Recurrent Convolutions: A Model Compression Point of View

Zhendong Zhang
Media Lab, Xidian University
Xi'an, 710071, China
zhd.zhang.ai@gmail.com

Cheolkon Jung
Media Lab, Xidian University
Xi'an, 710071, China
zhengzk@xidian.edu.cn

Abstract

Recurrent convolution (RC) shares the same convolutional kernels and unrolls them multiple times, which is originally proposed to model time-space signals. We suggest that RC can be viewed as a model compression strategy for deep convolutional neural networks. RC reduces the redundancy across layers and is complementary to most existing model compression approaches. However, the performance of an RC network can't match the performance of its corresponding standard one, i.e. with the same depth but independent convolutional kernels. This reduces the value of RC for model compression. In this paper, we propose a simple variant which improves RC networks: The batch normalization layers of an RC module are learned independently (not shared) for different unrolling steps. We provide insights on why this works. Experiments on CIFAR show that unrolling a convolutional layer several steps can improve the performance, thus indirectly plays a role in model compression.

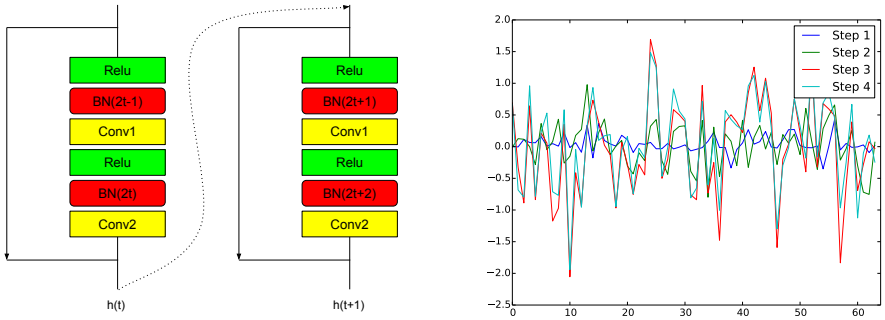
1 Introduction

Deep convolution neural networks (DCNNs) have achieved ground-breaking results on a broad range of fields, such as computer vision [1] and natural language processing [2]. Unfortunately, DCNNs are both computation intensive and memory intensive for industrial applications. Many approaches have been proposed recently to obtain more compact DCNNs while keep their performance as much as possible. Conceptually, those approaches fall in two categories: 1) Reduce the computational cost or memory usage of big DCNNs by weights pruning, quantization and sharing [3, 4, 5, 6]; 2) Improve the performance of small DCNNs by knowledge distillation [7] or other techniques.

In this paper, we explore a potential compression strategy which is complementary to most of the existing approaches: training a recurrent convolutional (RC) neural network. As the name suggests, the same convolutional kernels are unrolled multiple times on the computational graph. This is a weights sharing mechanism applied to the whole layer. Suppose there is a network with n RC layers each of which unrolls k times, then we say its depth is nk. If the performance of this network can match the performance of a standard DCNN with nk layers (Suppose other conditions are the same), we could say we compress the standard one with factor k. Then we can further compress the obtained RC network by applying other existing approaches, such as weight quantization. The key intuition is that RC can reduce the redundancy across layers by sharing weights of the whole layer. While most existing approaches work at a layer-wise manner and only remove part of a layer. This is why we say RC is a complementary strategy.

However, we find the performance of RC networks can't match the performance of DCNNs with the same depth. This significantly reduces the value of RC for model compression. In this paper, we aim to improve the performance of RC in a simple way. Specifically, we learn the batch normalization layers (BN) [8] independently at each unrolling step. We describe our insights in the next section.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.



(a) Unrolling of res-block (b) Moving means

Figure 1: The structure of each res-block is shown in (a). Note the BN layers are not shared. The moving means of the first BN layer over steps are shown in (b).

Experiments on CIFAR dataset demonstrate that such a simple variant works well. We also compare RC networks with their corresponding standard ones.

The idea of RC is not new. Many works have used it to model time-space signals [9] or to obtain a larger receptive field [10]. However, to our knowledge, none of them view RC as a potential model compression strategy and none of them compare RC networks with their corresponding standard ones strictly. The intention of this paper is to show that RC is a considerable or at least a heuristic solution for model compression.

