Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN

Shuai Li [*], Wanqing Li [*], Chris Cook [*], Ce Zhu [†], Yanbo Gao [†]
[*]School of Computing and Information Technology, University of Wollongong
[†]School of Electronic Engineering, University of Electronic Science and Technology of China
{sl669,wanqing,ccook}@uow.edu.au,eczhu@uestc.edu.cn,yanbogao@std.uestc.edu.cn

Abstract

Recurrent neural networks (RNNs) have been widely used for processing sequential data. However, RNNs are commonly difficult to train due to the well-known gradient vanishing and exploding problems and hard to learn long-term patterns. Long short-term memory (LSTM) and gated recurrent unit (GRU) were developed to address these problems, but the use of hyperbolic tangent and the sigmoid action functions results in gradient decay over layers. Consequently, construction of an efficiently trainable deep network is challenging. In addition, all the neurons in an RNN layer are entangled together and their behaviour is hard to interpret. To address these problems, a new type of RNN, referred to as independently recurrent neural network (IndRNN), is proposed in this paper, where neurons in the same layer are independent of each other and they are connected across layers. We have shown that an IndRNN can be easily regulated to prevent the gradient exploding and vanishing problems while allowing the network to learn long-term dependencies. Moreover, an IndRNN can work with non-saturated activation functions such as relu (rectified linear unit) and be still trained robustly. Multiple IndRNNs can be stacked to construct a network that is deeper than the existing RNNs. Experimental results have shown that the proposed IndRNN is able to process very long sequences (over 5000 *time steps), can be used to construct very deep networks (*21 *layers used in the experiment) and still be trained robustly. Better performances have been achieved on various tasks by using IndRNNs compared with the traditional RNN and LSTM.*

1. Introduction

Recurrent neural networks (RNNs) [16] have been widely used in sequence learning problems such as action recognition [8], scene labelling [4] and language processing [5], and have achieved impressive results. Compared with the feed-forward networks such as the convolutional neural networks (CNNs), a RNN has a recurrent connection where the last hidden state is an input to the next state. The update of states can be described as 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. σ is an element-wise activation function of the neurons, and N is the number of neurons in this RNN layer.

Training of the RNNs suffers from the gradient vanishing and exploding problem due to the repeated multiplication of the recurrent weight matrix. Several RNN variants such as the long short-term memory (LSTM) [10, 17] and the gated recurrent unit (GRU) [5] 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. Consequently, construction and training of a deep LSTM or GRU based RNN network is practically difficult. By contrast, existing CNNs using non-saturated activation function such as relu can be stacked into a very deep network (e.g. over 20 layers using the basic convolutional layers and over 100 layers with residual connections [12]) and be still trained efficiently. Although residual connections have been attempted for LSTM models in several works [44, 36], there have been no significant improvement (mostly due to the reason that gradient decays in LSTM with the use of the hyperbolic tangent and the sigmoid functions as mentioned above).

Moreover, 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 entangles all the neurons. This makes it hard to interpret and understand the roles of the trained neurons (e.g., what patterns each neuron responds to) since the simple visualization of the outputs of individual neurons [18] is hard to ascertain the function of one neuron without considering the others.

In this paper, a new type of RNN, referred to as independently recurrent neural network (IndRNN), is proposed. 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 RNN including:

- The gradient backpropagation through time can be regulated to effectively address the gradient vanishing and exploding problems.
- Long-term memory can be kept with IndRNNs to process long sequences. Experiments have demonstrated that an IndRNN can well process sequences over 5000 steps while LSTM could only process less than 1000 steps.
- An IndRNN can work well with non-saturated function such as relu as activation function and be trained robustly.
- Multiple layers of IndRNNs can be efficiently stacked, especially with residual connections over layers, to increase the depth of the network. An example of 21 layer-IndRNN is demonstrated in the experiments for language modelling.
- Behaviour of IndRNN neurons in each layer are easy to interpret due to the independence of neurons in each layer.

Experiments have demonstrated that IndRNN performs much better than the traditional RNN and LSTM models on the tasks of the adding problem, sequential MNIST classification, language modelling and action recognition.

