Deep learning has achieved tremendous success by training increasingly large models, which are then compressed for practical deployment. We propose a drastically different approach to compact and optimal deep learning: We decouple the Degrees of freedom (DoF) and the actual number of parameters of a model, optimize a small DoF with predefined random linear constraints for a large model of an arbitrary architecture, in one-stage end-to-end learning.

Specifically, we create a recurrent parameter generator (RPG), which repeatedly fetches parameters from a ring and unpacks them onto a large model with random permutation and sign flipping to promote parameter decorrelation. We show that gradient descent can automatically find the best model under constraints with in fact faster convergence.

Our extensive experimentation reveals a log-linear relationship between model DoF and accuracy. Our RPG demonstrates remarkable DoF reduction, and can be further pruned and quantized for additional runtime performance gain. For example, in terms of top-1 accuracy on ImageNet, RPG achieves 96% of ResNet18’s performance with only 18% DoF (the equivalent of one convolutional layer) and 52% of ResNet34’s performance with only 0.25% DoF! Our work shows significant potential of constrained neural optimization in compact and optimal deep learning.

1. Introduction

Deep neural networks as general optimization tools have achieved great success with increasingly more training data, deeper and larger neural networks: A recently developed NLP model, GPT-3 [8], has astonishing 175 billion parameters! While the model performance generally scales with the number of parameters [29], with parameters outnumbering training data, the model is significantly over-parameterized.

Many approaches have been proposed to remove redundancy in trained large models: neural network pruning, efficient network design spaces [30, 33, 51], parameter regularization [59, 60, 52, 47], model quantization [51, 50, 43], neural architecture search [70, 10, 58], recurrent models [4, 5, 62], multi-task feature encoding [49, 24], etc. Pruning-based model compression dates back to the late 80s [45, 39] and has enjoyed recent resurgence [23, 7]. They remove unimportant parameters from a pre-trained model and can achieve significant model compression.

Our work is a departure from mainstream approaches towards model optimization and parameter reduction: rather than compressing a large model, we directly optimize a lean model with a small set of free parameters (number of free...
parameters equal to degree of freedom of the model, or DoF), which can be linearly unpacked to a large model. Training the large model can be viewed as solving a neural optimization with a set of predefined linear constraints. One benefit of constrained neural optimization we observe is that it leads to a faster convergence rate (Section 5.6). Specifically, we define different layers in a neural network based on a fixed amount of DoF, which we call recurrent parameter generator (RPG). That is, we differentiate the number of model parameters and DoF. Traditionally, model parameters are treated independently of each other; the total number of parameters equals DoF. However, by tapping into how a core set of free parameters can be assigned to the neural network model, we can develop a large model of many parameters, which are linearly constrained by the small set of free parameters.

There is excess capacity in neural networks independent of how and where the parameters are used in the network, even at the level of individual scalar values. Surprisingly, backpropagation training of a deep network is able to cope with that the same parameter can be assigned to multiple random locations in the network without significantly impacting model performance. Our extensive experiments show that a large neural network does not need to be overparameterized to achieve competitive performance. Particularly, a ResNet18 can be implemented with DoF equivalent to one convolution layer in a ResNet18-vanilla (4.72 × DoF reduction) and still achieves 67.2% ImageNet top-1 accuracy. The proposed method is also extremely flexible in reducing model DoF. In some sense, the proposed RPG method can be viewed as an automatic model DoF reduction technique, which explores the optimal accuracy-parameter trade-off. When we reduce the model DoF, RPG demonstrates graceful performance degradation, and its compression results are frequently on par with the SOTA pruning methods besides the flexibility. Even if we reduce the Res18 backbone DoF to 36K, which is about 300 × reduction, ResNet18 can still achieve 40% ImageNet top-1 accuracy. Further, we show RPG can be quantized and pruned to improve FLOPs and runtime with relatively mild accuracy drops.