2 Methodology

In this paper, we focus on image classification via DCNNs. Thus there are no sequential inputs for RC layers. The state equation of RC is

$$h_{i+1} = f_{\mathbf{w}}(h_i) \tag{1}$$

where f is a set of differentiable operators including convolutions and f is parameterized by $\mathbf{w}$. Here, we set f to the so-called res-block used in ResNet [1]. Each res-block is a composition of convolutional layers, BN layers and Relu activations [11], as shown in figure 1(a). h_i and h_{i+1} are the input and output of a res-block at ith unrolling step respectively.

When we directly unroll each res-block several steps during training, the performance is not satisfactory. We believe that it is caused by the shared BN layers. A BN layer captures the moving mean and variance of its input, then scales and shifts the normalized input by its learned parameters. However, there are no reasons to expect the statistics of the inputs over different unrolling steps are the same. When we share the BN layers across unrolling steps, we merge those statistics over unrolling steps into a single mean and variance vector. This would hurt the performance of RC networks. Thus, we apply independent BN layers at each unrolling step. Then the number of BN layers is proportional to the number of unrolling steps. Since we focus on image classification (no sequential inputs), we can set the total unrolling steps of each res-block to a fixed number.

Using independent BN layers may also improve the representation power of RC networks. Because the parameters of BN are different across unrolling steps, the overall mapping function at each unrolling step is no longer the same. We can regard the parameters of BN layers as the additional sequential inputs to an RC module. Then the state equation of RC can be reformalized as follows:

$$h_{i+1} = f_{\mathbf{w}_c}(h_i, \mathbf{b}_{i+1}) \tag{2}$$

where $\mathbf{w}_c$ is the shared convolution kernels and $\mathbf{b}_{i+1}$ is the parameters of $(i+1)$th group of BN layers. This is the most general form of the state equation of recurrent neural networks although there is no explicit sequential input. Moreover, using independent BN layers is cheap on both computation and memory point of views, compared with the convolutional layers.

Now we turn to some practical considerations. Due to its recurrent nature, the input size and output size of each RC block should be the same. We change the size of feature map outside RC blocks.

Table 1: Test errors of RC-4 with different BN usage.

	No BN	Shared BN	Independent BN
CIFAR-10	10.37	21.00	7.65
CIFAR-100	38.56	54.52	30.44

Table 2: Comparisons of RC networks and ResNet.

	CIFAR-10	CIFAR-100	Depth	Number of parameters
RC-1	14.77	40.28	6	1.259M
RC-2	8.44	32.51	10	1.260M
RC-4	7.65	30.44	18	1.263M
ResNet-10	7.87	30.74	10	2.514M
ResNet-18	6.92	28.10	18	5.023M

Specifically, we reduce the spatial resolution via average pooling. We reduce the spatial resolution and increase the channels simultaneously via the invertible downsampling operation described in [12], which consists in reorganizing the initial spatial channels into the 4 spatially decimated copies obtainable by 2×2 spatial sub-sampling. To avoid gradient explosion and make the training more stable, we use gradient clip when training RC networks.

3 Experiments

We do three sets of experiments on CIFAR-10 and CIFAR-100 [13] datasets: 1) Study the effects of BN layer usage; 2) Compare RC networks with the standard ones; 3) Compare RC networks with different unrolling steps.

Both CIFAR-10 and CIFAR-100 have 50K training samples and 10K test samples, each of which is a 32×32 color image. The former has 10 classes of images while the later has 100 classes of images. All neural network models are implemented with Pytorch [14]. We train each model 3 times and average its test errors. See B for detailed experimental settings. We choose the res-block in 1(a) as the basic unrolling cell and we set the number of cells of each RC network to 2. Denote RC-i as the network whose cells unroll i steps. We train RC-1, RC-2 and RC-4 in this paper. For fair comparisons with the standard DCNNs, we slightly modify ResNet-18 used in [1] such that its computational graph is exactly the same as the one of unrolled RC-4, except the weight sharing mechanism. See A for detailed network architecture.

As shown in figure 1(b), the moving means of the first BN layer of RC-4 vary across unrolling steps. See C.2 for more results. Further, as shown in table 1, using independent BN layers improves the accuracy of RC-4 by a large margin. Using shared BN layers has even lower accuracy than without using BN layers. Those results support the independent learning of BN layers and our discussions in section 2.