2. Related Work

To address the gradient exploding and vanishing problems in RNNs, variants of RNNs have been proposed and typical ones are the long short-term memory (LSTM) [14], and the gated recurrent unit (GRU) [5]. Both LSTM and GRU enforce a constant error flow over time steps and use gates on the input and the recurrent input to regulate the information flow through the network. However, the use of gates makes the computation not parallelable and thus increases the computational complexity of the whole network. To process the states of the network over time in parallel, the recurrent connections are fixed in [3, 28]. While this strategy greatly simplifies the computational complexity, it reduces the capability of their RNNs since the recurrent connections are no longer trainable. In [1, 43], a unitary evolution RNN was proposed where the unitary recurrent weights are defined empirically. In this case, the norm of the backpropagated gradient can be bounded without exploding. By contrast, the proposed IndRNN solves the gradient exploding and vanishing problems without losing the power of trainable recurrent connections and without involving gate parameters.

In addition to changing the form of the recurrent neurons, 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 also reported in addressing the gradient problems. In [26], 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 [41], the recurrent weight matrix was further suggested to be a positive definite matrix with the highest eigenvalue of unity and all the remainder eigenvalues less than 1. In [33], the geometry of RNNs was investigated and a path-normalized optimization method for training was proposed for RNNs with relu activation. In [24], 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 the gradient exploding, they are not able to completely avoid the problem (the eigenvalues of the recurrent weight matrix may still be larger than 1 in the process of training). Moreover, the training of an IRNN is very sensitive to the learning rate. When the learning rate is large, the gradient is likely to explode. The proposed IndRNN solves gradient problems by making the neurons independent and constraining the recurrent weights. 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).

On the other hand, comparing with the deep CNN architectures which could be over 100 layers such as the residual CNN [12] and the pseudo-3D residual CNN (P3D) [37], most of the existing RNN architectures only consist of several layers (2 or 3 for example [23, 39, 26]). This is mostly due to the gradient vanishing and exploding problems which result in the difficulty in training a deep RNN. Since all the gate functions, input and output modulations in LSTM employ sigmoid or hyperbolic tangent functions as the activation function, it suffers from the gradient vanishing problem over layers when multiple LSTM layers are stacked into a deep model. Currently, a few models were reported that employ residual connections [12] between LSTM layers to make the network deeper [44]. However, as shown in [36], the deep LSTM model with the residual connections does not efficiently improve the performance. This may be partly due to the gradient decay over LSTM layers. On the contrary, for each time step, the proposed IndRNN with relu works in a similar way as CNN. Multiple layers of IndRNNs can be stacked and be efficiently combined with residual connections, leading to a deep RNN.

3. Independently Recurrent Neural Network

In this paper, we propose an independently recurrent neural network (IndRNN). It can be described as:

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

where recurrent weight $\mathbf{u}$ is a vector and $\odot$ represents Hadamard product. Each neuron in one layer is independent from others and connection between neurons can be achieved by stacking two or more layers of IndRNNs as presented later. 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 $\mathbf{w}_n$ and u_n are the n-th row of the input weight and recurrent weight, respectively. 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. Conventionally, a RNN is treated as multiple layer perceptrons over time where the parameters are shared. Different from the conventional RNNs, the proposed IndRNN provides a new perspective of recurrent neural networks as independently aggregating spatial patterns (i.e. through w) over time (i.e. through u). The correlation among different neurons can be exploited by stacking two or multiple layers. In this case, each neuron in the next layer processes the outputs of all the neurons in the previous layer.

The gradient backpropagation through time for an IndRNN and how it addresses the gradient vanishing and exploding problems are described in the next Subsection 3.1. Details on the exploration of cross-channel information are explained in Subsection 4. Different deeper and longer IndRNN network architectures are discussed in Subsection 4.1.