To summarize, we make three contributions: 1. We provide a new perspective towards automatic model size reduction: we define a neural network with certain DoF with random linear constraints. We discover that gradient descent can automatically solve constrained optimization for the best model with a faster convergence rate. This constrained neural optimization perspective is likely to benefit many other applications. 2. We propose the recurrent parameter generator (RPG), which decouples the network architecture and the network DoF. We can flexibly choose any desired DoF to construct the network given a specific neural network architecture. 3. By separating network architectures from parameters, RPG becomes a tool to understand the relationship between the model DoF and the network performance. We observe an empirical log-linear DoF-Accuracy relationship.

2. Related Work

Many works study model DoF reduction or compression. We discuss each one and its relationship to our work.

Model Pruning, Neural Architecture Search, and Quantization. Model pruning seeks to remove unimportant parameters in a trained model. Recently, it’s proposed to use neural architecture search as coarse-grained model pruning [68][16]. Another related effort is network quantization [31][50][43], which seeks to reduce the bits used for each parameter and can frequently reduce the model size by 4 × with minimal accuracy drop. More recently, [14] presents a framework for analyzing model scaling strategies that consider network properties such as FLOPs and activations.

Parameter Regularization and Priors. Regularization has been widely used to reduce model redundancy [38][47], al-
leviate overfitting \[52\] \[59\], and ensure desired mathematical regularity \[60\]. RPG can be viewed as a parameter regularization in the sense that weight sharing poses many equality constraints to weights and regularizes weights to a low-dimensional space. HyperNeat \[55\] and CPPNs \[54\] use networks to determine the weight between two neurons as a function of their positions. \[55\] \[34\] introduced a similar idea by providing a hierarchical prior for network parameters.

**Recurrent Networks and Deep Equilibrium Models.** Recurrence and feedback have been shown in psychology and neuroscience to act as modulators or competitive inhibitors to aid feature grouping \[21\], figure-ground segregation \[32\] and object recognition \[65\]. Recurrence-inspired mechanisms also achieve success in feed-forward models. There are two main types of employing recurrence based on if weights are shared across recurrent modules. ResNet \[26\], a representative of reusing similar structures without weight sharing, introduces parallel residual connections and achieves better performance by going deeper in networks. Similarly, some works \[56\] \[53\] also suggest iteratively injecting thus-far representations to the feed-forward network useful. Stacked inference methods \[48\] \[61\] \[63\] \[64\] are also related while they consider each output in isolation. Some find sharing weights across recurrent modules valuable. They demonstrate applications in temporal modelling \[63\] \[66\] \[56\], spatial attention \[44\] \[9\], pose estimation \[62\] \[11\], and so on \[41\] \[69\]. Such methods usually shine in modeling long-term dependencies. In this work, we recurrently share weights across different layers of a feedback network to reduce network redundancy.

Given stacking weight-shared modules improve the performance, researchers consider running even infinite depth of such modules by making the sequential modules converge to a fixed point \[40\] \[4\]. Employing such equilibrium models to existing networks, they show improved performance in many natural language processing \[4\] and computer vision tasks \[5\] \[61\]. One issue with deep equilibrium models is that the forward and backward propagation usually takes much more iterations than explicit feed-forward networks. Some work \[19\] improves the efficiency by making the backward propagation Jacobian free. Another issue is that infinite depth and fixed point may not be necessary or even too strict for some tasks. Instead of achieving infinite depth, our model shares parameters to a certain level. We empirically compare with equilibrium models in Section 5.

**Efficient Network Space and Matrix Factorization.** Convolution is an efficient and structured matrix-vector multiplication. Arguably, the most fundamental idea in building efficient linear systems is matrix factorization. Given the redundancy in deep convolutional neural network parameters, one can leverage the matrix factorization concept, e.g., factorized convolutions, and design more efficient network classes \[30\] \[33\] \[57\] \[51\].

