0.167 (the variance of the sum of two independent uniform distributions). The initial learning rate was set to 2×10^{-3} for models with tanh activation and set as 2×10^{-4} for models with ReLU activations. However, as the length of the sequence increases, the IRNN model does not converge

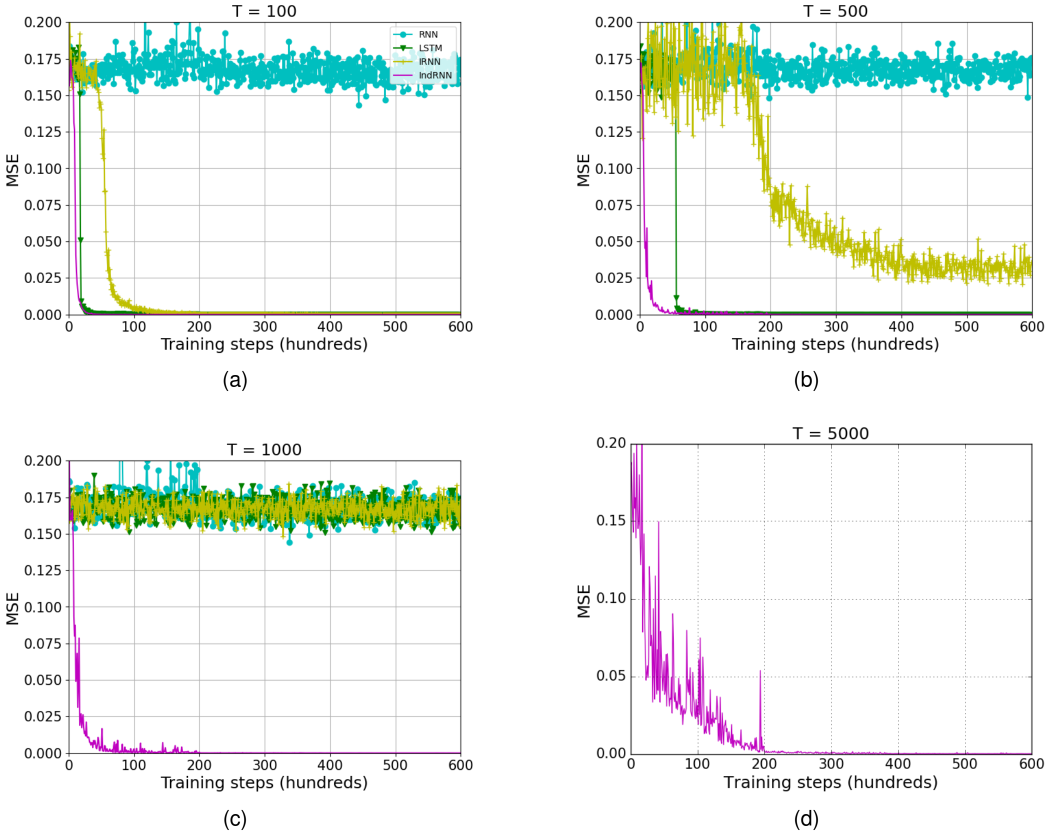


Fig. 4: Results of the adding problem for different sequence lengths. The legends for all figures are the same and thus only shown in (a).

and thus a smaller initial learning rate (10^{-5}) was used. The learning rate was reduced by a factor of 10 every 20K training steps. The training data and testing data were all generated randomly throughout the experiments, different from [33] which only used a set of randomly pre-generated data.

The results are shown in Fig. 4a, 4b and 4c. First, for short sequences ($T = 100$), most of the models (except RNN with tanh) performed well as they converged to a very small error (much smaller than the baseline). When the length of the sequences increases, the IRNN and LSTM models have difficulties in converging, and when the sequence length reaches 1000, IRNN and LSTM cannot minimize the error any more. However, the proposed IndRNN can still converge to a small error very quickly. This indicates that the proposed IndRNN can model a longer-term memory than the traditional RNN and LSTM.

From the figures, it can also be seen that the traditional RNN and LSTM can only keep a mid-range memory (about 500 - 1000 time steps). To evaluate the proposed IndRNN model for very long-term memory, experiments on sequences with length 5000 were conducted where the result is shown in Fig. 4d. It can be seen that IndRNN can still model it very well. Note that the noise in the result of IndRNN is because the initial learning rate (2×10^{-4}) was relatively large and once the learning rate dropped, the performance became robust. This demonstrates that IndRNN can effectively address the gradient exploding and vanishing problem over time and keep a long-term memory.

The complexity of IndRNN was evaluated using the adding problem (with the similar configuration as above) compared with the cuDNN LSTM and SRU [31] which has been reported to be much faster than LSTM. Sequences of different lengths including 256, 512 and 1024 are all tested to be consistent with [31]. A GPU P100 is used as the test platform and the training time for each batch (averaged over 100 batches) including both the forward and backward time (milliseconds) are measured. The comparison is shown in Fig. 2. It can be seen that IndRNN (both 1-layer and 2-layer) takes much less time than LSTM, and reaches 12.9 times faster than LSTM when sequence length is 1024. It is also faster than SRU [31], and even a two-layer IndRNN is faster than SRU for processing sequence of length 1000. Moreover, while the time for processing longer sequences using LSTM is dramatically increasing, the time using IndRNN only increases a little bit, indicating that the recurrent connection is very efficient.

5.2.2 Sequential MNIST Classification

Sequential MNIST classification has been widely used to evaluate RNN models in capturing long dependencies. The pixels of MNIST digits [54] are presented sequentially to the networks and classification is performed after reading all pixels. Each digit image consists of 784 pixels ($28 * 28$), and to achieve a good classification accuracy, the models need to remember the patterns of such a long sequence, thus

TABLE 2: Results in terms of accuracy (%) for the sequential MNIST and permuted MNIST.

	MNIST	pMNIST
IRNN [10]	95.0	82
uRNN [33]	95.1	91.4
RNN-path [8]	96.9	-
LSTM [33]	98.2	88
LSTM+Recurrent dropout [52]	-	92.5
LSTM+Recurrent batchnorm [53]	-	95.4
LSTM+Zoneout [18]	-	93.1
LSTM+Recurrent batchnorm+Zoneout [18]	-	95.9
Transformer (reported in [26])	98.2	-
R-Transformer [26]	99.1	-
IndRNN (6 layers)	99.0	96.0
IndRNN (12 layers)	99.37	96.84
res-IndRNN (12 layers)	99.39	97.02
dense-IndRNN	**99.48**	**97.20**

verifying the capability of the RNN models in capturing long-term dependency. To make the task even harder, the pixels are first processed with a fixed random permutation and then used for classification, known as the permuted MNIST classification. First, a six-layer IndRNN is used for test. Each layer contains 128 neurons, and batch normalization is inserted after each layer. Dropout with a dropping probability of 0.1 is used after each layer. 5% of the training data is reserved for validation. The results are shown in Table 2 in comparison with the existing methods. It can be seen that IndRNN achieved better performance than the existing RNN models.