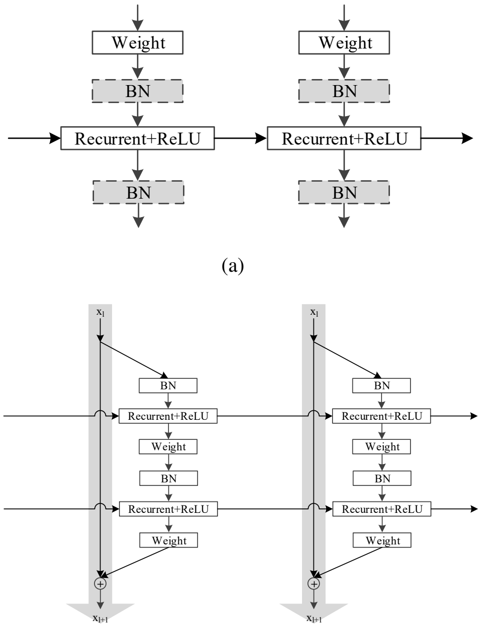
3.1. Backpropagation Through Time for An IndRNN

For the gradient backpropagation through time in each layer, the gradients of an IndRNN can be calculated independently for each neuron since there are no interactions among them in one layer. 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

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

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 [34]). Thus the training of an IndRNN is more robust than a traditional RNN. To solve the gradient exploding and vanishing problem over time, we only need to regulate the exponential term "$u_n^{T-t}\prod_{k=t}^{T-1}\sigma'_{n,k+1}$" to an appropriate range. This is further explained in the following together with keeping long and short memory in an IndRNN.

To keep long-term memory in a network, the current state (at time step t) would still be able to effectively influence the future state (at time step T) after a large time interval. Consequently, the gradient at time step T can be effectively propagated to the time step t. By assuming that the minimum effective gradient is ϵ, a range for the recurrent weight of an IndRNN neuron in order to keep long-term memory can be obtained. Specifically, to keep a memory of $T-t$ time steps, $|u_n| \in [\sqrt[(T-t)]{\frac{\epsilon}{\prod_{k=t}^{T-1}\sigma'_{n,k+1}}}, +\infty)$ according to (4) (ignoring the gradient backpropagated from the objective at time step T). That is, to avoid the gradient vanishing for a neuron, the above constraint should be met. In order to avoid the gradient exploding problem, the range needs to be further constrained to $|u_n| \in [\sqrt[(T-t)]{\frac{\epsilon}{\prod_{k=t}^{T-1}\sigma'_{n,k+1}}}, \sqrt[(T-t)]{\frac{\gamma}{\prod_{k=t}^{T-1}\sigma'_{n,k+1}}}]$ where γ is the largest gradient value without exploding. 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 is either 0 or 1. Considering that the short-term memories can be important for the performance of the network as well, especially for a multiple layers RNN, the constraint to the range of the recurrent weight with relu activation function can be relaxed to $|u_n| \in [0, \sqrt[(T-t)]{\gamma}]$. When the recurrent weight is 0, the neuron only uses the information from the current input without keeping any memory from the past. In this way, different neurons can learn to keep memory of different lengths. Note that the regulation on the recurrent weight u is different from the gradient clipping technique. For the gradient clipping or gradient norm clipping [35], the calculated gradient is 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 neu-



(a)

(b)

Figure 1. Illustration of (a) the basic IndRNN architecture and (b) the residual IndRNN architecture.

ron may not be accurate. On the contrary, the regulation proposed here essentially maintains the gradient in an appropriate range without affecting the gradient backprogated through this neuron.

4. Multiple-layer IndRNN

As mentioned above, neurons in the same IndRNN layer are independent of each other, and cross channel information over time is explored through multiple layers of IndRNNs. To illustrate this, we compare a two-layer IndRNN with a traditional 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.

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

$$\mathbf{h}_{s,t} = \mathbf{W}_s \mathbf{h}_{f,t} \qquad (6)$$

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

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

Thus

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

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

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

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 where the recurrent weight of the second layer is zero and the input weight of the second layer is invertible.

It is known that a non-diagonalizable matrix can be made diagonalizable with a perturbation matrix composed of arbitrarily small entries, which is shown in the supplementary file. A stable RNN network needs to be robust to small perturbations (in order to deal with precision errors for example). It is mathematically 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 traditional RNN with a nonlinear activation function.

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 $M \times N + N \times N$, while the number of parameters using one-layer IndRNN is $M \times N + N$. For a two-layer 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.

In all, the cross-channel information can be well explored with a multiple-layer IndRNN although IndRNN neurons are independent of each other in each layer.