### 3. Recurrent Parameter Generator

**Linearly Constrained Neural Optimization.** Consider optimizing a network with input data \(X\), parameters \(W\) and loss function \(L\). The optimization can be written as:

\[
\min L(X; W) \text{ s.t. } \hat{W} = GW \text{ (or equally } R\hat{W} = 0) \quad (1)
\]

where \(\hat{W} = GW\) refers to a set of linear constraints, where \(G \in \mathbb{R}^{N \times M}\) is a full-rank tall matrix (i.e. \(N \geq M\)). Here we refer to \(W\) as the constrained parameters and \(\hat{W}\) as the free parameters. This constraint is a change of variable, i.e., the constrained parameter \(\hat{W}\) is linearly generated from the free parameter \(W\) by generating matrix \(G\). We can consider \(W\) as a compressed model, which is unpacked into \(\hat{W}\) to construct the large neural network. \(\hat{W}\) is directly optimized via gradient descent and free to update. In this linearly constrained neural optimization, the model DoF is equivalent to \(M\), which is the dimension of \(W\). An equivalent form of the constraint \(\hat{W} = GW\) is \(R\hat{W} = 0\), where \(R \in \mathbb{R}^{(N-M) \times N}\) can be derived from SVD of \(G\).

**Recurrent Parameter Generator.** Let’s assume that we construct a deep convolutional neural network containing \(L\) different convolution layers. Let \(K_1, K_2, \ldots, K_L\) be the corresponding \(L\) convolutional kernels. Rather than using separate sets of parameters for different convolution layers, we create a single set of parameters \(W \in \mathbb{R}^M\) and use it to generate the corresponding parameters \(\hat{W} = \)

---

1A kernel contains all the filters of one layer. In this paper, we treat each convolutional kernel as a vector. When the kernel is used to do the convolution, it will be reshaped into the corresponding shape.
\[ [K_1^T, K_2^T, \ldots, K_L^T]^T \in \mathbb{R}^N \] for each convolution layer:

\[ K_i = G_i \cdot W, i \in \{1, \ldots, L\} \tag{2} \]

where \( G_i \) is a fixed predefined generating matrix, which is used to generate \( K_i \) from \( W \). We call \( G = [G_1^T, \ldots, G_L^T]^T \) and \( W \) the \textit{recurrent parameter generator} (RPG). In this work, we always assume that the size of \( W \) is not larger than the total parameters of the model, i.e., \(|W| \leq \sum_i |K_i|\). This means an element of \( W \) will generally be used in more than one layer of a neural network. Additionally, the gradient of \( W \) is a linear superposition of the gradients from each convolution layer. During the neural network training, let’s assume convolution kernel \( K_i \) receives gradient \( \frac{\partial \ell}{\partial K_i} \), where \( \ell \) is the loss function. Based on the chain rule, it is clear that the gradient of \( W \) is:

\[ \frac{\partial \ell}{\partial W} = \sum_{i=1}^L G_i^T \cdot \frac{\partial \ell}{\partial K_i} \tag{3} \]

**Generating Matrices and Destructive Weight Sharing.**

There are various ways to create the generating matrices \( \{G_i\} \). While in general \( G \) can be any full-rank tall matrix, this paper focuses on the destructive generating matrices, which are random orthogonal matrices and could prevent different kernels from sharing the representation during weight sharing. Random generating matrices empirically improve the model capacity when the model DoF is fixed. We provide an intuitive theoretical explanation of how random orthogonal matrices prevent representation sharing as follows.

For easier discussion, let us consider a special case, where all of the convolutional kernels have the same size and are used in the same shape in the corresponding convolution layers. The dimension of \( W \) is equal to that of one convolutional layer kernel. In other words, \( \{G_i\} \) are square matrices, and the spatial sizes of all of the convolutional kernels have the same size, \( d_{in} \times d_{out} \times w \times h \), and the input channel dimension \( d_{in} \) is always equal to the output channel dimension \( d_{out} \). In this case, a filter \( f \) in a kernel can be treated as a vector in \( \mathbb{R}^{dwh} \). Further, we choose \( G_i \) to be a block-diagonal matrix \( G_i = \text{diag}(A_1, A_2, \ldots, A_1) \), where \( A_i \in O(dwh) \) is an orthogonal matrix that generates each filter of the kernel \( K_i \) from \( W \), and \( O(\cdot) \) denotes the orthogonal group. Similar to the Proposition 2 in [13], we show in the Appendix that: if \( A_i, A_j \) are sampled from the \( O(dwh) \) Haar distribution and \( f_i, f_j \) are the corresponding filters (generated by \( G_i, G_j \) respectively from the same set of entries of \( W \)) from \( K_i, K_j \) respectively, then we have \( E[f_i f_j^T] = 0 \) and \( E[\frac{f_i f_j^T}{\|f_i f_j^T\|}] = \frac{1}{\text{dwh}} \). Since \( dwh \) is usually large, the corresponding filters from \( K_i, K_j \) are close to orthogonal and generally dissimilar. This shows that even when \( \{K_i\} \) are generated from the same entries of \( W \), they are prevented from sharing the representation.