To further verify that IndRNN can be stacked into very deep networks and still be effectively trained with a good performance, a deep IndRNN with 12 layers, a deep residual IndRNN (12 layers) and a densely connected IndRNN are also tested. For the deep IndRNN and the residual IndRNN, the settings are the same as the above 6-layer IndRNN, e.g., 128 neurons and dropout (0.1) applied after each layer. For the densely connected IndRNN, the growth rate is set to 16, and dropout is applied after the input (0.2), each dense layer (0.2), each bottleneck layer (0.1) and each transition layer (0.1). The results are also shown in Table 2. Compared with the 6-layer IndRNN, deeper IndRNN (12 layers) achieves better performance, higher than 99% and 97% in terms of accuracy for the normal MNIST classification and permuted MNIST classification, respectively. Moreover, the residual IndRNN and the densely connected IndRNN outperforms the simple stacked IndRNN by further improving the accuracy to 99.48% and 97.20%, respectively, validating its effectiveness.

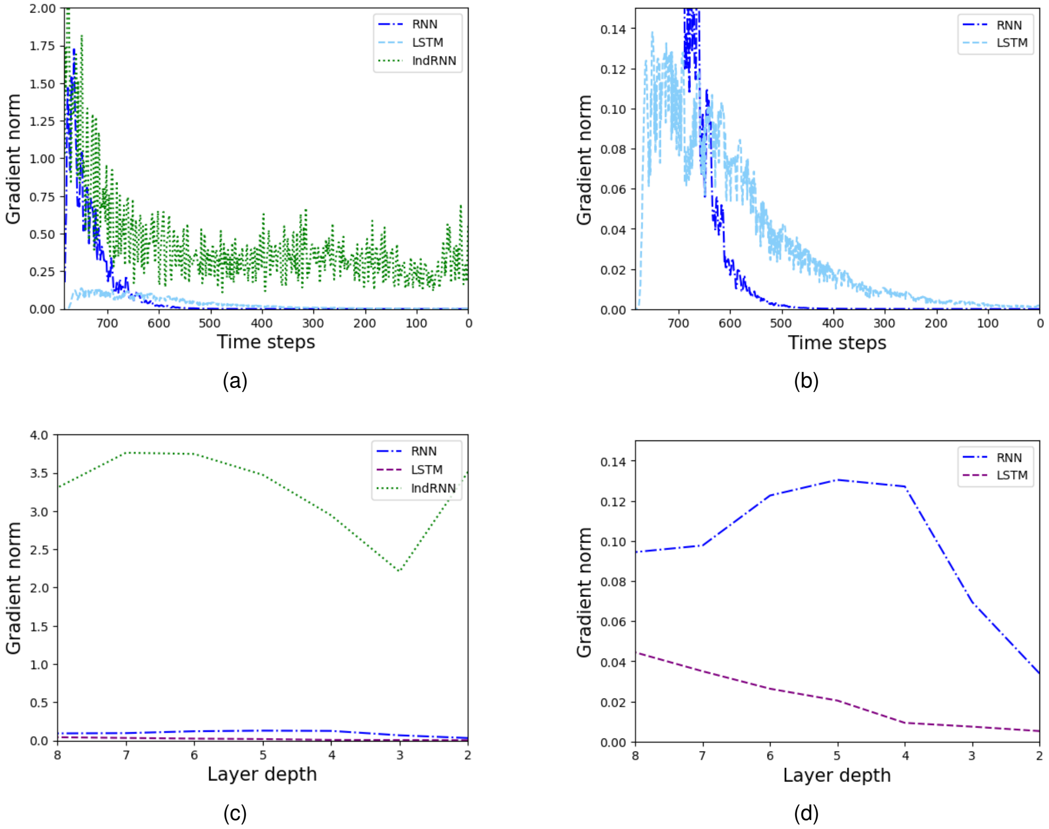
To experimentally compare the gradient behaviour of different models, the gradient flow over time and layer depth are illustrate in Fig. 5. Models of 8 layers are used for different RNNs and the other settings are the same as above. The gradients are obtained under a similar training accuracy (92%) for all RNNs, i.e., a similar loss and gradient backpropagated from the objective. For the gradient flow over time, the gradient backpropagated to the input is used with 784 entries. For the gradient flow over layer depth, the average gradient norm of the input processing weights, which is in the layer depth gradient backpropagation path, is used. The gradients of the first layer is omitted since it processes the pixel input with different dimensions from the other layers. The gradients are obtained as the average gradient norms over an epoch to be stable for comparison. Fig. 5a shows the gradient change over time and Fig. 5b further zoomed the result of RNN and LSTM for better visualization. It can be seen that the gradient of RNN and LSTM are usually smaller than IndRNN, and the gradient decays over time. Especially for RNN, the gradient decays very quickly and close to zero in the time direction. By contrast, IndRNN maintains a relatively large gradients over time without decaying. On the other hand, as shown in Fig. 5c and 5d, in the direction of layer depth, IndRNN also maintains relatively large gradients while the gradients of RNN and LSTM drops quickly. Especially for LSTM, the gradient decays severely in the layer depth direction which agrees with the analysis. It is worth noting that a LSTM of 12 layers cannot be trained to converge in our experiments mostly due to the above gradient problem. In all, the gradient of IndRNN behaves very well in both directions of time and layer depth as theoretically analysed above, and thus it is able to construct deep models and process long sequences.

TABLE 3: Results of char-level PTB for our proposed IndRNN model in comparison with results reported in the literature, in terms of BPC.

	Test
RNN-tanh [9]	1.55
RNN-ReLU [8]	1.55
RNN-TRec [9]	1.48
RNN-path [8]	1.47
HF-MRNN [55]	1.42
LSTM [18]	1.36
LSTM+Recurrent dropout [52]	1.32
LSTM+Recurrent batchnorm [53]	1.32
LSTM+Zoneout [18]	1.27
HyperLSTM + LN [56]	1.25
Hierarchical Multiscale LSTM + LN [57]	1.24
Fast-slow LSTM [58]	1.19
Neural Architecture Search [59]	1.21
Transformer (reported in [26])	1.45
R-Transformer [26]	1.24
IndRNN (6 layers, 50 steps)	1.26
IndRNN (6 layers, 100 steps)	1.21
res-IndRNN (21 layers, 50 steps)	1.21
res-IndRNN (21 layers, 100 steps)	1.17
dense-IndRNN (50 steps)	1.19
dense-IndRNN (100 steps)	1.16

