Revisiting the Independence Assumption for Recurrent Neural Networks

Abstract

Commonly used RNN models like LSTMs are known to be parameter-heavy and difficult to train. In this work, we explore efficient RNN designs for learning sequential data. Especially, we observe that the independence assumption, appearing in the independently recurrent neural network (IndRNN), can be a general inductive bias suitable for efficient sequence learning. Based on this, we propose the independent reversible recurrent neural network (IRevRNN), which is a lightweight and effective RNN model with strong representational power. IRevRNN constrains the recurrent dynamics to be independent for each neuron, allowing for fast asynchronous computation in parallel. To model complex sequential dependencies with moderate memory footprint, IRevRNN employs reversible residual blocks, which effectively introduce long-term memory channels. We conduct experiments on a variety of sequence learning tasks, including pixel-by-pixel image classification, skeleton-based action recognition, and video classification. Experimental results show that IRevRNN consistently outperforms other RNN baselines.

1 Introduction

Recurrent neural networks (RNNs) have been the textbook deep learning approach for learning sequential data [10]. Models such as long short-term memory (LSTM) [17] and gated recurrent unit (GRU) [6] tackle the well-known exploding/vanishing gradient problem of vanilla RNNs [10] and are widely used in practice. However, their gating mechanisms result in parameter-heavy architectures which are hard to train, and they still have trouble handling very long sequences [26]. These issues limit their application to sequence learning problems and prompt research directions which seek alternative solutions.

Most notably, recent studies related to the attention mechanism and the transformer architecture [36, 7, 4], have achieved state-of-the-art results for sequence learning tasks like machine translation and question answering, leading many to believe that RNN-based approaches are simply outdated. However, attention-based architectures typically have large model sizes and require massive data for (pre)training. This can be unrealistic for many learning scenarios, which might have limited training data, restricted computational resources, or demand fast inference. More efficient alternatives are thus needed in these cases.

The goal of this work is to construct a lightweight RNN model that can tackle these resource-limited learning scenarios, with the help of reasonable inductive biases. For computer vision applications, convolutional neural networks (CNNs), with local receptive fields and shared kernel weights, have been a predominant choice for learning high-dimensional image data. Highly efficient CNN variants such as SqueezeNet [19] and MobileNet [18] have also been proposed. For sequence learning, RNNs follow the Markov assumption [10] and have shared parameterization across the sequence dimension. We seek in this work additional guiding principles to construct more efficient RNNs.

To achieve this, we revisit the work of Li et al. [26] which proposes the independently recurrent neural network (IndRNN). In Section 3, we discuss the independence assumption which is made in the IndRNN model and remark that it is not tied with the specific architecture. Instead, it can be a general inductive bias suited for efficient sequence learning. We then propose the independent

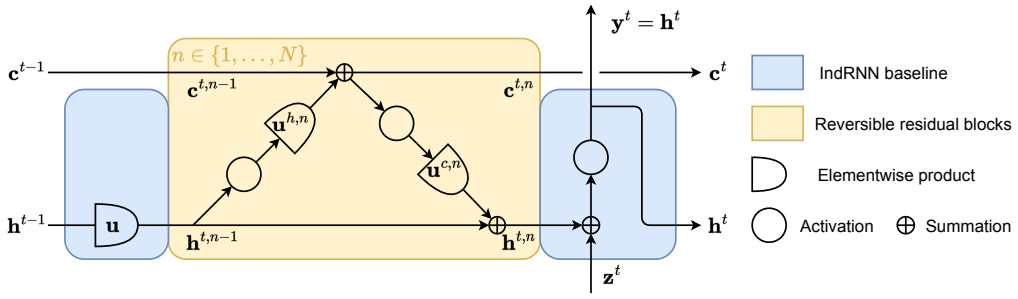

Figure 1: Illustration of the proposed IRevRNN structure. IRevRNN makes the independence assumption: all the computations are elementwise, thus the sequential dependencies are computed independently for each neuron, and can be carried out asynchronously in parallel. The reversible residual blocks can model complex sequential relations, and they effectively construct a long-term memory channel.

reversible recurrent neural network (IRevRNN) model, which also respects the independence assumption, but has stronger representational power enabled by the reversible residual blocks [9] integrated into its design. IRevRNN is parameter-efficient, has fast asynchronous parallel computation, and is endowed with long-term memory channels. Experiments in Section 4 demonstrate that it can effectively address various sequence learning problems and consistently outperform all other RNN baselines.

2 Related work

Vanilla RNNs are known to suffer from the exploding/vanishing gradient problem [10] and cannot handle long sequences. To cope with this issue, there exists two main directions. The more classical one, taken by the commonly used LSTM [17] and GRU [6] models, use the gating mechanism to regulate the recurrent flow and stabilize the training. These methods tend to have complex and parameter-heavy architectures. Moreover, they still struggle with problems involving very long sequential data [26].