Though \( \{G_i\} \) are not updated during training, the size of \( G_i \) can be quite large in general, which can create additional computation and storage overhead. In practice, we can use permutation and element-wise random sign reflection to construct a subset of the orthogonal group as permutations and sign reflections could be implemented with high simplicity and negligible cost. A simple demonstration of \( \{G_i\} \) is demonstrated in Fig 2. Since pseudo-random numbers are used, it takes only two random seeds to store a random permutation and an element-wise random sign reflection.

**Even Parameter Sampling and Model Ring.** While it is easy to randomly sample elements from \( W \) when generating parameters for each layer, it may not be optimal as some elements in \( W \) may not be evenly used, and some elements in \( W \) used at all due to sampling fluctuation. A simple equalization technique can be used to guarantee all elements of \( W \) are evenly sampled. Suppose the size of \( W \) is \( M \), and the size of parameter \( W \) of the model to be generated is \( N, N > M \). As we mentioned earlier, there are \( L \) layers and they require \( \{|K_1|, \ldots, |K_L|\} \) parameters respectively. As \( N > M \), we can use \( W \) as a ring: we first draw the first \( |K_1| \) parameters from \( W \) followed by a pre-generated random permutation \( p_1 \) and a pre-generated random element-wise sign flipping \( b_1 \) to construct layer-1 kernel \( K_1 \). Then we draw the next \( |K_2| \) parameters from \( W \) followed by a pre-generated random permutation \( p_2 \) and a pre-generated random element-wise sign flipping \( b_2 \). We continue this process and wrap around when there is not enough entries left from \( W \). We refer to \( W \) together with this sampling strategy as \textit{model rings} since the free parameters are recurrently used in a loop. We illustrate the general parameter generator in Fig 2L and RPG in Fig 2L. This For data saving efficiency, we just need to save several random seed numbers instead of saving the pre-generated permutations \( \{p_1, \ldots, p_L\} \) and sign flipping operations \( \{b_1, \ldots, b_L\} \).

**Batch Normalization.** Model performance is relatively sensitive to the batch normalization parameters. For better performance, each convolution layer needs to have its own batch normalization parameters. In general, however, the size of batch normalization is relatively negligible. Yet when \( W \) is extremely small (e.g., 36K parameters), the size of batch normalization should be considered.

4. RPG at Multiple Scales

We discuss the general idea of parameter generators where only one RPG is shared globally across all layers previously. We could also create several local RPGs, each of which is shared at certain scales, such as blocks and subnetworks. Such RPGs may be useful for certain applications such as recurrent modeling.

\footnote{Permutations and element-wise random sign reflection conceptually are subgroups from the orthogonal group, but we shall never use them in the matrix form for the obvious efficiency purpose.}
RPGs at Block-Level. Many existing network architectures reuse the same design of network blocks multiple times for higher learning capacity, as discussed in the related work. Instead of using one global RPG for the entire network, we could alternatively create several RPGs that are shared within certain network blocks. We take Res18 [26] as a concrete example. Res18 has four building blocks. Every block has 2 residual convolution modules. We create four local RPGs for Res18. Each RPG is shared within the corresponding building block, where the size of the RPG is flexible and can be determined by users. Fig 3M illustrates how RPGs can be shared at the block-level.