5.2.3 Char-level Penn Treebank

The character-level and word-level language modeling tasks are also used for evaluation. In this subsection, we first evaluate the performance using the character-level Penn Treebank (PTB-c) dataset. The test setting is similar to [53]. Different network architectures are also tested, including a six-layer IndRNN, a 21-layer residual IndRNN (to demonstrate that the IndRNN network can be very deep with the residual connections) and a densely connected IndRNN. For the six-layer IndRNN and the 21-layer residual IndRNN, 2000 hidden neurons are used and dropout with dropping probabilities of 0.25 and 0.3 were used, respectively. While for the densely connected IndRNN, the growth rate is set to 256 and 600 hidden neurons are used for the embedding layer. Dropout is applied after each dense layer (0.5), each bottleneck layer (0.1) and each transition layer (0.4). Dropout of 0.3 is used at the last transition layer, which is connected to the output layer. The batch size is set to 128



(a) (b)

(c) (d)

Fig. 5: Illustration of the gradient flow over time and layer depth for different RNNs. (b) and (d) are zoomed from (a) and (c), respectively, with the gradients of RNN and LSTM for better visualization and comparison. The gradient of RNN decays quickly over time and the gradient of LSTM decays quickly over layer depth.

and the sequence length $T = 50$ and $T = 100$ were both tested in training and testing.

Table 3 shows the performance of the proposed IndRNN models in comparison with the existing methods, in terms of bits per character metric (BPC). Compared with the traditional RNN and LSTM models, it can be seen that the proposed IndRNN model achieved better performance. In addition, most IndRNN models in Table 3 except the one with 6 layers and 50 steps outperformed the popular language model Transformer. By comparing the performance of the proposed IndRNN model with different depth, it can be seen that a deeper residual IndRNN or a densely connected IndRNN further improves the performance over the relatively shallow (6 layers) model. Moreover, an improvement can be further achieved with longer temporal dependencies (from time step 50 to 100), indicating that the proposed IndRNN can effectively learn long-term patterns.

5.2.4 Word-level Penn Treebank

In this subsection, the performance of the proposed IndRNN on the word-level Penn Treebank dataset is evaluated. The test setting is similar to [18]. A 12-layer residual IndRNN with 2000 neurons in each layer and a densely connected IndRNN with the growth rate of 256 were used for test. An embedding layer with 600 neurons is used. The weight tying [60], [61] of the input embedding and the final output weight is adopted and accordingly the last transition layer contains 600 neurons, which processes the IndRNN features from 2000 to 600. For the residual IndRNN, dropout with a dropping probability of 0.45 was used among IndRNN layers while 0.5 was used for the last IndRNN layer. For the densely connected IndRNN, dropout is applied after each dense layer (0.5), each bottleneck layer (0.1) and each transition layer (0.4). Dropout of 0.65 is used after the embedding layer and the last transition layer, and dropout (0.2) on the embedding weight as in [19] is also applied for both IndRNN networks. The batch size is set to 128. Sequence length $T = 50$ was used in training and testing.

The results are shown in Table 4 in comparison with the existing methods. It can be seen that the proposed IndRNN model achieved better performance than the traditional RNN and LSTM models including the AWD-LSTM [19], which is heavily optimized with weight drop, activation regularization, etc. The results can be further enhanced by post-processing with techniques such as neural cache models [62], mixture of ssoftmaxes (MOS) [63] and dynamic evaluation [64]. Here, dynamic evaluation [64] is used as an example and the others are not further discussed. The results are shown in the end of Table 4 with a perplexity of 50.97, which is also better than the AWD-LSTM [19] with the dynamic evaluation. Notice that both res-IndRNN and dense-IndRNN models outperformed the Transformer as shown in Table 4.

TABLE 4: Results of word-level PTB for our proposed IndRNN model in comparison with results reported in the literature, in terms of perplexity.

	Test
RNN-LDA + KN-5 + cache [65]	92.0
Deep RNN [38]	107.5
CharCNN [66]	78.9
LSTM [18]	114.5
LSTM+Recurrent dropout [52]	87.0
LSTM+Zoneout [18]	77.4
LSTM+Variational Dropout [67]	73.4
Pointer Sentinel LSTM [68]	70.9
RHN [43]	65.4
Neural Architecture Search [59]	62.4
AWD-LSTM [19]	58.8
AWD-LSTM+Finetue [19]	57.3
AWD-LSTM+Finetue [19]+dynamic eval [64]	51.1
SRU ([31])	60.3
Transformer (reported in [26])	122.37
R-Transformer [26]	84.38
res-IndRNN (12 layers)	58.99
dense-IndRNN	55.24
dense-IndRNN+dynamic eval [64]	49.95

TABLE 5: Result comparison of the RNN based skeleton action recognition methods on NTU RGB+D dataset.

Method	CS	CV
1 Layer RNN (reported in [20])	56.02%	60.24%
2 Layer RNN (reported in [20])	56.29%	64.09%
1 Layer LSTM (reported in [20])	59.14%	66.81%
2 Layer LSTM (reported in [20])	60.09%	67.29%
1 Layer RNN	57.84%	64.28%
2 Layer RNN	62.17%	71.00%
4 Layer RNN	61.78%	68.28%
6 Layer RNN	58.23%	63.49%
1 Layer LSTM	66.01%	73.87%
2 Layer LSTM	67.93%	77.55%
4 Layer LSTM	66.02%	69.12%
6 Layer LSTM	58.50%	67.42%
IndRNN-tanh (4 layers)	76.56%	85.32%
IndRNN-tanh (6 layers)	80.36%	88.61%
IndRNN (4 layers)	77.23%	88.30%
IndRNN (6 layers)	80.44%	89.45%
res-IndRNN (100 layers)	83.26%	92.00%
dense-IndRNN	**83.38%**	**91.81%**

TABLE 6: Results of all skeleton based methods on NTU RGB+D dataset.

Method	CS	CV
SkeletonNet(CNN) [69]	75.94%	81.16%
JDM+CNN [70]	76.20%	82.30%
Clips+CNN+MTLN [71]	79.57%	84.83%
Enhanced Visualization+CNN [72]	80.03%	87.21%
HCN [73]	86.5%	91.1%
TCN + TTN [74]	77.55%	84.25%
STGCN [75]	81.5%	88.3%
PB-GCN [76]	**87.5%**	93.2%
1 Layer PLSTM [20]	62.05%	69.40%
2 Layer PLSTM [20]	62.93%	70.27%
ST-LSTM + Trust Gate [1]	69.2%	77.7%
JL_d+RNN [22]	70.26%	82.39%
STA-LSTM [21]	73.40%	81.20%
Pose conditioned STA-LSTM [77]	77.10%	84.50%
R-Transformer [26]	75.69%	81.56%
dense-IndRNN-aug	86.70%	**93.97%**

5.3 Skeleton based Action Recognition

5.3.1 NTU RGB+D dataset

In this subsection, the skeleton based action recognition [20], [78] is used to evaluate the performance of the proposed IndRNN. The widely used NTU RGB+D dataset [20]. The NTU RGB+D dataset contains 56880 sequences (over 4 million frames) of 60 action classes, including Cross-Subject (CS) (40320 and 16560 samples for training and testing, respectively) and Cross-View (CV) (37920 and 18960 samples for training and testing, respectively) evaluation protocols [20]. In each evaluation protocol, 5% of the training data (randomly selected) was used for evaluation as suggested in [20]. The joint coordinates of two subject skeletons were used as input. If only one is present, the second was set as zero. For this dataset, when multiple skeletons are present in the scene, the skeleton identity captured by the Kinect sensor may be changed over time. Therefore, an alignment process was first applied to keep the same skeleton saved in the same data array over time. 20 frames were sampled from each instance as one input in the same way as in [79] and batch size was set to 128.