Alternatively, several approaches address the gradient explosion/vanishing problem by constraining the recurrent connections: IRNN [22] initializes the recurrent weight matrix to (scaled) identity, which eases the RNN training and enables the use of rectified linear unit (ReLU) activation. On the other hand, unitary evolution RNN (uRNN) [1] builds up unitary hidden connections whose complex eigenvalues are guaranteed to have absolute value one, which effectively avoids the gradient explosion/vanishing problem.

IndRNN [25, 26] follows the second direction and constrains the recurrent weight matrix to always be diagonal, unlike IRNN which only requests diagonalization during initialization. Despite the simple structure, IndRNN is able to outperform other RNN baselines and can build up deep network models to further boost the performance [25].

RNNs are not the only solution for sequence modeling. Several alternatives have also been proposed for learning sequential data. This includes temporal convolutional networks [23, 3], attention-based transformer architectures [36, 7, 4], differentiable memory models [12, 16], . RNNs can also be combined with the attention mechanism. For instance, recurrent independent mechanism [11] groups together nearly independent recurrent components which only communicate via an attention bottleneck.

3 Independent reversible RNN

In this section, we propose the Independent reversible recurrent neural network (IRevRNN). We begin with some important concepts in Section 3.1 and then formulate the IRevRNN model in Sections 3.2 and 3.3. Finally, we discuss its representational power and computational aspects in Sections 3.4 and 3.5.

3.1 Important concepts

Before introducing the IRevRNN model, we discuss two important concepts which it builds on: the independence assumption for RNN and the reversible residual blocks.

Independence assumption for RNN Given an input sequence $(\mathbf{x}^1, \ldots, \mathbf{x}^T)$, a vanilla RNN layer produces an output $\mathbf{y}^t$ for each step t using the following formula [10]:

$$\mathbf{z}^t = \mathbf{W}\mathbf{x}^t + \mathbf{b}, \tag{1}$$

$$\mathbf{y}^t = f(\mathbf{U}\mathbf{y}^{t-1} + \mathbf{z}^t) \tag{2}$$

where f is the activation function, $\mathbf{W}, \mathbf{b}$ are the non-recurrent weights and biases processing the input entries $\mathbf{x}^t$ at each step, $\mathbf{z}^t$ is the current pre-activation, and $\mathbf{U}$ is the recurrent weight matrix. Vanilla RNNs have dense recurrent connections to model general dependencies.

From a probabilistic aspect, vanilla RNNs make the Markov assumption $p(\mathbf{y}^t|\mathbf{x}^{1:t}, \mathbf{y}^{1:t-1}) = p(\mathbf{y}^t|\mathbf{x}^t, \mathbf{y}^{t-1})$ and model the dependency $p(\mathbf{y}^t|\mathbf{x}^t, \mathbf{y}^{t-1})$ with Eq. (2).

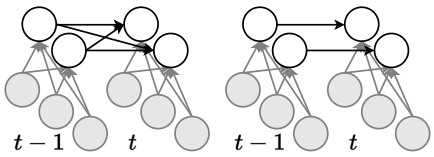

Figure 2: Comparison of vanilla RNN (left) IndRNN (right). Vanilla RNNs have dense recurrent connections while IndRNN has one-on-one recurrent links separately for each neuron.

To avoid exploding/vanishing gradients, IndRNN [26] uses a diagonal recurrent matrix $\mathbf{U} = \text{diag}(\mathbf{u})$, leading to the update

$$\mathbf{y}^t = f(\mathbf{u} \odot \mathbf{y}^{t-1} + \mathbf{z}^t) \tag{3}$$

with $\odot$ denoting elementwise product. IndRNN effectively makes the following independence assumption:

$$\forall t, \forall i, \quad p(y_i^t|\mathbf{x}^t, \mathbf{y}^{t-1}) = p(y_i^t|\mathbf{x}^t, y_i^{t-1}), \tag{4}$$

which approximates the sequential dependencies with independent neuronwise relations for each neuron y_i.

As we will see in the following, the independence assumption depicted by Eq. (4) is actually a general inductive bias suited for efficient sequence modeling, and it is also followed by our IRevRNN model.

Reversible residual blocks The residual block proposed by He [14] is a widely used design pattern for building up deep network architectures and achieving state-of-the-art results for deep learning tasks. To lessen the memory requirement during training, Gomez propose a variant called the reversible residual block [9], where all the intermediate results in the blocks can be reconstructed from the output and thus do not need to be stored for gradient computation during backpropagation.