RPGs at Sub-Network-Level. Reusing sub-networks, or recurrent networks, has achieved success in many tasks as they iteratively refine and improve the prediction. Parameters are often shared when reusing the sub-networks. This may not be optimal as sub-networks at different stages iteratively improve the prediction, and shared parameters may limit the learning capacity at different stages. However, not sharing parameters at all greatly increases the model size. RPG can be created for each sub-network. Such design leads to a much smaller DoF, while parameters of different sub-networks are orthogonal by undergoing destructive changes. We show applications of sub-network-level RPGs for pose estimation and multitask regression (Section 5.3 and 5.4). Fig 3R illustrates sub-network-level RPGs.

5. Experimental Results

We evaluate the performance of RPG with various tasks illustrated in Fig 3. For classification, RPG was used for the entire network except for the last fully-connected layer. We discuss performance with respect to backbone DoF, the actual number of parameters of the backbone. For example, Res18 has 11M backbone parameters and 512K fc parameters, and RPG was applied to reduce 11M backbone DoF only.

5.1. CIFAR Classification

Implementation Details. CIFAR experiments use 128 batch size, 5e-4 weight decay, initial learning rate of 0.1 with gamma of 0.1 at epoch 60, 120 and 160. We use Kaiming initialization [25] with adaptive scaling. Shared parameters are initialized with a particular variance and scale the parameters for each layer to make it match the Kaiming initialization.

Compared to Deep Equilibrium Models. As a representative of implicit models, deep equilibrium models [4] reduce model DoF by finding fix points via additional optimizations. We compare the image classification accuracy on CIFAR10 and CIFAR100, as well as the inference time on CIFAR100 (Table 1). Following the settings of MDEQ [5], an image was sequentially fed into the initial convolutional block, the multi-scale deep equilibrium block (dubbed as MS block), and the classification head. MDEQ [5] achieves infinite MS blocks by finding the fixed point of the MS block. We reuse the MS block two to four times without increasing the model DoF. RPG achieves 3% - 6% gain on CIFAR10 and 3% - 6% gain on CIFAR100. RPG inference time is 15 - 25 times smaller than MDEQ since MDEQ needs additional time to solve equilibrium during training.

Global RPG with Varying Model DoF. We create one global RPG to generate parameters for convolution layers of ResNet and refer to it as ResNet-RPG. We report CIFAR100 top-1 accuracy of ResNet-RPG18 and ResNet-RPG34 at different model DoF (Table 2) and Fig 6 in Appendix B. Compared to ResNet, ResNet-RPG achieves higher accuracy at the same model DoF. Specifically, we achieve 36% CIFAR100 accuracy with only 8K backbone DoF. Further, ResNet34-RPG achieves higher accuracy than ResNet18-RPG, indicating increasing time complexity gives performance gain. We observe log-linear DoF-accuracy relationship, with details in Power Law of the following subsection.

Local RPGs at the Block-Level. In the previous Res-RPG experiments, we use 128 batch size, 5e-4 weight decay, initial learning rate of 0.1 with gamma of 0.1 at epoch 60, 120 and 160. We use Kaiming initialization [25] with adaptive scaling. Shared parameters are initialized with a particular variance and scale the parameters for each layer to make it match the Kaiming initialization.

Compared to Deep Equilibrium Models. As a representative of implicit models, deep equilibrium models [4] reduce model DoF by finding fix points via additional optimizations. We compare the image classification accuracy on CIFAR10 and CIFAR100, as well as the inference time on CIFAR100 (Table 2). Following the settings of MDEQ [5], an image was sequentially fed into the initial convolutional block, the multi-scale deep equilibrium block (dubbed as MS block), and the classification head. MDEQ [5] achieves infinite MS blocks by finding the fixed point of the MS block. We reuse the MS block two to four times without increasing the model DoF. RPG achieves 3% - 6% gain on CIFAR10 and 3% - 6% gain on CIFAR100. RPG inference time is 15 - 25 times smaller than MDEQ since MDEQ needs additional time to solve equilibrium during training.
Table 3: ResNet-RPG consistently achieves higher performance at the same model DoF. We report ImageNet and CIFAR100 top-1 accuracy and backbone DoF for ResNet-vanilla and ResNet-RPG.