4.1. Deeper and Longer IndRNN Architectures

In the proposed IndRNN, the processing of the input ($\mathbf{W}\mathbf{x}_t + \mathbf{b}$) is independent at different timesteps and can be implemented in parallel as in [3, 28]. The proposed IndRNN can be extended to a convolutional IndRNN where, instead of processing input of each time step using a fully connected weight ($\mathbf{W}\mathbf{x}_t$), it is processed with convolutional operation ($\mathbf{W} * \mathbf{x}_t$, where $*$ denotes the convolution operator).

The basic IndRNN architecture is shown in Fig. 1(a), where "weight" and "Recurrent+ReLU" denote the processing of input and the recurrent process at each step with relu as the activation function. By stacking this basic architecture, a deep IndRNN network can be constructed. Compared with an LSTM-based architecture using the sigmoid and hyperbolic tangent functions decaying the gradient over layers, a non-saturated activation function such as relu reduces the gradient vanishing problem over layers. 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. 1(a).

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. Also the layers used to process the input can be of the residual structures in the same way as in CNN [12]. With the simple structure of IndRNN, it is very easy to extend it to different networks architectures. For example, in addition to simply stacking IndRNNs or stacking the layers for processing the input, IndRNNs can also be stacked in the form of residual connections. Fig. 1(b) shows an example of a residual IndRNN based on the "pre-activation" type of residual layers in [13]. At each time step, the gradient can be directly propagated to the other layers from the identity mapping. Since IndRNN addresses the gradient exploding and vanishing problems over time, the gradient can be efficiently propagated over different time steps. Therefore, the network can be substantially deeper and longer. The deeper and longer IndRNN network can be trained end-to-end similarly as other networks.

5. Experiments

In this Section, evaluation of the proposed IndRNN on various tasks are presented.

5.1. Adding Problem

The adding problem [14, 1] 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 [17], it is not included in the report. 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.

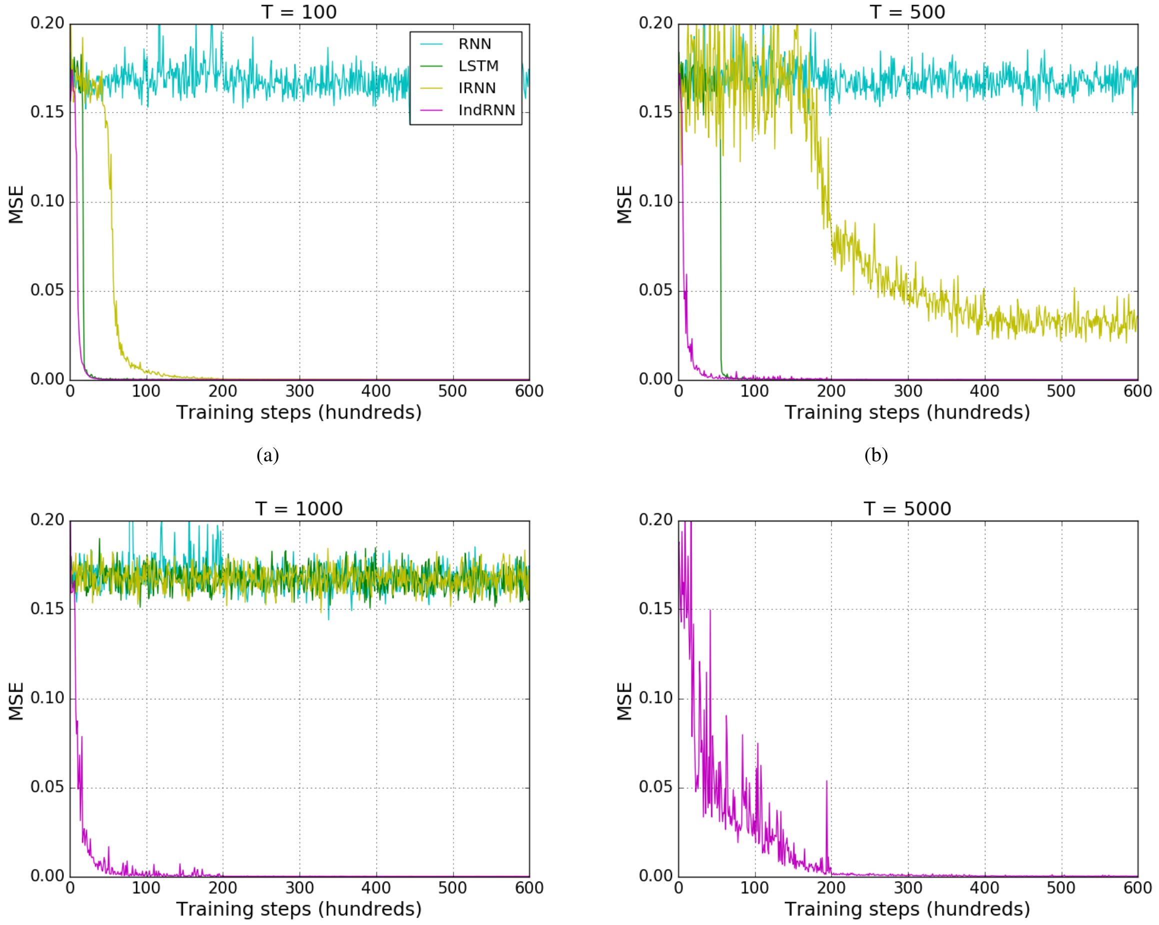
Mean squared error (MSE) was used as the objective function and the Adam optimization method [22] was used for training. The baseline performance (predicting 1 as the output regardless of the input sequence) is mean squared error of 0.167 (the variance of the sum of two independent uniform distributions). 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 do not converge 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 [1] which only used a set of randomly pre-generated data.