First, IndRNN with different settings are evaluated first, including different IndRNN architectures (plain IndRNN and dense IndRNN), different depths (4 layers and 6 layers) and different activation functions (ReLU and Tanh). To be specific, a four-layer IndRNN and a six-layer IndRNN with 512 hidden neurons were both tested to demonstrate the effectiveness of the proposed IndRNN with a plain IndRNN architecture. ReLU and tanh are both evaluated to show the performance of IndRNN under different activation functions. Then a densely connected IndRNN is used to further validate the effectiveness of a deep architecture. The growth rate is set to 48. For the four-layer IndRNN and six-layer IndRNN, dropout [67] was applied after each IndRNN layer with a dropping probability of 0.4 and 0.3 for models with ReLU and tanh, respectively, for both CS and CV settings. For the densely connected IndRNN, dropout is applied after the input processing layer (0.5), each dense layer (0.5), each bottleneck layer (0.1) and each transition layer (0.3). For comparison, we also included the deep RNN and LSTM of different layers to show the difference in constructing deep RNNs. The settings of the RNN and LSTM models are similar to [20] with 128 neurons for LSTM and 512 neurons for RNN (both with dropout 0.5). 1 layer, 2 layer, 4 layers and 6 layers are used for evaluation.

The results of different settings are shown in Table 5. By comparing the performance of RNN, LSTM and plain IndRNN, it can be seen that the proposed IndRNN greatly improves the performance over the conventional RNN models on the same task. For CS, LSTM of 2 layers can only achieve accuracies of 67.93% while a 4-layer IndRNN already achieved 78.58%, improving performance by over 10%. For CV, LSTM of 2 layers only achieved accuracies of 77.55% while 4-layer IndRNN already achieved 83.75%, improving performance by over 6%. The results of RNN and LSTM in our experiments are higher than those reported in [20] and here we use the better results for comparison. On the other hand, by comparing the performance of RNN, LSTM and IndRNN with different layers, it can be seen that the performance of RNN and LSTM cannot

TABLE 7: Ablation study on the γ (recurrent weight constraint).

γ	1	2	10	No Constraint
	80.82%	80.68%	80.43%	80.24%

be further improved by simply increasing the number of layers, which is the same as demonstrated in [20], [79]. On the contrary, by increasing the 4-layer IndRNN to a 6-layer IndRNN, the performance is further improved to 81.80% and 87.97% for CS and CV, respectively. Moreover, with a deep densely connected IndRNN, the performance is further improved to 84.88% and 90.34% for CS and CV, respectively, which is significantly better than the conventional RNN and LSTM methods. From the perspective of different activation functions, it can be seen that IndRNN with tanh also achieves much better performance than the conventional LSTM, although slightly worse than IndRNN with ReLU. Considering the wide use and spread of the non-saturated activation function such as ReLU, we focus on IndRNN with ReLU. More importantly, ReLU does not introduce any weight decay when activated, making the gradient vanishing problem greatly alleviated. Therefore, we only presented the results of IndRNN with tanh in this experiment and not further span on it.

To further demonstrate that IndRNN can construct very deep networks, a 100 layers IndRNN with residual connections is further tested. The setting is the same as above, only the learning rate is set to $6 * 10^{-4}$ to reduce overfitting. The results are also shown in Table 5. It can be seen that a deeper IndRNN still can be trained robustly with improved performance. We further conducted an ablation study on the γ, the largest gradient magnitude to avoid gradient exploding, which is the only hyperparameter introduced by IndRNN. It also controls the range of the recurrent weights by $|u_n| \leq \sqrt[(T-t)]{\gamma}$. Experiments with different γs are conducted on the CS setting with a 6-layer IndRNN and results are shown in Table 7. It can be seen that the results are relatively close, indicating that IndRNN is not very sensitive to the choice of γ when processing short sequences. Moreover, γ can be simply set to 1, which shows a good performance and avoids the gradient vanishing and exploding as described in Subsection 3.1. Accordingly, the recurrent weights are constrained to be in the range of $[-1, 1]$.

The final results on the NTU RGB+D dataset are shown in Table 6 including comparisons with the existing methods. Considering that feature augmentation including the geometric features [22] and temporal feature augmentation [76] have shown to be useful, feature augmentation is further adopted on top of the densely connected IndRNN. In addition, we also applied the data augmentation in the time dimension where sequences of length 10-30 are randomly sampled as input. The result is also shown in Table 6. It can be seen that the performance is further improved and better than the RNN based methods even equipped with enhancing techniques such as attention.

It is noted that the performance is slightly worse than the GCN based models [76], which applied graph convolution to process the input spatially. However, the performance of IndRNN is only achieved with the coordinates as input without any spatial processing techniques. It should be pointed out that GCN can be used together with IndRNN to take the advantages of both networks, for instance, skeleton features instead of simple coordinates can be extracted using a GCN in a local spatial/temporal window and used as input to an IndRNN. This will be studied in the future.

TABLE 8: Results of all skeleton based methods on NTU RGB+D 120 dataset.