<table>
<thead>
<tr>
<th>Model</th>
<th>CIFAR100</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc. (%)</td>
<td>R18-RPG</td>
<td>R18-vanilla</td>
</tr>
<tr>
<td>ImageNet</td>
<td>41.0</td>
<td>67.2</td>
</tr>
<tr>
<td>CIFAR100</td>
<td>70.7</td>
<td>77.4</td>
</tr>
</tbody>
</table>

Table 4: RPG outperforms CPM at the same DoF. We report pose estimation performance (model DoF) on MPII human pose dataset. The metric is PCKh@0.5.

<table>
<thead>
<tr>
<th>Sub-net</th>
<th>CPM [62]</th>
<th>RPG</th>
<th>No shared w.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x sub-net</td>
<td>84.7 (3.3M)</td>
<td>87.3 (3.3M)</td>
<td>88.1 (3.3M)</td>
</tr>
<tr>
<td>2x sub-nets</td>
<td>86.1 (3.3M)</td>
<td>86.5 (3.3M)</td>
<td>87.1 (6.7M)</td>
</tr>
<tr>
<td>4x sub-nets</td>
<td>86.5 (3.3M)</td>
<td>87.3 (3.3M)</td>
<td>88.0 (13.3M)</td>
</tr>
</tbody>
</table>

5.3. Pose Estimation

Implementation Details. We superpose sub-networks for pose estimation with a globally shared RPG. Hourglass networks [46] are used as the backbone. An input image is first fed to an initial convolution block to obtain a feature map, which is then fed to multiple stacked pose estimation sub-networks. Each sub-network outputs a pose estimation prediction, which is penalized by the pose estimation loss. Convolutional pose machine (CPM) [62] share all sub-network weights. We create one global RPG to generate parameters for each sub-network. Our model size is set to the same as CPM. We also compare with larger models where parameters of sub-networks are not shared.

We evaluate on MPII Human Pose dataset [2], a benchmark for articulated human pose estimation, which consists of over 28K training samples over 40K people with annotated body joints. We use the hourglass network [46] as backbone and follow all their settings.

Results and Analysis. We report the Percentage of Correct Key-points at 50% threshold (PCK@0.5) of different methods in Table 4. CPM [62] share all parameters for different sub-networks. We use one RPG that is shared globally at the same size as CPM. For reference, we also compare with the no-sharing model as the performance ceiling. Adding the number of recurrences leads to performance gain for all methods. At the same model size, RPG achieves higher PCK@0.5 compared to CPM. Increasing the number of parameters by not sharing sub-network parameters also leads to some performance gain.

5.4. Multi-Task Regression

Implementation Details. We superpose sub-networks for multi-task regression with multiple RPGs at the building-block level. We focus on predicting depth and normal maps.

Table 5: RPG achieves the best accuracy without sharing batch normalize parameters and with permutation and sign reflection. We report multitask regression errors on S3DIS with sub-net architecture as [49]. Lower is better. All methods share the same DoF. Sub-net is reused once.

<table>
<thead>
<tr>
<th>RMSE (%)</th>
<th>Depth</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla model</td>
<td>25.5</td>
<td>41.0</td>
</tr>
<tr>
<td>RPG with shared BN</td>
<td>24.7</td>
<td>40.3</td>
</tr>
<tr>
<td>Reuse &amp; new BN</td>
<td>24.0</td>
<td>39.4</td>
</tr>
<tr>
<td>Reuse &amp; new BN &amp; perm. and reflect.</td>
<td>22.8</td>
<td>39.1</td>
</tr>
</tbody>
</table>
A log-linear DoF-accuracy relationship exists for RPGs applied to vision transformer ViT \cite{17}. b) RPG converges faster than the vanilla model. We plot the CIFAR10 accuracy (smoothed by moving average) versus training iterations for Res18-vanilla and Res18-RPG. RPG converges at 1k iterations while the vanilla model converges at 1.7k. c) RPG consistently converges faster. The reduction becomes substantial with the increasing batchsize, e.g., at batchsize 1024, RPG takes 41\% less iterations to converge. Denote final accuracy as $P_f$, the convergence iteration is defined when current smoothed accuracy (by moving average) is within 5\% range of $P_f$.