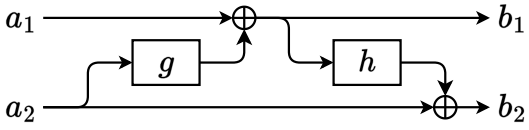

Figure 3: Schema of a reversible residual block with two residual transformations g, h which can contain learnable parameters. This particular structure allows for reconstructing the inputs (a_1, a_2) from the outputs (b_1, b_2).

An example reversible residual block, which includes two residual transformations g and h, is shown in Figure 3. In this example, the outputs (b_1, b_2) can be computed from the inputs (a_1, a_2) following

$$b_1 = a_1 + g(a_2), \tag{5}$$

$$b_2 = a_2 + h(a_1 + g(a_2)), \tag{6}$$

and (a_1, a_2) can be reconstructed from (b_1, b_2) via

$$a_2 = b_2 - h(b_1), \tag{7}$$

$$a_1 = b_1 - g(b_2 - h(b_1)). \tag{8}$$

3.2 Formulation of IRevRNN recurrent update

Using the ideas presented in Section 3.1, we propose the independent reversible recurrent neural network (IRevRNN), whose structure is depicted in Figure 1. Analogous to LSTM [17], IRevRNN also maintains two recurrent states $(\mathbf{h}^t, \mathbf{c}^t)$. To model complex sequential dependencies, IRevRNN makes use of N reversible residual blocks, where the n-th block is parameterized by weights $(\mathbf{u}^{h,n}, \mathbf{u}^{c,n})$. At each time step, IRevRNN computes the layer output $\mathbf{y}^t$ and the new recurrent states $(\mathbf{h}^t, \mathbf{c}^t)$ with the following steps:

- Given the old recurrent states $(\mathbf{h}^{t-1}, \mathbf{c}^{t-1})$, the inputs to the reversible blocks $(\mathbf{h}^{t,0}, \mathbf{c}^{t,0})$ are computed as

$$\mathbf{h}^{t,0} = \mathbf{u} \odot \mathbf{h}^{t-1}, \tag{9}$$
$$\mathbf{c}^{t,0} = \mathbf{c}^{t-1}; \tag{10}$$

- Iterating over the N reversible blocks, the outputs $(\mathbf{h}^{t,n}, \mathbf{c}^{t,n})$ of each block n are obtained via

$$\mathbf{c}^{t,n} = \mathbf{c}^{t,n-1} + \mathbf{u}^{h,n} \odot \tanh(\mathbf{h}^{t,n-1}), \tag{11}$$
$$\mathbf{h}^{t,n} = \mathbf{h}^{t,n-1} + \mathbf{u}^{c,n} \odot \tanh(\mathbf{c}^{t,n}); \tag{12}$$

- Finally, given the current pre-activation $\mathbf{z}^t$ and any chosen activation function f (ReLU), the layer output $\mathbf{y}^t$ and the new recurrent states $(\mathbf{h}^t, \mathbf{c}^t)$ are updated as follows:

$$\mathbf{y}^t = f(\mathbf{h}^{t,N} + \mathbf{z}^t), \tag{13}$$
$$\mathbf{h}^t = \mathbf{y}^t, \tag{14}$$
$$\mathbf{c}^t = \mathbf{c}^{t,N}. \tag{15}$$

Discussion Several remarks can be made regarding the IRevRNN formulation:

- IRevRNN respects the independence assumption since all the operations (sum, product, activation functions) are elementwise and the update characterizes the dependency $p(h_i^t, c_i^t | \mathbf{x}^t, h_i^{t-1}, c_i^{t-1})$;
- The residual transformations in reversible blocks admit a pre-activation form, following the observation from He that it can improve the performance of residual networks [15];
- The non-linear activation functions in the residual transformations are fixed to $\tanh$ since in practice this can stabilize the recurrent flow. On the other hand, the final activation function f can be chosen freely, and we use ReLU in practice which produces good results. (Section 4.5 for detailed quantitative analysis.)

Parameter initialization IRevRNN defines the set of parameters $(\mathbf{u}, \mathbf{u}^{h,1}, \mathbf{u}^{c,1}, \ldots, \mathbf{u}^{h,N}, \mathbf{u}^{c,N})$ which are initialized as follows:

- Since $\mathbf{u}$ has a similar role the recurrent weight in IndRNN, we initialize it following the same setting as discussed in Li [26];
- For the weights $(\mathbf{u}^{h,1}, \mathbf{u}^{c,1}, \ldots, \mathbf{u}^{h,N}, \mathbf{u}^{c,N})$ on the residual branches, we follow existing work [2] and initialize them with random close-to-zero Gaussian noise.

Choosing the number of blocks N The number of reversible blocks N is a hyperparameter of IRevRNN where the optimal choice differs depending on the network architecture and experimental setting. As shown in Section 4.5, using a single reversible block can already significantly improve the result, and typically a small value of N would already yield optimal performance.