As shown in table 2, the accuracy of RC networks is improved when their cells are unrolled more steps. But when the standard ResNet and the unrolled RC network have the same depth, the former is still better (the gap is not large). It is worth noting that RC-4 whose original depth is 6 has better performance than Resnet-10. We also show the number of parameters of each model trained on CIFAR-10 without further compression. As shown in figure 2, the convergence speed of RC networks and the standard ones looks similar. In another word, RC networks are easy to train.

4 Discussion

We suggest recurrent convolution is a considerable strategy for model compression. RC reduces the redundancy across layers (which is ignored by most of the compression methods). We can train an RC network and then further compress it via existing approaches. We also suggest it is significantly better to learn independent BN parameters at each unrolling step when training RC networks. Experiments on CIFAR dataset demonstrate that unrolling the same convolutional layer several steps can improve the accuracy of the whole network, thus indirectly plays a role in model compression. We believe that the performance of RC could be further improved.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[3] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[4] Karen Ullrich, Edward Meeds, and Max Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.

[5] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin, "Thinet: A filter level pruning method for deep neural network compression," *arXiv preprint arXiv:1707.06342*, 2017.

[6] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[8] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Computer Science*, 2015.

[9] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao, "Recurrent convolutional neural networks for text classification.," in *AAAI*, 2015, vol. 333, pp. 2267–2273.

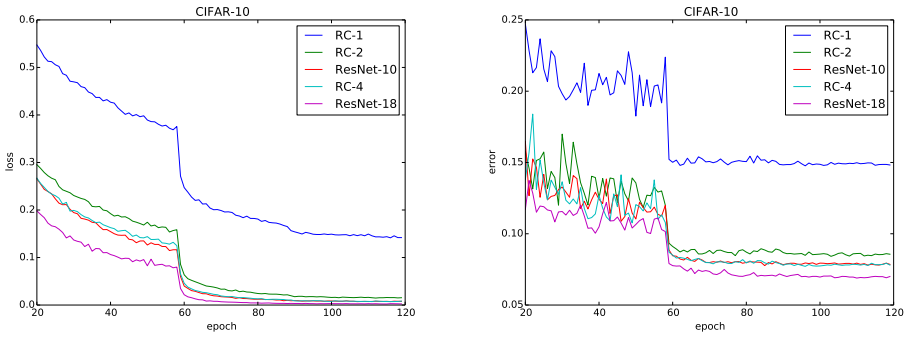
[10] Pedro HO Pinheiro and Ronan Collobert, "Recurrent convolutional neural networks for scene labeling," in *31st International Conference on Machine Learning (ICML)*, 2014, number EPFL-CONF-199822.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, "Deep sparse rectifier neural networks," *Journal of Machine Learning Research*, vol. 15, 2010.

[12] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[13] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.



(a) Training loss (b) Test errors

Figure 2: Training loss and test errors on CIFAR-10 are shown in (a) and (b) respectively. Note that the range of epoch is $[20, 120]$. We remove the first 20 epochs for better visualization. We remove the last 40 epochs for RC-4 and ResNet-18 because the curves have no noteworthy changes.

A Network Architecture

Denote $Block(i, k)$ as the kth unrolling step of an RC res-block with i channels. Denote $InvPool(2)$ as the invertible downsampling operation described in [12]. Then the architecture of RC-4 is:

$$
\begin{gathered}
Conv(64) \to BN(64) \to Relu \to \\
Block(64, 1) \to Block(64, 2) \to AvgPool(2) \to Block(64, 3) \to Block(64, 4) \to \\
InvPool(2) \to \\
Block(256, 1) \to Block(256, 2) \to AvgPool(2) \to Block(256, 3) \to Block(256, 4) \to \\
GlobalAvgPool \to Dense(10/100)
\end{gathered}
$$

For RC-1, $AvgPool(2)$ is applied before the first unrolling step. For RC-2, $AvgPool(2)$ is applied before the second unrolling step. Moreover, we slightly modify the standard ResNet such that its computational graph is exactly the same as the one of unrolled RC network.