Table 6: RPG achieves higher post-pruning CIFAR10 accuracy and similar post-pruning accuracy drops as SOTA fine-grained pruning approach IMP \cite{18}. Fine-grained pruning is used for reducing DoF.

<table>
<thead>
<tr>
<th>DoF before pruning</th>
<th>Pruned acc.</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R18-IMP \cite{19}</td>
<td>92.3</td>
<td>90.5</td>
</tr>
<tr>
<td>R18-RPG</td>
<td>95.0</td>
<td>93.0</td>
</tr>
</tbody>
</table>

Table 7: RPG achieves similar post-pruning ImageNet performance as SOTA coarse-grained approach Knapsack \cite{1} at the same FLOPs. Coarse-grained pruning is used for reducing RPG FLOPs.

<table>
<thead>
<tr>
<th>DoF before pruning</th>
<th>Pruned acc.</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R18-Knapsack</td>
<td>11.2M</td>
<td>69.35%</td>
</tr>
<tr>
<td>Pruned R18-RPG</td>
<td>5.6M</td>
<td>69.10%</td>
</tr>
</tbody>
</table>

from a given image. We stack multiple SharpNet \cite{49}, a network for monocular depth and normal estimation. Specifically, we create multiple RPGs at the SharpNet building-block level. That is, parameters of corresponding blocks of different sub-networks are generated from the same RPG.

We evaluate the monocular depth and normal prediction performance on a 3D indoor scene dataset \cite{3}, which contains over 70K images with corresponding depths and normal maps covering over 6,000 m$^2$ indoor area. We follow all settings of SharpNet \cite{49}, a SOTA monocular depth and normal estimation method.

**Results and Analysis.** We report the mean square errors for depth and normal estimation in Table 5. Compared to one-time inference without recurrence, our RPG network gives 3\% and 2\% gain for depth and normal estimation, respectively. Directly sharing weights but using new batch normalization layers decreases the performance by 1\% and 0.3\% for depth and normal. Sharing weights and normalization layers further decrease the performance by 0.7\% and 0.9\% for depth and normal.

**5.5. Pruning RPG**

**Fine-Grained Pruning.** Fine-grained pruning methods aim to reduce the model DoF by sparsifying weight matrices. Such methods usually do not reduce the inference speed, although custom algorithms \cite{20} may improve the speed. At the same model DoF, RPG outperforms state-of-the-art fine-grained pruning method IMP \cite{18}. Accuracy drops of RPG and IMP are similar, both around 2\% (Table 6). It is worth noting that although IMP has no run time improvement in regular settings, it could save inference time with customized sparse GPU kernels \cite{20}.

**Coarse-Grained Pruning.** While RPG is not designed to reduce FLOPs, it can be combined with coarse-grained pruning to reduce FLOPs. We prune RPG filters with the lowest $\ell_1$ norms. Table 7 shows that the pruned RPG achieves on-par performance as state-of-the-art coarse-grained pruning method Knapsack \cite{1} at the same FLOPs.

**5.6. Analysis**

**Convergence rate.** Compared with the vanilla model, RPG optimizes in a parameter subspace $W = GW$ with fewer DoF. Would such constrained optimization lead to a faster convergence rate? We analyze the convergence rate of Res18-vanilla and Res18-RPG (DoF is 5.5M, 50\% of the vanilla model) with different batchsizes. All models are trained with multi-step SGD optimizer and they all reach > 94.1\% final CIFAR10 accuracy. For simplicity, we analyze the first optimization stage where learning rate has not decayed.

Fig. 5 plots the accuracy (smoothed with moving averages) v.s. training iterations with batchsize 1024. RPG has a faster convergence rate than vanilla models. We also analyze the smoothed accuracy and identify the convergence iteration versus batchsize in Fig. 5. RPG consistently converges faster than the vanilla model, and the reduction becomes substantial with the increasing batchsize.

**Comparison to Model Compression Methods.** We report ResNet-RPG performance with different model DoF and existing compression methods on ImageNet (Fig. 1). RPG networks outperform SOTA methods such as \cite{11, 16, 28, 27, 15, 17}. For example, at the same model DoF, our RPG...
network has 0.6% gain over the knapsack pruning \[1\], a SOTA method of ImageNet pruning.

