Identity Connections in Residual Nets Improve Noise Stability

Shuzhi Yu [1] **Carlo Tomasi** [1]

Abstract

Residual Neural Networks (ResNets) achieve state-of-the-art performance in many computer vision problems. Compared to plain networks without residual connections (PlnNets), ResNets train faster, generalize better, and suffer less from the so-called degradation problem. We introduce simplified (but still nonlinear) versions of ResNets and PlnNets for which these discrepancies still hold, although to a lesser degree. We establish a 1-1 mapping between simplified ResNets and simplified PlnNets, and show that they are exactly equivalent to each other in expressive power for the same computational complexity. We conjecture that ResNets generalize better because they have better noise stability, and empirically support it for both simplified and fully-fledged networks.

1. Introduction

Deep convolutional neural networks have rapidly improved their performance in image classification tasks in recent years. So-called *residual networks* (He et al., 2016), in particular, differ from regular ones by the mere addition of an identity function to certain layers. It has been shown this small change makes residual networks train faster than their plain counterparts, in that they converge to the same training risk in fewer epochs, and reduce the so-called *degradation problem* (He et al., 2016), that is, their training risk increases more slowly as new layers are added. In addition, ResNets generalize better, that is, they achieve lower classification risk for the same training risk.

In an attempt to understand why a small change in network architecture leads to the advantages listed above, we introduce a formal transformation between simplified versions of residual networks and their nearest kin, which we call (simplified) *plain networks*. Specifically, for every simplified residual network we show how to construct a simplified plain network of equal parametric complexity, and *vice versa*, such that the two networks are exactly equivalent to each other, if exact arithmetic is used. By "equivalent" we mean that the two networks implement the same input/output relation. Thus, the differences in performance between plain and residual networks relate to the initialization and training, instead of expressive power. In summary, *simplified residual networks are equivalent to simplified plain networks of equal parametric complexity initialized in different ways for training.*

The simplification we made relative to the networks used in the literature amounts to removing batch normalization layers and allowing for skip connections (the identity added to residual networks) only across individual layers, rather than across groups of layers. However, our simplified architectures exhibit qualitatively similar differences between plain and residual versions as the fully-fledged ones do, albeit to a lesser degree. Because of this, our insights may help understand commonly used architectures also, a claim we support with experiments on fully-fledged networks.

To explain the advantages above, we observe that adding an identity across a layer makes some of the layer's weights close to 1, a value much higher than the small random values used for initializing either type of network (plain or residual). We show empirically that the large weights present in residual networks decrease the relative sensitivity of the output of a network to changes in layer parameters. It has been shown recently (Arora et al., 2018) that such a decrease in sensitivity (increase in *noise stability*) is a good predictor of improved generalization. In addition, we empirically observe better noise stability in the fully-fledged residual networks as well.

Some recent theoretical work analyzes linear ResNets, from which ReLUs are removed (Hardt & Ma, 2017; Kawaguchi, 2016; Li et al., 2016; Zaeemzadeh et al., 2018) or analyzes shallow nonlinear ResNets (Orhan & Pitkow, 2018). In addition, alternative views of ResNets have been proposed, such as the *unraveled view* (Veit et al., 2016; Littwin & Wolf, 2016; Huang et al., 2016; Abdi & Nahavandi, 2016), the *unrolled iterative estimation view* (Greff et al., 2017;

Material in this thesis is based upon work supported by the National Science Foundation under Grant No. CCF-1513816.

[1]Department of Computer Science, Duke University, Durham, NC, USA. Correspondence to: Shuzhi Yu <shuzhiyu@cs.duke.edu>.

ICML 2019 Workshop on Understanding and Improving Generalization in Deep Learning, Long Beach, California, 2019. Copyright 2019 by the author(s).

Jastrzebski et al., 2018), the *dynamic system view* (Chang et al., 2018b;a; Liao & Poggio, 2016), and the *ensemble view* (Huang et al., 2018). In contrast, we analyze simplified but deep and non-linear networks. Our work does not reason by analogy, but instead explores properties of skip connections through a mapping between ResNets and PlnNets.

Previous work has shown the importance of weight initialization for training deep neural networks and proposed good initialization algorithms (Mishkin & Matas, 2016; He et al., 2015; Glorot & Bengio, 2010; Krähenbühl et al., 2016). We show that a ResNet is a PlnNet with a special initialization, and that the resulting weights, which involve entries much larger than in previous studies, improve generalization ability.