B Experimental Details

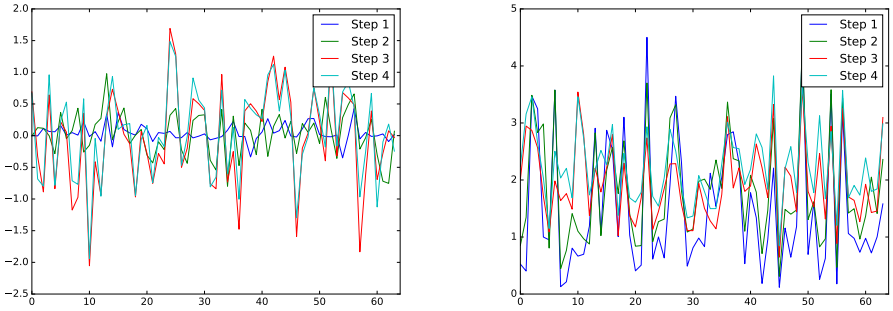
For fair comparisons, most of the hyper-parameters keep the same for all experiments.

We train all of the networks via SGD with moment 0.9 and weight decay 1e-4. Batchsize is set to 128. Initial learning rate is set to 0.1. We reduce the learning rate by factor 10 at 60th epoch and 90th epoch. The networks whose depth are larger than 10 are trained with 160 epochs. While the networks whose depth are smaller or equal to 10 are trained with 120 epochs. See table 2 for the depth of networks. For all experiments, we clip the gradients to $[-0.1, 0.1]$. We find gradient clip makes the training process more stable and improves the performance for RC networks. For standard networks, gradient clip has nearly no effect on their performance. Other hyper-parameters are set to Pytorch's default settings.

C More Experimental Results

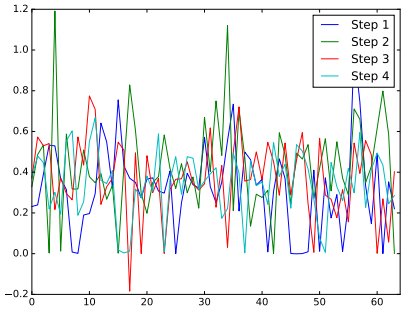
C.1 Convergence Speed

Training loss and test errors on CIFAR-10 are shown in figure 2. The convergence speed of RC networks and standard networks is similar.

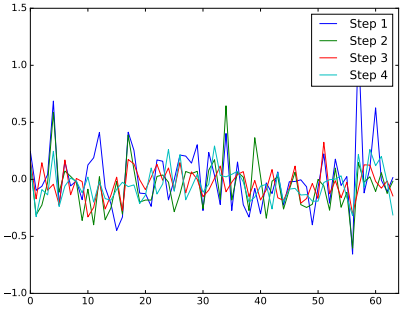


(a) μ

(b) σ^2



(c) γ



(d) β

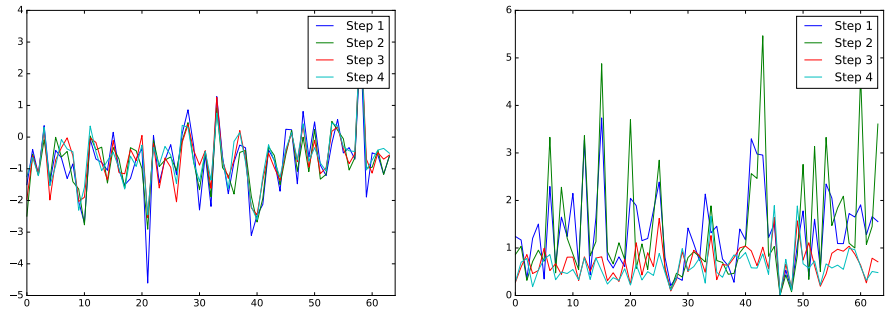
Figure 3: Variables of the first BN layer of RC-4 trained on CIFAR-10 are shown in this figure.

C.2 Parameters of BN

The inference process of a BN layer is formulated as:

$$\gamma \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

where μ is the moving mean and σ^2 is the moving variance. γ and β are the learned parameters by SGD. We show those variables of both the first BN layer and the second BN layer of RC-4 trained on CIFAR-10 in figure 3 and 4 respectively.



(a) μ (b) σ^2

(c) γ (d) β

Figure 4: Variables of the second BN layer of RC-4 trained on CIFAR-10 are shown in this figure.