Method	CS	CV
Part-aware LSTM [20] (reported in [80])	25.5%	26.3%
Soft RNN [81] (reported in [80])	36.3%	44.9%
ST-LSTM [79]	55.7%	57.9%
GCA-LSTM [82]	58.3%	59.2%
Two-Stream Attention LSTM [83]	61.2%	63.3%
Body Pose Evolution Map [84]	64.6%	66.9%
SkeleMotion [85]	66.9%	67.7%
IndRNN (4 layers)	70.01%	72.05%
IndRNN (6 layers)	72.39%	75.04%
dense-IndRNN	74.62%	77.37%
dense-IndRNN-aug	**76.55%**	**79.18%**

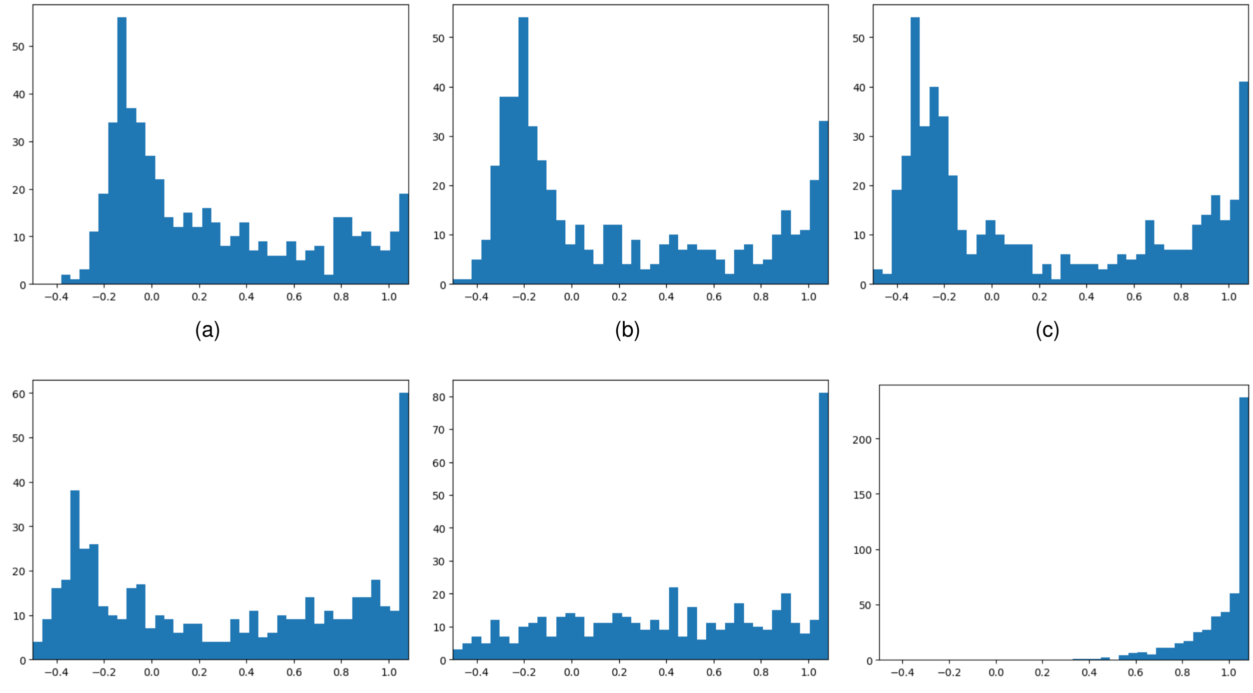
5.3.2 NTU RGB+D 120 dataset

The new NTU RGB+D 120 dataset [80], enhanced on the NTU RGB+D dataset is also adopted for evaluation. It contains 114480 video samples (over 8 million frames) from 120 different action classes, by adding another 57600 samples and another 60 classes from the NTU RGB+D dataset. Both the CS and CV evaluation protocols from the NTU RGB+D dataset are used. The other settings include the training and testing setups are the same as the one used in the above NTU RGB+D dataset.

The results on the NTU RGB+D 120 dataset are shown in Table 8 with comparisons against the existing methods. Similar to the results on the NTU RGB+D dataset, the proposed IndRNN achieves much better performance against the conventional RNN/LSTM based methods. It reaches 74.60% and 77.37% in terms of accuracy for the CS and CV settings, respectively, which is much better than the 61.2% and 63.3% for attention enhanced LSTM methods and the state-of-the-art methods using CNN with accuracy of 66.9% and 67.7%.

5.3.3 Visualization of the Recurrent Weight and Memory

With the mapping between the recurrent weight and memory shown in subsection 3.2, the memory learned by the network for the task can be understood by visualizing the recurrent weights. Fig. 6 shows the histograms of the learned recurrent weights (the above six-layer IndRNN obtained under the CS setting for the NTU RGB+D dataset). It can be seen that for the first 5 layers, the learned recurrent weights severely deviate from the uniformly initialized recurrent weights. Especially for the higher layers such as layer 4 and layer 5, the recurrent weights are mostly around 1, learning the long-term memory. On the contrary, for shallower layers such as layer 0 and layer 1, in addition to the large recurrent weights around 1, there is also a group of weights being close to 0, learning the very short-term memory. This makes sense since the shallow layers process the very basic features in a short period while as the layers increase, high-level features in a longer period are



(a) (b) (c)

(d) (e) (f)

Fig. 6: Histograms of the learned recurrent weights for different layers, where (a)-(f) correspond to the 1st-6th layers. x axis and y axis represent the value of the recurrent weight and the frequency.

processed. Surprisingly, the number of recurrent weights around 0.5 is small indicating that only a small number of neurons keep mid-range memory. Another interesting fact is that although the recurrent weights are initialized to be positive, part of the recurrent weights are learned to be negative. Note that for keeping different ranges of memories, only the absolute value of the recurrent weights are relevant and thus the negative recurrent weights can still keep different ranges of memories. However, negative recurrent weights may cause oscillation as shown in the gradient backpropagation process. It is hard to intuitively understand how the negative recurrent weights work in the network yet. For the last layer, it can be seen that most of the recurrent weights stay closely to 1 to keep long-term memory, which agrees with the previous assumption shown at the end of Section 3.2.

6 CONCLUSION

In this paper, we proposed an independently recurrent neural network (IndRNN) with a new recurrent connection using the Hadamard product, making neurons in one layer independent of each other. It effectively solves the gradient vanishing and exploding problems by regulating the recurrent weights, which makes it able to be efficiently process very long sequences. Different deep IndRNN architectures, including the basic IndRNN, residual IndRNN and densely connected IndRNN, have been proposed, which can be much deeper than the traditional RNNs. With the gradient better regulated in IndRNN, non-saturated activation functions such as ReLU can be used and trained very robustly. Moreover, with the neurons in one layer independent from each other, each neuron can be better interpreted without effects from others. Also, the relationship between the memory and the recurrent weight has been established through gradient backpropagation, and the learned memories can be understood by investigating the recurrent weights. Experiments on multiple fundamental tasks have verified the advantages of the proposed IndRNN over existing RNN models.

REFERENCES

[1] J. Liu, A. Shahroudy, D. Xu, A. C. Kot, and G. Wang, “Skeleton-based action recognition using spatio-temporal lstm network with trust gates,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 3007–3021, 2018.

[2] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, and N. Zheng, “View adaptive neural networks for high performance skeleton-based human action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 1963–1978, 2018.

[3] X.-Y. Zhang, F. Yin, Y.-M. Zhang, C.-L. Liu, and Y. Bengio, “Drawing and recognizing chinese characters with recurrent neural network,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 849–862, 2016.

[4] Q. Wu, C. Shen, P. Wang, A. Dick, and A. van den Hengel, “Image captioning and visual question answering based on attributes and external knowledge,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 1367–1381, 2016.