Noise stability A recent paper (Arora et al., 2018) introduces bounds for the ability of a network to generalize that are tighter than standard measures of hypothesis-space complexity are able to provide. They compress a network to a simpler, approximately equivalent one, and tie generalization bounds to a count of the number of parameters of the simpler network. Compressibility, and therefore generalization, improve if the network is *noise-stable*, in the sense that perturbations of its parameters do not change the input/output function implemented by the network too much. We show empirically that (fully fledged) ResNets are more noise-stable than their plain counterparts.

2. Simplified PlnNets and ResNets

A *Plain Neural Network* (PlnNet) and a *Residual Neural Network* (ResNet) are the concatenation of *plain blocks* and *residual blocks* respectively, each of which computes a transformation between input $\mathbf{x}$ and output $\mathbf{y}$ of the following form respectively:

$$\mathbf{y} = f(p(\mathbf{x}, \mathbf{p})) \quad \text{and} \quad \mathbf{y} = f(\mathbf{x} + p(\mathbf{x}, \mathbf{r}))$$

where function f is a ReLU nonlinearity. In this expression, p is a cascade of one or more of the standard blocks used in convolutional neural networks, and has parameter vector $\mathbf{p}$ and $\mathbf{r}$ respectively. Thus, a residual block merely adds an identity connection from the input $\mathbf{x}$ to the output p of a plain block.

A specific architecture of the residual block that contains two convolutional layers has been shown empirically to outperform several other variants. However, the goal of our investigation is not to explain the performance of the best architecture, but rather to compare plain networks with residual networks. We have found empirically that even a drastically simplified block architecture (shown in Figure 5) preserves, to a lesser but still clear extent, the advantages of ResNets over PlnNets mentioned above (see Supplementary Material (SM) Section A). For example, the classification error for a 20- and 44-layer simplified PlnNets are 0.1263 and 0.2213 respectively. In comparison, those of a 20- and 44-layer simplified ResNets are 0.1131 and 0.1182 respectively. Because of this, in this paper we compare cascades of *depth-1* plain blocks of the form in Figure 5(a) with cascades of depth-1 residual blocks of the form in Figure 5(b).

More specifically, PlnNets (or ResNets) of L layers are for us the concatenation of a convolutional layer with ReLU, $3N$ plain (or residual) blocks, an average pooling layer and a final, fully-connected layer followed by a soft-max layer (more details in SM Figure 4). All the filters have size 3×3 in our architecture.

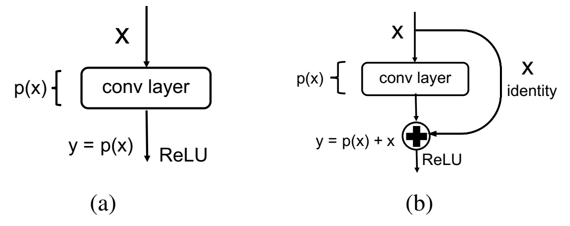


Figure 1. Simplified (a) plain block and (b) corresponding residual block.

3. Simplified PlnNets and ResNets are Equivalent

We now show that *the simplified plain and residual blocks are exactly equivalent to each other*, in the sense that a plain block with convolutional layer $p(\mathbf{x}, \mathbf{p})$ can be made to have the exact same input/output relationship of a residual block with convolutional layer $r(\mathbf{x}, \mathbf{r}) = \mathbf{x} + p(\mathbf{x}, \mathbf{r})$ as long as the parameter vectors $\mathbf{p}$ and $\mathbf{r}$ relate to each other as follows.

Consider a plain network block p with C input channels, whose convolutional layer has D kernels (and therefore D output channels) of width $U = 2U_h + 1$ and height $V = 2V_h + 1$ for nonnegative integers U_h and V_h (the *half-widths* of the kernel). Then, the set of kernels can be stored in a $D \times C \times U \times V$ array. We index the first two dimensions by integers in the intervals $[1, D]$ and $[1, C]$, and the last two in the intervals $[-U_h, U_h]$ and $[-V_h, V_h]$.