The results are shown in Fig. 2(a), 2(b) and 2(c). 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. 2(d). 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.

5.1.1 Analysis of Neurons' Behaviour

In the proposed IndRNN, neurons in each layer are independent of each other which allows analysis of each neuron's behaviour without considering the effect coming from other neurons. Fig. 3(a) and 3(b) show the activation of the neurons in the first and second layers, respectively, for one random input with sequence length 5000. It can be



(a)

(b)

(c)

(d)

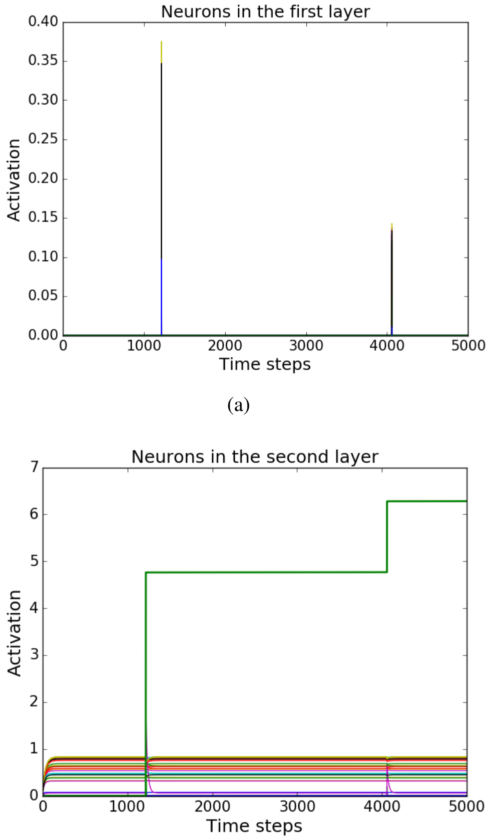
Figure 2. Results of the adding problem for different sequence lengths. The legends for all figures are the same and thus only shown in (a).

seen that neurons in the first layer mainly pick up the information of the numbers to be added, where the strong responses correspond to the locations to be summed indicated by the sequence. It can be regarded as reducing noise, i.e., reducing the effect of other non-useful inputs in the sequence. For the second layer, one neuron aggregates inputs to long-term memory while others generally preserve their own state or process short-term memory which may not be useful in the testing case (since only the hidden state of the last time step is used as output). From this result, we conjecture that only one neuron is needed in the second layer to model the adding problem. Moreover, since neurons in the second layer are independent from each other, one neuron can still work with the others removed (which is not possible for the traditional RNN models).