[5] B. Shuai, Z. L. Zuo, B. Wang, and G. Wang, “Scene segmentation with dag-recurrent neural networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 1480–1493, 2018.

[6] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, “A fully trainable network with rnn-based pooling,” *Neurocomputing*, 2019.

[7] M. I. Jordan, “Serial order: A parallel distributed processing approach,” *Advances in psychology*, vol. 121, pp. 471–495, 1997.

[8] B. Neyshabur, Y. Wu, R. R. Salakhutdinov, and N. Srebro, “Path-normalized optimization of recurrent neural networks with relu activations,” in *Advances in Neural Information Processing Systems*, 2016, pp. 3477–3485.

[9] D. Krueger and R. Memisevic, "Regularizing rnns by stabilizing activations," in *Proceeding of the International Conference on Learning Representations*, 2016.

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

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

[12] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2017.

[13] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," 1991.

[14] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 2342–2350.

[15] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *Proceedings of the Empirical Methods in Natural Language Processing*, 2014.

[16] J. Kim, M. El-Khamy, and J. Lee, "Residual lstm: Design of a deep recurrent architecture for distant speech recognition," in *INTERSPEECH*, 2017.

[17] J. Li, C. Liu, and Y. Gong, "Layer trajectory lstm," *ArXiv*, vol. abs/1808.09522, 2018.

[18] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, H. Larochelle, A. Courville *et al.*, "Zoneout: Regularizing rnns by randomly preserving hidden activations," *Proceeding of the International Conference on Learning Representations*, 2016.

[19] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *international conference on learning representations*, 2018.

[20] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "Ntu rgb+ d: A large scale dataset for 3d human activity analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1010–1019.

[21] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu, "An end-to-end spatio-temporal attention model for human action recognition from skeleton data." in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017, pp. 4263–4270.

[22] S. Zhang, X. Liu, and J. Xiao, "On geometric features for skeleton-based action recognition using multilayer lstm networks," in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 148–157.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[24] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (indrnn): Building a longer and deeper rnn," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5457–5466.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, pp. 6000–6010, 2017.

[26] Z. Wang, Y. Ma, Z. Liu, and J. Tang, "R-transformer: Recurrent neural network enhanced transformer," *ArXiv*, vol. abs/1907.05572, 2019.

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

[28] G.-B. Zhou, J. Wu, C.-L. Zhang, and Z.-H. Zhou, "Minimal gated unit for recurrent neural networks," *International Journal of Automation and Computing*, vol. 13, no. 3, pp. 226–234, 2016.

[29] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and trainability in recurrent neural networks," *Proceeding of the International Conference on Learning Representations*, 2016.

[30] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," *Proceeding of the International Conference on Learning Representations*, 2016.

[31] T. Lei and Y. Zhang, "Training rnns as fast as cnns," *arXiv preprint arXiv:1709.02755*, 2017.

[32] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," *Proceeding of the International Conference on Learning Representations*, 2017.

[33] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," *Proceedings of the International Conference on Machine Learning*, 2015.

[34] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas, "Full-capacity unitary recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4880–4888.

[35] J. Zhang, Y. Lin, Z. Song, and I. S. Dhillon, "Learning long term dependencies via fourier recurrent units," *Proceedings of the International Conference on Machine Learning*, 2018.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks." in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, no. 2, 2017, p. 3.

[38] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *Proceeding of the International Conference on Learning Representations*, 2013.

[39] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *International Conference on Machine Learning*, 2015, pp. 2067–2075.

[40] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[41] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 550–558.

[42] S. Pradhan and S. Longpre, "Exploring the depths of recurrent neural networks with stochastic residual learning," *Report*.

[43] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," *Proceedings of the International Conference on Machine Learning*, 2016.

[44] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 1017–1024.

[45] Y. Wu, S. Zhang, Y. Zhang, Y. Bengio, and R. R. Salakhutdinov, "On multiplicative integration with recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 2856–2864.

[46] N. Kalchbrenner, I. Danihelka, and A. Graves, "Grid long short-term memory," *Proceeding of the International Conference on Learning Representations*, 2015.

[47] A. Graves, S. Fernández, and J. Schmidhuber, "Multi-dimensional recurrent neural networks," in *Proceedings of the 17th International Conference on Artificial Neural Networks*. Springer-Verlag, 2007, pp. 549–558.

[48] B. Parlett, "Laguerre's method applied to the matrix eigenvalue problem," *Mathematics of Computation*, vol. 18, no. 87, pp. 464–485, 1964.

[49] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceeding of the International Conference on Learning Representations*, 2014.

[52] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent dropout without memory loss," *Proceedings of the International Conference on Computational Linguistics*, 2016.

[53] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville, "Recurrent batch normalization," *Proceeding of the International Conference on Learning Representations*, 2016.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[55] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. Cernocky, "Subword language modeling with neural networks," *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

[56] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[57] J. Chung, S. Ahn, and Y. Bengio, “Hierarchical multiscale recurrent neural networks,” *Proceeding of the International Conference on Learning Representations*, 2016.

[58] A. Mujika, F. Meier, and A. Steger, “Fast-slow recurrent neural networks,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5917–5926.

[59] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *Proceeding of the International Conference on Learning Representations*, 2016.

[60] H. Inan, K. Khosravi, and R. Socher, “Tying word vectors and word classifiers: A loss framework for language modeling,” *Proceeding of the International Conference on Learning Representations*, 2016.

[61] O. Press and L. Wolf, “Using the output embedding to improve language models,” in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, vol. 2, 2017, pp. 157–163.

[62] E. Grave, A. Joulin, and N. Usunier, “Improving neural language models with a continuous cache,” *Proceeding of the International Conference on Learning Representations*, 2017.

[63] Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen, “Breaking the softmax bottleneck: A high-rank rnn language model,” *Proceeding of the International Conference on Learning Representations*, 2017.

[64] B. Krause, E. Kahembwe, I. Murray, and S. Renals, “Dynamic evaluation of neural sequence models,” *Proceedings of the International Conference on Machine Learning*, pp. 2766–2775, 2018.

[65] T. Mikolov and G. Zweig, “Context dependent recurrent neural network language model,” *IEEE Spoken Language Technology Workshop (SLT)*, vol. 12, pp. 234–239, 2012.

[66] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, “Character-aware neural language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016, pp. 2741–2749.

[67] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” in *Advances in neural information processing systems*, 2016, pp. 1019–1027.

[68] S. Merity, C. Xiong, J. Bradbury, and R. Socher, “Pointer sentinel mixture models,” *Proceeding of the International Conference on Learning Representations*, 2016.

[69] Q. Ke, S. An, M. Bennamoun, F. Sohel, and F. Boussaid, “Skeletonnet: Mining deep part features for 3-d action recognition,” *IEEE Signal Processing Letters*, vol. 24, no. 6, pp. 731–735, 2017.