3.3 Recurrent networks with IRevRNN layers

The recurrent update of IRevRNN involves the pre-activations $\mathbf{z}^t$, which are typically the output of a non-recurrent weight layer, , a fully connected or a convolutional layer. Since the recurrent update in Section 3.2 makes no assumption on how $\mathbf{z}^t$ should be computed, the non-recurrent part can be chosen freely to match the task at hand. The recurrent structure of IRevRNN, together with a chosen non-recurrent part, forms a complete RNN layer that processes sequential data. Figure 4 shows how sequential data is processed by an IRevRNN layer.

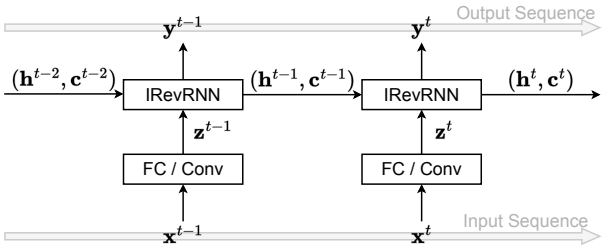

Figure 4: Illustration of how a complete IRevRNN layer, which consists of a non-recurrent weight layer ("FC / Conv") and a recurrent structure ("IRevRNN") depicted in Figure 1, processes sequential input and produces the corresponding output.

Although in principle an IRevRNN layer has a similar role to an LSTM layer, practical recurrent networks using IRevRNN layers tend to have different structures compared to the LSTM-based counterparts. LSTM-based models typically only have a few recurrent layers and tend to rely on prepending a deep non-recurrent part to extract good features from complex individual sequence entries like image frames (, [29]). On the other hand, the lightweight nature of IRevRNN encourages its usage in all layers of the network, and much deeper IRevRNN-based recurrent architectures can be constructed.

Since the IRevRNN can be added to all layers, this also provides a systematic approach to create new IRevRNN-based recurrent networks by converting non-recurrent networks designed to model individual sequence entries: each activation layer in the original network can be replaced with an IRevRNN with the same activation function. In Section 4.4, to solve the task of video classification, we construct an IRevRNN model by converting a PreResNet18 network originally designed for image classification, and we are able to achieve state-of-the-art performance.

Similar to the case of feed-forward neural networks, in practice, we also find it beneficial to include regularization in the IRevRNN-based RNN architectures. To do so, we typically add a batch normalization [20], and optionally also a dropout [35, 8] layer after each IRevRNN layer.

3.4 Representational power of IRevRNN

While following the independence assumption and having a moderate size, IRevRNN has the flexibility to represent complex sequential relations, thanks to its versatile reversible residual blocks. We specifically adopt the reversible blocks, since they yield comparable performance the standard residual blocks [9] while enabling the reconstruction of intermediate results, which reduces the required storage for backpropagation.

Interestingly, we also observe a synergy between the reversible block and the recurrent structure. As illustrated in Figure 1, a long-term memory channel, which tracks the recurrent state $\mathbf{c}^t$, emerges naturally from the reversible residual structure. This is similar to the long-term memory of LSTM [17] and can potentially propagate long-term influences from distant prior steps.

IndRNN, which has obtained surprisingly good results for sequence learning [26, 25] despite the simplistic structure, is in fact a particular case of IRevRNN without any reversible block. The reversible blocks enable the representation of non-linear sequential relations, and experiments in Section 4 show that IRevRNN models consistently improve upon IndRNN baselines.

3.5 Efficiency and computational aspects

From the practical perspective, IRevRNN is highly computationally and memory efficient:

- The independence assumption makes the recurrent computation scale linearly the layer width, instead of the quadratic scaling in the case of vanilla RNN or LSTM/GRU. Additionally, the recurrent computations on each neuron can be carried out concurrently and asynchronously. This allows for massively parallel computation and avoids the slow synchronization at each step. Since asynchronous computation is hard to implement efficiently in common tensor-based deep learning frameworks, we have written a custom CUDA extension for IRevRNN to fully benefit from the speedup of its asynchronous, massively parallel operations on GPUs.
- IRevRNN has a small parameter size which scales linearly the layer width: an IRevRNN layer with N reversible blocks and M neurons has a total of $(2N + 1) \cdot M$ parameters, resulting in a reduced memory footprint. The reversible residual blocks further help to economize the memory usage during training, which is independent of the block count N, since intermediate results are reconstructed and no extra storage is needed for backpropagation.

Thanks to its lightweight and efficient design, deep IRevRNN models can be built and effectively trained.