To verify the above conjecture, an experiment was conducted where the first IndRNN layer is initialized with the trained weights and the second IndRNN layer only consists of one neuron initialized with the weight of the neuron that keeps the long-term memory. Accordingly, the final fully connected layer used for output is a neuron with only one input and one output, i.e., two scalar values including one weight parameter and one bias parameter. Only the final output layer was trained/fine-tuned in this experiment and the result is shown in Fig. 4. It can be seen that with only one IndRNN neuron in the second layer, the model is still able to model the adding problem very well for sequences with length 5000 as expected.

5.2. Sequential MNIST Classification

Sequential MNIST classification is another problem that is widely used to evaluate RNN models. The pixels of MNIST digits [27] are presented sequentially to the networks and classification is performed after reading all pixels. To make the task even harder, the permuted MNIST classification was also used where the pixels are processed with a fixed random permutation. Since an RNN with tanh does not converge to a high accuracy (as reported in the lit-



(a)

(b)

Figure 3. Neurons' behaviour in different layers of the proposed IndRNN for long sequences (5000 time steps) in the adding problem.

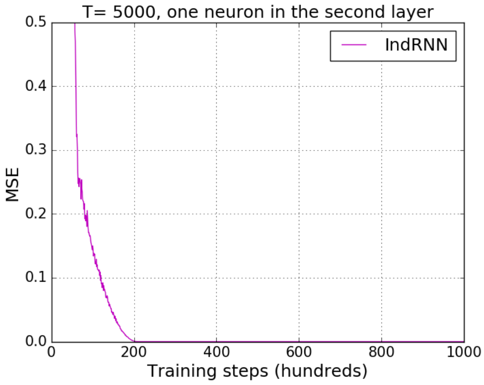


Figure 4. Result of the adding problem with just one neuron in the second layer for sequences of length 5000.

Table 1. Results (in terms of error rate (%)) for the sequential MNIST and permuted MNIST.

	MNIST	pMNIST
IRNN [26]	5.0	18
uRNN [1]	4.9	8.6
RNN-path [33]	3.1	-
LSTM [1]	1.8	12
LSTM+Recurrent dropout [38]	-	7.5
LSTM+Recurrent batchnorm [7]	-	4.6
LSTM+Zoneout [23]	-	6.9
LSTM+Recurrent batchnorm+Zoneout	-	4.1
IndRNN (6 layers)	1.0	4.0

erature [26]), only IndRNN with relu was evaluated. As explained in Section 4.1, IndRNN can be stacked into a deep network. Here we used a six-layer IndRNN, and each layer has 128 neurons. To accelerate the training, batch normalization is inserted after each layer. The Adam optimization was used with the initial learning rate 2×10^{-4} and reduced by a factor of 10 every 600K training steps. The results are shown in Table 1 in comparison with the existing methods. It can be seen that IndRNN achieved better performance than the existing RNN models.

5.3. Language Modeling

In this subsection, we evaluate the performance of the proposed IndRNN on the language modelling task using the character-level Penn Treebank (PTB-c) dataset. The test setting is similar to [7]. A six-layer IndRNN with 2000 hidden neurons is used for the test. To demonstrate that the IndRNN network can be very deep with the residual connections, a 21-layer residual IndRNN as shown in Fig. 1(b) in Subsection 4.1 was adopted. The frame-wise batch normalization [25] is applied, and the batch size is set to 128. Adam was used for training with initial learning rate set to 2×10^{-4} and dropped by a factor of 10 when performance on the validation set was no longer improved. The sequences are non-overlapping and length $T = 50$ was used in training and testing. Dropout [9] with a dropping probability of 0.25 and 0.3 were used for the 6-layer IndRNN and the residual IndRNN. Performance was evaluated using bits per character metric (BPC).

The results are shown in Table 2 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. It can also been seen that with a deeper residual IndRNN, the performance can be further improved.

5.4. Skeleton based Action Recognition

The NTU RGB+D dataset [39] was used for the skeleton based action recognition. This dataset is currently the largest action recognition dataset with skeleton modality. It contains 56880 sequences 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 [39]. In each evaluation protocol, 5% of the

Table 2. Results of PTB-c for our proposed IndRNN model in comparison with results reported in the literature,in terms of BPC.