**Storage.** RPG models only need to save the effective parameter \(W\), which has the size of the model DoF, since the generation matrix \(G\) is saved as a random seed at no cost. The storage space of the model file can be diminished to satisfy a smaller storage limit for inference and a faster model file transfer. Empirically on PyTorch platform, ResNet18-vanilla model file is 45MB. With no accuracy loss, ResNet18-RPG model save file size is 23MB (\(\downarrow 49\%\)). With 2 percentage point accuracy loss, RPG save file size is 9.5MB (\(\downarrow 79\%\)).

**Generalizability.** We report the performance gap between training and validation set on ImageNet (Table 8(a)) and MPII pose estimation (Table 8(b)). CPM \[62\] serves as the baseline pose estimation method. RPG models consistently achieve lower gaps between training and validation sets, indicating the RPG model suffers less from over-fitting.

We also report the out-of-distribution performance of RPG models. ObjectNet \[6\] contains 50k images with 113 classes overlapping with ImageNet. Existing models are reported to have a large performance drop on ObjectNet. We directly evaluate the performance of ImageNet-trained model on ObjectNet without any fine-tuning (Table 8(c)). With the same backbone DoF, R18-RPG achieves a 3% gain compared to R18-vanilla. With the same network architecture design, R34-RPG achieves 0.5% gain compared to R34. This indicates RPG networks have higher out-of-distribution performance even with smaller model DoF.

**Quantization.** Network quantization can reduce model size with minimal accuracy drop. It is of interest to study if RPG models, whose parameters have been shrunk, can be quantized. After 8-bit quantization, the accuracy of ResNet18-RPG (5.6M DoF) only drop 0.1 percentage point on ImageNet, indicating RPG can be quantized for further model size reduction. Details are in Appendix A.

**Security.** Permutation matrices generated by the random seed can be considered as security keys to decode the model. Further, only random seeds to generate generating matrix \(G\) need to be saved and transferred at negligible cost.

### 5.7. Ablation Studies

We conduct ablation studies on CIFAR100 to analyze functions of permutation and reflection matrices (Fig 1b). We evaluate ResNet-RPG34 with 2M backbone DoF. Permutation and sign reflection together achieves 76.5% accuracy, while permutation only achieves 75.8%, and sign reflection only achieves 71.1%. Training with neither permutation nor reflection matrices achieves 70.7%. This suggests permutation and sign reflection matrices increase RPG performance.

### 6. Discussion

The common practice in neural network compression is to prune weights from a trained large model with many parameters or degrees of freedom (DoF). Our key insight is that a direct and drastically different approach might work faster and better: We start from a lean model with a small DoF, which can be linearly unpacked into a large model with many parameters. Then we can let the gradient descent automatically find the best model under the linear constraints. Our work is a departure from mainstream approaches towards model optimization and parameter reduction. We show how the model DoF and actual parameter size can be decoupled: we can define an arbitrary network of an arbitrary DoF.

We limit our scope to optimization with random linear constraints, termed destructive weight sharing. However, in general, there might also exist nonlinear RPGs and efficient nonlinear generation functions to create convolutional kernels from a shared model ring \(W\). Further, although RPG focuses on reducing model DoF, it can be quantized and pruned to further reduce the FLOPs and runtime.

To sum up, we develop an efficient approach to build an arbitrarily complex neural network with any amount of DoF via a recurrent parameter generator. On a wide range of applications, including classification, pose estimation and multitask regression, we show RPG consistently achieves higher performance at the same model DoF. Further, we show such networks converge faster, are less likely to overfit and have higher performance on out-of-distribution data.

RPG can be added to any existing network flexibly with any amount of DoF at the user’s discretion. It provides new perspectives for recurrent models, equilibrium models, and model compression. It also serves as a tool for understanding relationships between network properties and network DoF by factoring out the network architecture.

### References


Alvin Wan, Xiaoliang Dai, Peihzho Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Pnetv2: Differentiable neural architecture