4 Experiments

To empirically evaluate the proposed IRevRNN model, we conduct a series of experiments in this section. Sections 4.2, 4.3, and 4.4 compare IRevRNN against other RNN baselines on various sequence learning tasks including pixel-by-pixel image classification, skeleton-based action recognition, and video classification. And Section 4.5

conducts ablation studies on IRevRNN to gain more insights about its model design.

4.1 General settings

For all our experiments, we use PyTorch [30] to handle the network definition and training process. As discussed in Section 3.5, to achieve the fast asynchronous computation following the independence assumption, we created a custom CUDA implementation for IRevRNN (including IndRNN as a special case) and wrapped it as a custom PyTorch extension. To put emphasis on the constrained learning scenarios, we focus on experimental setups where trainings can be carried out on a single GPU in reasonable time.

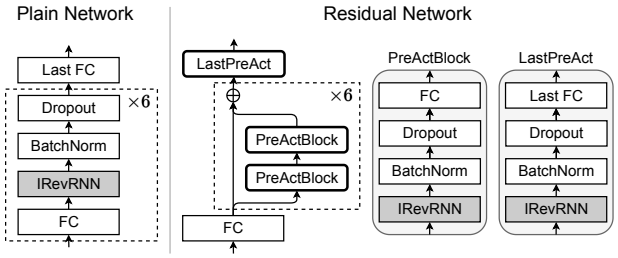

Figure 5: Schema of the plain network and the residual network used in the experiments in Sections 4.2 and 4.3. "FC" denotes a non-recurrent fully connected layer and "Last FC" denotes the final classifier head which processes the last entry of its sequential input. The IRevRNN recurrent structures are marked in gray.

For experiments in Sections 4.2 and 4.3, we employ two common designs for our RNN models:

Plain network As illustrated in Figure 5 left, this is a simple sequential architecture with 6 IRevRNN layers. The non-recurrent components are fully connected layers. And a batch normalization layer, as well as an RNN-adapted dropout layer proposed by Gal and Ghahramani [8], are added after each IRevRNN layer for regularization. To produce the final class prediction, a fully connected classifier head is added which acts on the last step of the recurrent output;

Residual network Depicted in Figure 5 right, this is a more complicated network with 6 residual blocks. We use the same types of layers as in the plain network except for the pre-activation reordering [15]. This amounts to a total of 13 RNN layers and can better model complex sequential dependencies.

To provide a broader scope of references, we also include some additional results reported by other papers on the same tasks. These results are marked with * in Tables 1, 3, 4, and 5, and readers should be aware that they might be obtained with different experimental setups and in some occasions might not be directly comparable.

4.2 Pixel-by-pixel image classification

To begin with, we consider the pixel-by-pixel MNIST benchmark [22] which is commonly used for comparing RNN models. The input consists of black-and-white 28×28 MNIST [24] images as flattened one-dimensional pixel sequences, and the goal is to classify the handwritten digits in this pixel sequence form and predict the digit label. To increase the difficulty and focus on modeling long-term dependencies, the pixel-by-pixel MNIST classification task is also conducted in permuted form, where a predefined permutation is applied to shuffle all input pixel sequences.

To run the experiments, we split the training data into 57000 images for training and 3000 for validation. We construct a plain and a residual version for both the IndRNN and the IRevRNN models, resulting in a total of four RNNs for comparison. The model designs are explained in Section 4.1. We fix the size of all hidden layers to 128 and use a dropout rate of 0.1. For Plain-IRevRNN and Res-IRevRNN models we use 3 reversible blocks. The models are trained with batch size 32 and Adam optimizer [21]. We set the initial learning rate to 0.0002 and weight decay to 0.0001. We schedule all trainings with early-stopping based on the validation loss and reduce the learning rate by a factor of 0.1 whenever there is no improvement for 100 epochs. We stop the training when the model has converged with a learning rate of 0.000002 and can no longer improve.

The results are collected in Table 1. We observe that IRevRNN models consistently outperform the IndRNN counterparts. Also, IRevRNN can be used to build up

Models	Sequential	Permuted
LSTM [1]	98.2%*	88%*
IRNN [26]	95%*	82%*
uRNN [1]	95.1%*	91.4%*
Plain-IndRNN	99.20%	95.76%
Res-IndRNN	99.41%	96.45%
Plain-IRevRNN	99.26%	96.83%
Res-IRevRNN	99.52%	97.48%

Table 1: Test accuracy on unpermuted (sequential) and permuted pixel-by-pixel MNIST. Results marked with * are taken from the respective cited papers. We see that the deep IRevRNN model (Res-IRevRNN) can be effectively trained and outperforms all other RNN baselines.

deep RNN models (Res-IRevRNN) which achieve the best performance among all tested RNN baselines.