[70] C. Li, Y. Hou, P. Wang, and W. Li, “Joint distance maps based action recognition with convolutional neural networks,” *IEEE Signal Processing Letters*, vol. 24, no. 5, pp. 624–628, 2017.

[71] Q. Ke, M. Bennamoun, S. An, F. Sohel, and F. Boussaid, “A new representation of skeleton sequences for 3d action recognition,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[72] M. Liu, H. Liu, and C. Chen, “Enhanced skeleton visualization for view invariant human action recognition,” *Pattern Recognition*, vol. 68, pp. 346–362, 2017.

[73] C. Li, Q. Zhong, D. Xie, and S. Pu, “Co-occurrence feature learning from skeleton data for action recognition and detection with hierarchical aggregation,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 7 2018, pp. 786–792.

[74] L. S, Q. Wang, and P. Turaga, “Temporal transformer networks: Joint learning of invariant and discriminative time warping,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 426–12 435.

[75] S. Yan, Y. Xiong, D. Lin, and xiaoou Tang, “Spatial temporal graph convolutional networks for skeleton-based action recognition,” *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7444–7452, 2018.

[76] K. C. Thakkar and P. J. Narayanan, “Part-based graph convolutional network for action recognition.” *Proceedings of the British Machine Vision Conference*, p. 270, 2018.

[77] F. Baradel, C. Wolf, and J. Mille, “Pose-conditioned spatio-temporal attention for human action recognition,” *arXiv preprint arXiv:1703.10106*, 2017.

[78] A. Shahroudy, T.-T. Ng, Y. Gong, and G. Wang, “Deep multimodal feature analysis for action recognition in rgb+d videos,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 1045–1058, 2018.

[79] J. Liu, A. Shahroudy, D. Xu, and G. Wang, “Spatio-temporal lstm with trust gates for 3d human action recognition,” in *European Conference on Computer Vision*. Springer, 2016, pp. 816–833.

[80] J. Liu, A. Shahroudy, M. Perez, G. Wang, L. yu Duan, and A. C. Kot, “Ntu rgb+d 120: A large-scale benchmark for 3d human activity understanding,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 10, 2019.

[81] J. fang Hu, W.-S. Zheng, L. Ma, G. Wang, J.-H. Lai, and J. Zhang, “Early action prediction by soft regression,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 11, pp. 2568–2583, 2019.

[82] J. Liu, G. Wang, P. Hu, L. yu Duan, and A. C. Kot, “Global context-aware attention lstm networks for 3d action recognition,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3671–3680, 2017.

[83] J. Liu, G. Wang, L. yu Duan, K. Abdiyeva, and A. C. Kot, “Skeleton-based human action recognition with global context-aware attention lstm networks,” *IEEE Transactions on Image Processing*, vol. 27, pp. 1586–1599, 2018.

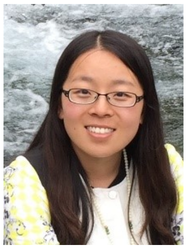
[84] M. Liu and J. Yuan, “Recognizing human actions as the evolution of pose estimation maps,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1159–1168, 2018.

[85] C. Caetano, J. S. de Souza, F. Brémond, J. A. dos Santos, and W. R. Schwartz, “Skelemotion: A new representation of skeleton joint sequences based on motion information for 3d action recognition,” *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–8, 2019.

Shuai Li is currently with the School of Control Science and Engineering, Shandong University (SDU), China, as a Professor and QiLu Young Scholar. He was with the School of Information and Communication Engineering, University of Electronic Science and Technology of China (UESTC), China, as an Associate Professor from 2018-2020. He received his Ph.D. degree from the University of Wollongong, Australia, in 2018. His research interests include image/video coding, 3D video processing and computer vision. He was a co-recipient of two best paper awards at the IEEE BMSB 2014 and IIH-MSP 2013, respectively.

Wanqing Li (M’97-SM’05) received his Ph.D. in electronic engineering from The University of Western Australia. He was a Principal Researcher (98-03) at Motorola Research Lab and a visiting researcher (08, 10 and 13) at Microsoft Research US. He is currently an Associate Professor and Co-Director of Advanced Multimedia Research Lab (AMRL) of UOW, Australia. His research areas are machine learning, 3D computer vision, 3D multimedia signal processing and medical image analysis. Dr. Li currently serves as an Associate Editor for IEEE Transactions on Circuits and Systems for Video Technology and IEEE Transactions on Multimedia. He was an Associator for Journal of Visual Communication and Image Representation.

Chris Cook is an electrical engineer with a BSc and BE from the University of Adelaide, and he received his PhD from the University of New South Wales in 1976. He is a Fellow of the Institute of Engineers Australia and a Chartered Engineer. He has worked for Marconi Avionics in the UK, for GEC Australia and was the founding Managing Director of the Automation and Engineering Applications Centre Ltd which designed and built industrial automation systems. He recently retired after 14 years as Executive Dean of Engineering and Information Sciences at the University of Wollongong but remains an active researcher in robotics and machine intelligence.



Yanbo Gao is currently with the School of Software, Shandong University (SDU), Jinan, China, as an Associate Professor. She was with the School of Information and Communication Engineering, University of Electronic Science and Technology of China (UESTC), Chengdu, China, as a Post-doctor from 2018-2020. She received her Ph.D. degree from UESTC in 2018. Her research interests include video coding, 3D video processing and light field image coding. She was a co-recipient of the best student paper awards at the IEEE BMSB 2018.

APPENDIX A
RELATIONSHIP BETWEEN INDRNN AND RNN

The relationship between IndRNN and the traditional RNN is illustrated in the following, where we show that under certain circumstances the traditional RNN is only a special case of a two-layers IndRNN. First, from the perspective of number of neurons, as shown in 3.3, for a N-neuron RNN network with input of dimension M, the number of parameters using traditional RNN is $M \times N + N \times N$, while the number of parameters using one-layer IndRNN is $M \times N + N$. For a two-layers IndRNN where both layers consist of N neurons, the number of parameters is $M \times N + N \times N + 2 \times N$, which is of a similar order to the traditional RNN. Therefore, in the following we compare a two-layer IndRNN with a single layer RNN. For simplicity, the bias term is ignored for both IndRNN and traditional RNN.