	Test
RNN-tanh [24]	1.55
RNN-relu [33]	1.55
RNN-TRec [24]	1.48
HF-MRNN [32]	1.42
RNN-path [33]	1.47
LSTM [23]	1.36
LSTM+Recurrent dropout [38]	1.32
LSTM+Recurrent batchnorm [7]	1.32
HyperLSTM + LN [11]	1.25
Hierarchical Multiscale LSTM + LN [6]	1.24
LSTM+Zoneout [23]	1.27
IndRNN (6 layers)	1.26
IndRNN (21 layers)	1.21

training data was used for evaluation as suggested in [39] and 20 frames were sampled from each instance as one input in the same way as in [30]. 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. A four-layer IndRNN and a six-layer IndRNN with 512 hidden neurons were both tested. Batch size was 128 and the Adam optimization was used with the initial learning rate 2×10^{-4} and decayed by 10 once the evaluation accuracy does not increase. Dropout [9] was applied after each IndRNN layer with a dropping probability of 0.25 and 0.1 for CS and CV settings, respectively.

The final result is shown in Table 3 including comparisons with the existing methods. It can be seen that the proposed IndRNN greatly improves the performance over other RNN or LSTM models on the same task. For CS, RNN and LSTM of 2 layers can only achieve accuracies of 56.29% and 60.09% while a 4-layer IndRNN achieved 78.58%. For CV, RNN and LSTM of 2 layers only achieved accuracies of 64.09% and 67.29% while 4-layer IndRNN achieved 83.75%. As demonstrated in [30, 39], the performance of LSTM cannot be further improved by simply increasing the number of parameters or increasing the number of layers. However, 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. This performance is better than the state-of-the-art methods including those with attention models [40, 2] and other techniques [45, 30].

Table 3. Results of all skeleton based methods on NTU RGB+D dataset.

Method	CS	CV
Deep learning on Lie Group [15]	61.37%	66.95%
JTM+CNN [42]	73.40%	75.20%
Res-TCN [21]	74.30%	83.10%
SkeletonNet(CNN) [19]	75.94%	81.16%
JDM+CNN [29]	76.20%	82.30%
Clips+CNN+MTLN [20]	79.57%	84.83%
Enhanced Visualization+CNN [31]	80.03%	87.21%
1 Layer RNN [39]	56.02%	60.24%
2 Layer RNN [39]	56.29%	64.09%
1 Layer LSTM [39]	59.14%	66.81%
2 Layer LSTM [39]	60.09%	67.29%
1 Layer PLSTM [39]	62.05%	69.40%
2 Layer PLSTM [39]	62.93%	70.27%
JL_d+RNN [45]	70.26%	82.39%
STA-LSTM [40]	73.40%	81.20%
ST-LSTM + Trust Gate [30]	69.20%	77.70%
Pose conditioned STA-LSTM[2]	77.10%	84.50%
IndRNN (4 layers)	78.58%	83.75%
IndRNN (6 layers)	81.80%	87.97%

6. Conclusion

In this paper, we presented an independently recurrent neural network (IndRNN), where neurons in one layer are independent of each other. The gradient backpropagation through time process for the IndRNN has been explained and a regulation technique has been developed to effectively address the gradient vanishing and exploding problems. Compared with the existing RNN models including LSTM and GRU, IndRNN can process much longer sequences. The basic IndRNN can be stacked to construct a deep network especially combined with residual connections over layers, and the deep network can be trained robustly. In addition, independence among neurons in each layer allows better interpretation of the neurons. Experiments on multiple fundamental tasks have verified the advantages of the proposed IndRNN over existing RNN models.

References

[1] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015. 2, 5, 7

[2] F. Baradel, C. Wolf, and J. Mille. Pose-conditioned spatio-temporal attention for human action recognition. *arXiv preprint arXiv:1703.10106*, 2017. 8

[3] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016. 2, 4

[4] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki. Scene labeling with lstm recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3547–3555, 2015. 1

[5] 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. *arXiv preprint arXiv:1406.1078*, 2014. 1, 2

[6] J. Chung, S. Ahn, and Y. Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016. 8

[7] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016. 7, 8

[8] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015. 1

[9] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016. 7, 8

[10] 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. 1

[11] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 8

[12] 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*, pages 770–778, 2016. 1, 2, 5