Additionally, we also examine results on the pixel-by-pixel FashionMNIST classification task, which is based on the FashionMNIST [37] dataset. FashionMNIST consists of gray-scale 28×28 images depicting fashion clothing items and is a harder task compared to MNIST classification. Since it has the same size and format as the MNIST dataset, we reuse the previously discussed experimental setting from the pixel-by-pixel MNIST task.

Models	Sequential	Permuted
Plain-IndRNN	92.68%	87.96%
Res-IndRNN	93.29%	88.29%
Plain-IRevRNN	93.48%	88.45%
Res-IRevRNN	93.51%	88.65%

Table 2: Test accuracy on unpermuted (sequential) and permuted pixel-by-pixel FashionMNIST. Again, IRevRNNs, especially the deeper Res-IrevRNN model, yield the best results.

Table 2 summarizes the results. Here the conclusions are consistent the MNIST case, where IRevRNN models outperform IndRNN baselines and Res-IRevRNN has the best performance overall.

4.3 Skeleton-based action recognition

Next, we consider the task of skeleton-based action recognition, where human actions are tracked with key points which mark the body joints, and the goal is to recognize the action performed by the human subjects based on the key point trajectories.

We start with the NTU RGB+D dataset [32], which contains a total of 56880 recorded sequences belonging to 60 action classes. It is commonly evaluated with a cross-subject (40320 training sequences, 16560 test sequences) and a cross-view (37920 training sequences, 18960 test sequences) split. We normalize the data by centering the positions around the key point marking the middle of the spline. Since some action classes are mutual actions involving two human subjects, we set the input entries to always be a concatenation of two skeleton data. For action classes where only a single subject is involved, we simply double the skeleton features.

For the training process, we reuse the setting from the official IndRNN [26, 25] implementation[1], and train on mini-batches of 128 skeleton sequences with length 20. The trained network architectures are the ones introduced in Section 4.1 with hidden size set to 512, and Adam optimizer [21] is used with an initial learning rate of 0.0002 and weight decay 0.0001. Validation loss is used for early stopping and scheduling the learning rate. The results are collected in Table 3.

The above results suggest that IRevRNN models can outperform other RNN baselines, and the deeper Res-IRevRNN architecture achieves the best performance in both cross-subject and cross-view settings.

Furthermore, we also run the experiments on the NTU RGB+D 120 dataset [27]. This is a newer extension that has more sequences (114480) and action classes (120), but the data format remains the same. We use the same experimental setup as for the NTU RGB+D dataset, and summarize the final results in Table 4.

Similar to the NTU RGB+D results, we see that the Res-IRevRNN model can consistently outperform other RNN baselines for the task of skeleton-based action recognition.

[1] https://github.com/Sunnydreamrain/IndRNN_pytorch

Models	Cross-Subject	Cross-View
RNN [32]	56.29%*	64.09%*
LSTM [32]	60.09%*	67.29%*
P-LSTM [32]	62.93%*	70.27%*
STA-LSTM [33]	73.40%*	81.20%*
JL_d LSTM [39]	70.26%*	82.39%*
VA-LSTM [38]	79.4%*	87.6%*
Plain-IndRNN	77.81%	81.79%
Res-IndRNN	80.92%	85.62%
Plain-IRevRNN	82.04%	86.39%
Res-IRevRNN	83.31%	88.02%

Table 3: Cross-subject and cross-view test accuracies on the NTU RGB+D dataset. Results marked with * are reported by the respective cited papers. We see that IRevRNN models, especially Res-IRevRNN, have the best results for both the cross-subject and the cross-view settings.

Models	Cross-Subject	Cross-Setup
ST-LSTM [27]	55.7%*	57.9%*
GCA-LSTM [27]	58.3%*	59.2%*
Two-Stream Attn. [27]	61.2%*	63.3%*
Plain-IndRNN	68.15%	69.69%
Res-IndRNN	70.59%	74.37%
Plain-IRevRNN	72.42%	74.71%
Res-IRevRNN	73.98%	77.75%

Table 4: Cross-subject and cross-setup test accuracies on the NTU RGB+D 120 dataset. Results marked with * are reported by the respective cited papers. Again, Res-IRevRNN achieves the state-of-the-art among all tested RNN baselines.

4.4 Video classification

Also, we look at the video classification task, which presents some new challenges compared to the previous problems. Especially, at each step, we need to process a video frame, which is a high-dimensional RGB image that cannot be handled by simple fully connected layers in reasonable time. Thus more elaborate network design is needed to handle this problem.

In this section, we consider the UCF101 dataset [34] for video classification. UCF101 consists of 13320 video clips categorized into 101 action classes. For data preprocessing, we follow an existing code base[2], which splits the dataset into 9537 videos for training, 756 videos for validation, and 3783 videos for testing. We rescale all video frames to have 128 pixels in height, and random crops yielding 32 consecutive frames of size 112×112 are used during training. At test time, spatial-temporal center crops are used instead to obtain deterministic predictions for each video. Besides random cropping, a random clip-wise horizontal flip is also added during training for data augmentation.