Assume a simple N-neuron two-layer network where the recurrent weights for the second layer are zero which means the second layer is just a fully connected layer shared over time. The Hadamard product ($\mathbf{u} \odot \mathbf{h}_{t-1}$) can be represented in the form of matrix product by $diag(u_1, u_2, \ldots, u_N)\mathbf{h}_{t-1}$. In the following, $diag(u_1, u_2, \ldots, u_N)$ is shortened as $diag(u_i)$. Assume that the activation function is a linear function $\sigma(x) = x$. The first and second layers of a two-layer IndRNN can be represented by (5) and (6), respectively.

$$\mathbf{h}_{f,t} = \mathbf{W}_f \mathbf{x}_{f,t} + diag(u_{fi})\mathbf{h}_{f,t-1} \tag{5}$$

$$\mathbf{h}_{s,t} = \mathbf{W}_s \mathbf{h}_{f,t} \tag{6}$$

Assuming $\mathbf{W}_s$ is invertible, then

$$\mathbf{W}_s^{-1}\mathbf{h}_{s,t} = \mathbf{W}_f \mathbf{x}_{f,t} + diag(u_{fi})\mathbf{W}_s^{-1}\mathbf{h}_{s,t-1} \tag{7}$$

Thus

$$\mathbf{h}_{s,t} = \mathbf{W}_s\mathbf{W}_f \mathbf{x}_{f,t} + \mathbf{W}_s diag(u_{fi})\mathbf{W}_s^{-1}\mathbf{h}_{s,t-1} \tag{8}$$

By assigning $\mathbf{U} = \mathbf{W}_s diag(u_{fi})\mathbf{W}_s^{-1}$ and $\mathbf{W} = \mathbf{W}_s\mathbf{W}_f$, it becomes

$$\mathbf{h}_t = \mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} \tag{9}$$

which is a traditional RNN. Note that this only imposes the constraint that the recurrent weight ($\mathbf{U}$) is diagonalizable. Therefore, the simple two-layer IndRNN network can represent a traditional RNN network with a diagonalizable recurrent weight ($\mathbf{U}$). In other words, under linear activation, a traditional RNN with a diagonalizable recurrent weight ($\mathbf{U}$) is a special case of a two-layer IndRNN.

It is known that a non-diagonalizable matrix can be made diagonalizable with a perturbation matrix composed of small entries. A stable RNN network needs to be robust to small perturbations (in order to deal with precision errors for example). It is possible to find an RNN network with a diagonalizable recurrent weight matrix to approximate a stable RNN network with a non-diagonalizable recurrent weight matrix. Therefore, a traditional RNN with a linear activation is a special case of a two-layer IndRNN. For a traditional RNN with a nonlinear activation function, its relationship with the proposed IndRNN is yet to be established theoretically. However, we have shown empirically that the proposed IndRNN can achieve better performance than a

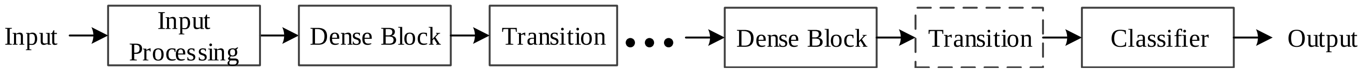

Input
Input Processing
Dense Block
Transition
Dense Block
Transition
Classifier
Output

Fig. 7: Framework of the densely connected IndRNN used in the experiments.

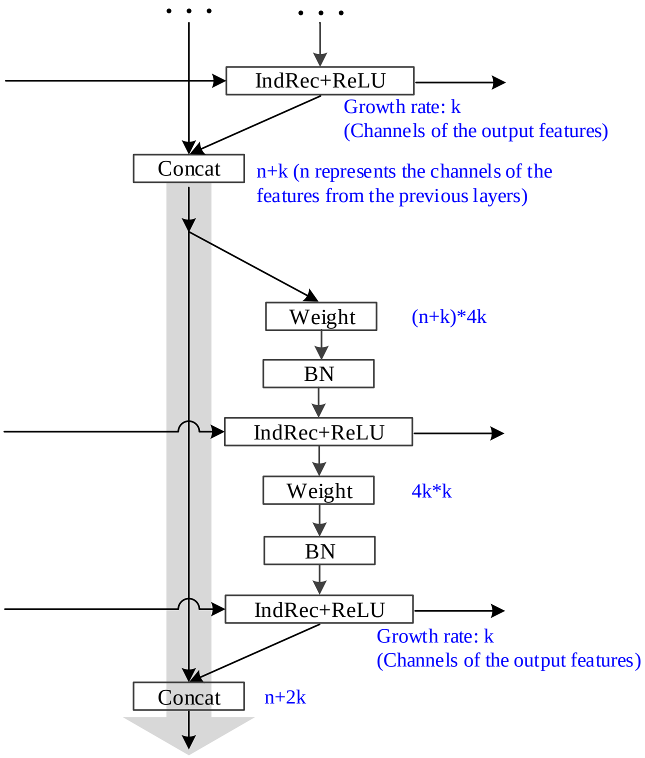

IndRec+ReLU
Growth rate: k
(Channels of the output features)
Concat
n+k (n represents the channels of the features from the previous layers)
Weight
(n+k)*4k
BN
IndRec+ReLU
Weight
4k*k
BN
IndRec+ReLU
Growth rate: k
(Channels of the output features)
Concat
n+2k

Fig. 8: Detailed illustration of the dense IndRNN block.

traditional RNN with a nonlinear activation function in the experiments. Therefore, the cross-channel information can be well explored with a multiple-layer IndRNN although IndRNN neurons are independent of each other in each layer. In all, multiple-layer IndRNNs can be used in place of the traditional RNNs.

APPENDIX B
DESCRIPTION OF THE DENSELY CONNECTED INDRNN SETTING

The framework of the densely connected IndRNN process is shown in Fig. 7, which generally follows the conventional denseCNN [37]. First, the input is processed into features usually with the embedding layer (in the language modelling task) or one IndRNN layer (in the other recognition tasks). Then the initial features are processed with several dense IndRNN blocks and transition layers. As stated in Subsection 5.1, three dense IndRNN blocks are used with 8, 6 and 4 dense IndRNN layers, respectively. While in the conventional denseCNN [37] the last transition layer is not needed after the final dense block, we find that it usually improves the performance in our experiments. Therefore, the last transition layer is also added in our dense-IndRNN. Finally, a classifier is added in the end for recognition.

Fig. 8 further shows the details of the dense IndRNN block. Each dense IndRNN layer produces k channels of output features, termed as growth rate. After one dense IndRNN layer, the features are increased to $n+k$ channels, where n represents the channels of the features from the previous layers. For the next IndRNN layer, the first weight processes the $n+k$ dimensional to a $4k$-channel feature, then further processed by BN and IndRec+ReLU. The second weight processes the $4k$-channel feature to k-channel and further processed for output. Concatenated with the input $n+k$-channel feature, the final output feature of the dense IndRNN layer becomes $n+2k$ channels. For the transition layers, it processes the feature (N channels) from the dense IndRNN block and reduces it to half ($N/2$ channels). The key component of dense-IndRNN different from the residual-IndRNN is the concatenation operation of the dense layers and the shortcut, which allows the features to be reused in the following layers.