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 5

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2, 5

[15] Z. Huang, C. Wan, T. Probst, and L. Van Gool. Deep learning on lie groups for skeleton-based action recognition. *arXiv preprint arXiv:1612.05877*, 2016. 8

[16] M. I. Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997. 1

[17] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015. 1, 5

[18] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015. 1

[19] 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*, 24(6):731–735, 2017. 8

[20] Q. Ke, M. Bennamoun, S. An, F. Sohel, and F. Boussaid. A new representation of skeleton sequences for 3d action recognition. *arXiv preprint arXiv:1703.03492*, 2017. 8

[21] T. S. Kim and A. Reiter. Interpretable 3d human action analysis with temporal convolutional networks. *arXiv preprint arXiv:1704.04516*, 2017. 8

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[23] 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. *arXiv preprint arXiv:1606.01305*, 2016. 2, 7, 8

[24] D. Krueger and R. Memisevic. Regularizing rnns by stabilizing activations. In *Proceeding of the International Conference on Learning Representations*, 2016. 2, 8

[25] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 2657–2661. IEEE, 2016. 7

[26] 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. 2, 7

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[28] T. Lei and Y. Zhang. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755*, 2017. 2, 4

[29] C. Li, Y. Hou, P. Wang, and W. Li. Joint distance maps based action recognition with convolutional neural networks. *IEEE Signal Processing Letters*, 24(5):624–628, 2017. 8

[30] 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*, pages 816–833. Springer, 2016. 8

[31] M. Liu, H. Liu, and C. Chen. Enhanced skeleton visualization for view invariant human action recognition. *Pattern Recognition*, 68:346–362, 2017. 8

[32] 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. 8

[33] 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*, pages 3477–3485, 2016. 2, 7, 8

[34] B. Parlett. Laguerre's method applied to the matrix eigenvalue problem. *Mathematics of Computation*, 18(87):464–485, 1964. 3

[35] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013. 3

[36] S. Pradhan and S. Longpre. Exploring the depths of recurrent neural networks with stochastic residual learning. *Report*. 1, 2

[37] Z. Qiu, T. Yao, and T. Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5533–5541, 2017. 2

[38] S. Semeniuta, A. Severyn, and E. Barth. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016. 7, 8

[39] 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*, pages 1010–1019, 2016. 2, 7, 8

[40] 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 *AAAI*, pages 4263–4270, 2017. 8

[41] S. S. Talathi and A. Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*, 2015. 2

[42] P. Wang, Z. Li, Y. Hou, and W. Li. Action recognition based on joint trajectory maps using convolutional neural networks. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 102–106. ACM, 2016. 8

[43] 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*, pages 4880–4888, 2016. 2

[44] 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. 1, 2

[45] 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*, pages 148–157. IEEE, 2017. 8

Appendix A

Proposition 1. A non-diagonalizable square matrix can be made diagonalizable by adding a perturbation matrix with arbitrarily small entries.

Proof: For an square matrix **U**, its Jordan canonical form is a block diagonal matrix $J = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_p \end{bmatrix}$ and for some S, $U = SJS^{-1}$. Each Jordan block is a square matrix of the form $J_i = \begin{bmatrix} \lambda_i & 1 & & & & \\ & \lambda_i & 1 & & & \\ & & & \ddots & & \\ & & & \ddots & & \\ & & & & & 1 \\ & & & & & \lambda_i \end{bmatrix}$ and λ_i represents the eigenvalue of the matrix. For non-diagonalizable matrices, the dimension of a Jordan block (denoted by m) is larger than 1 ($m > 1$), which corresponds to repeated eigenvalues. By adding a perturbation matrix $S\Delta S^{-1}$ where $\Delta = \begin{bmatrix} \Delta_1 & & \\ & \ddots & \\ & & \Delta_p \end{bmatrix}$ and $\Delta_i = \begin{bmatrix} 0 & & & \\ & \epsilon_1 & & \\ & & \ddots & \\ & & & \epsilon_{m-1} \end{bmatrix}$ (ϵ is arbitrary and different from each other), the eigenvalues of **U** can be made different and thus the matrix becomes diagonalizable. Since ϵ can be arbitrarily small, the entries of the perturbation matrix can be arbitrarily small.