In terms of network architectures, we construct three RNNs for comparison. They are all based on a common PreResNet18 [15] backbone, but use LSTM, IndRNN, and IRevRNN respectively. As discussed in Section 3.3, there exists two different designs for RNNs:

- For the LSTM model, we start with a CNN backbone to extract high-level features individually from each video frame, followed by several LSTM layers to learn the sequential dependencies. We create the CNN backbone by removing the final classifier head of the PreResNet18 model, then append 2 LSTM layers for recurrent processing, followed by a global average pooling and a final linear classifier head. We refer to this model as "ResNet-LSTM";
- For IRevRNN and IndRNN models, we instead add the recurrent structure for all layers, at the positions where activation layers are originally situated. This results in RNN architectures with recursive pre-activation residual blocks as illustrated in Figure 6. We denote these models "Res-IndRNN" and "Res-IRevRNN".

The above RNN models are trained with the AdamW optimizer [28] for 100 epochs from scratch. The initial learning rate is set to 0.0001 and is divided by 10 whenever the validation loss has not decreased for 10 epochs. We set weight decay to 1.0 and other optimizer options to the default setting. We use batch size 8 during training and for Res-IRevRNN we fix the number of reversible blocks

[2] https://github.com/leftthomas/R2Plus1D-C3D

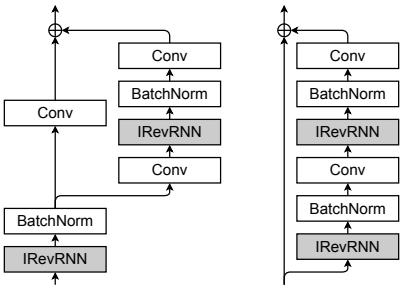

Figure 6: Recursive pre-activation residual blocks using IRevRNN structures (gray), with (left) or without (right) shortcut projection.

to 1. The results are collected in Table 5 along with some additional references reported by other papers[3].

Models	Accuracy
SVM [34]	43.9%*
ResNet18 [13]	42.4%*
C3D [5]	51.6%*
Conv3D-NAS [31]	58.6%*
ResNet-LSTM	53.45%
Res-IndRNN	64.95%
Res-IRevRNN	65.19%

Table 5: Test accuracies on the UCF101 dataset. Results marked with * are taken from the respective cited papers. All the results are obtained with trainings from scratch. We observe that IRevRNN achieves the best accuracy compared to the baselines.

We observe that with a standard network backbone and data augmentation pipeline, the IRevRNN model can achieve good results and outperform other baselines. It also shows that making new IRevRNN models from non-recurrent backbones is a viable approach.

[3]We are unable to find any result on UCF101 for RNN models that are trained from scratch. Non-RNN results are included as alternatives.

4.5 Ablation studies

Finally, we conduct ablation studies in this section to analyze the effect of hyperparameters in IRevRNN and obtain insight into its model design. In particular, we investigate the choice of reversible block count N, as well as the types of activation functions used in IRevRNN.

Number of reversible blocks N Since the number of neurons M in IRevRNN is determined by the size of $\mathbf{z}^t$, only the number of reversible blocks N can be chosen freely for the IRevRNN structure. In general, increasing N results in a more complicated recurrent model which has stronger representational power for sequential dependencies. However, it would also increase the computational cost, since the number of model parameters ($(2N+1)\cdot M$) and the computational complexity scale linearly the reversible block count. It is thus helpful to investigate how the number of reversible blocks N affects the model performance in practice to have better insight into its optimal choice.

We reuse the experimental setting for the unpermuted pixel-by-pixel FashionMNIST task and compare the test accuracies of Plain-IRevRNN with varying reversible blocks. The results are visualized in Figure 7 as a line plot.

There are two main observations from Figure 7:

- The reversible residual block structure is clearly beneficial for IRevRNN. We see a visible performance boost even in the case when $N = 1$. The long-term memory introduced by the reversible block might be a possible reason for this improvement;

- A few reversible blocks can already yield optimal results. This means that we can expect improved performance with only minor computational overhead.

Choice of activation functions IRevRNN contains activation functions and their proper choice also necessitates an in-depth investigation. There are two types of activation functions: the intermediate ones on the residual branches, and the final output activation. Again, we reuse the unpermuted FashionMNIST setup and train Plain-IRevRNNs while varying the type of either the residual

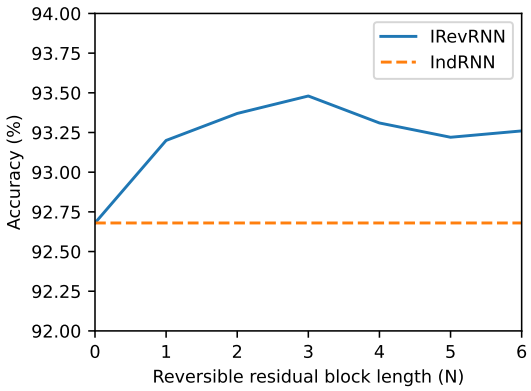

Figure 7: Test accuracies of Plain-IRevRNN models with varying choices of reversible block count (N) on unpermuted pixel-by-pixel FashionMNIST dataset. As a reference, the result of IndRNN is indicated with the orange dashed line. We see that a small number of blocks can already significantly boost the result.

or the output activation function. In each scenario, we experiment with ReLU, Sigmoid, and Tanh activations. The results are shown in Tables 6 and 7.

Activation	Accuracy
ReLU	Failed
Sigmoid	92.41%
Tanh	93.48%

Table 6: Results with varying choice of activation function on the residual branches. We see that Tanh is the optimal choice and ReLU leads to instability.

For the choice of residual activation function, we see from Table 6 that Tanh is the optimal choice. In particular, choosing the ReLU activation leads to unstable training. Sigmoid, while having a similar shape as Tanh, is also a suboptimal choice, since it yields a significant performance deterioration in practice.

In terms of the final output activation, Table 7 shows that it can be freely chosen without breaking the training. However, the final result will depend on this choice. Here, similar to the case of feed-forward networks, we see

Activation	Accuracy
Sigmoid	90.43%
Tanh	93.12%
ReLU	93.48%

Table 7: Results with different choices of output activation functions. Here the ReLU activation achieves the best performance.

that the ReLU activation can outperform the more "old school" Sigmoid and Tanh activations. Furthermore, we notice that Sigmoid and Tanh based model variants are also slower to converge and take longer to train.

5 Conclusion

In this work, we propose the independent reversible recurrent neural network (IRevRNN), which is an efficient RNN model for sequence learning. Based on the independence assumption of sequential dependencies, as well as the reversible residual structure, IRevRNN can adequately represent complex sequential relations while having a small model size and moderate computational requirements. IRevRNN admits fast asynchronous computation in parallel, and our experiments show that it consistently outperforms other RNN baselines on a variety of sequence learning problems.

While the community currently has a much stronger focus on research related to the attention mechanism and transformer-based architectures, we believe that recurrent neural networks still represent an important approach for tackling sequence learning. Especially, lightweight and efficient RNNs, such as the IRevRNN model proposed in this work, could potentially be a promising approach for online learning scenarios. Using efficient RNN models to learn from data streams, especially in a real-time setting, could be an interesting direction for future research.

References

[1] Martín Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *ICML*, 2016. 2, 7

[2] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Gary Cottrell, and Julian J. McAuley. Rezero is all you need: fast convergence at large depth. In *UAI*, 2021. 4

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. 2

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020. 1, 2

[5] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *CVPR*, 2017. 9

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 1, 2

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019. 1, 2

[8] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, 2016. 5, 6

[9] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. In *NIPS*, 2017. 2, 3, 5

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 1, 2, 3

[11] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *ICLR*, 2021. 2

[12] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gomez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John P. Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. 2

[13] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *CVPR*, 2018. 9

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 4, 6, 8

[16] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. In *ICLR*, 2017. 2

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 1, 2, 3, 5

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017. 1

[19] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016. 1

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6, 7

[22] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015. 2, 6

[23] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *CVPR*, 2017. 2

[24] 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

[25] Shuai Li, Wanqing Li, Chris Cook, Yanbo Gao, and Ce Zhu. Deep independently recurrent neural network (indrnn). *arXiv preprint arXiv:1910.06251*, 2019. 2, 5, 7

[26] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper RNN. In *CVPR*, 2018. 1, 2, 3, 4, 5, 7

[27] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C Kot. Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2684–2701, 2019. 7, 8

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 8

[29] Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015. 5

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 6

[31] Wei Peng, Xiaopeng Hong, and Guoying Zhao. Video action recognition via neural architecture searching. In *ICIP*, 2019. 9

[32] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3d human activity analysis. In *CVPR*, 2016. 7, 8

[33] Sijie Song, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *AAAI*, 2017. 8

[34] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 8, 9

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 5

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 1, 2

[37] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 7

[38] Pengfei Zhang, Cuiling Lan, Junliang Xing, Wenjun Zeng, Jianru Xue, and Nanning Zheng. View adaptive recurrent neural networks for high performance human action recognition from skeleton data. In *ICCV*, 2017. 8

[39] Songyang Zhang, Xiaoming Liu, and Jun Xiao. On geometric features for skeleton-based action recognition using multilayer lstm networks. In *WACV*, 2017. 8