We define the *ID entries* of that set of kernels to be the kernel coefficients in positions $(c, c, 0, 0)$ of the array, for $c = 1, \ldots, \min(C, D)$. These are the entries that identities are added to in residual networks, and therefore the plain network p and residual network r implement the same function if

$$\mathbf{p} \;=\; \mathcal{T}(\mathbf{r}) \;=\; \mathbf{r} + \boldsymbol{\rho} \quad \text{where} \quad \boldsymbol{\rho} = \begin{cases} 1 & \text{for all ID entries} \\ 0 & \text{elsewhere.} \end{cases} \tag{1}$$

Of course this transformation is easily invertible:

$$\mathbf{r} \;=\; \mathcal{T}^{-1}(\mathbf{p}) \;=\; \mathbf{p} - \boldsymbol{\rho}$$

so that a plain network can be transformed to an equivalent residual network as well.

The transformations $\mathcal{T}$ and $\mathcal{T}^{-1}$ are extended from blocks to networks by applying the block transformation to each of the network's $3N$ blocks in turn.

A crucial consequence of this equivalence is that the better performance of residual networks over plain networks, which manifests itself even when the networks have their simplified form, does not derive from differences in expressive power between the two architecture. ***Simplified residual networks are simplified plain networks that are trained into a different local minimum of the same optimization landscape.***

4. Training Plain and Residual Networks

As commonly done, we define the loss of a network as the cross-entropy between prediction and truth at the output of the final soft-max layer, and the training risk is the average loss over the training set. In practice, it is common in the literature to add *weight decay*, an L_2 regularization term, to the risk function to penalize large parameter values and improve generalization. (Hinton, 1987; Krogh & Hertz, 1992)

4.1. Equivalent Training of Equivalent Networks

With slight abuse of notation, we henceforth denote with p and r the transformation performed by an entire network (plain or residual), rather than by a single block.

If a plain network p and the corresponding residual network r have parameter vectors $\hat{\mathbf{p}}$ and $\hat{\mathbf{r}}$, respectively, the functions $p(x, \hat{\mathbf{p}})$ and $r(x, \hat{\mathbf{r}})$ the two networks implement are different. However, if the initial parameter vectors $\mathbf{p}_0$ and $\mathbf{r}_0$ satisfy the equation

$$\mathbf{p}_0 = \mathcal{T}(\mathbf{r}_0) \; , \tag{2}$$

the two networks $p(\mathbf{x}, \mathbf{p}_0)$ and $r(\mathbf{x}, \mathbf{r}_0)$ are input-output-equivalent to each other.

In addition, since subsets of corresponding layers in p and r are also equivalent to each other, inspection of the forward and backward passes of back-propagation shows immediately that the gradient of the training risk function with respect to $\mathbf{p}$ in p is the same as that with respect to $\mathbf{r}$ in r. Thus, if training minimizes the training risk, and the sequence of training sample mini-batches is the same for both training histories, a plain network p and a residual network r whose initial weights satisfy equation 2 remain equivalent at all times throughout training.

To ensure equivalence between plain and residual networks even in the presence of weight decay, the penalty $\|\mathbf{p}\|^2$ for residual networks is replaced by $\|\mathcal{T}^{-1}(\mathbf{p})\|^2$. In this way, if plain network p and residual network r are equivalent, both their costs and the gradients of their cost are equal to each other as well (experiment details in SM Section B). We can therefore draw the following conclusion.

If exact arithmetic is used, training plain network p with initialization $\mathbf{p}_0 = \mathcal{T}(\mathbf{r}_0)$ and weight-decay penalty $\|\mathcal{T}^{-1}(\mathbf{p})\|^2$ is equivalent to training residual network r with initialization $\mathbf{r}_0$ and weight-decay penalty $\|\mathbf{r}\|^2$. If the two networks are trained with the same sequence of mini-batches, the plain and residual networks $p(\mathbf{x}, \mathbf{p}_e)$ and $r(x, \mathbf{r}_e)$ at epoch e during training are equivalent to each other for every e, and so are, therefore, the two networks obtained at convergence.

4.2. ResNets are PlnNets with Large ID Entries

The equivalence established in the previous Section allows viewing a simplified residual network as a simplified plain network with different weights:

$$r(\mathbf{x}, \mathbf{r}) = p(\mathbf{x}, \mathcal{T}(\mathbf{r})) \; .$$

Because of this, instead of comparing a plain network $p(\mathbf{x}, \mathbf{p})$ with a residual network $r(\mathbf{x}, \mathbf{r})$, we can compare the two plain networks $p(\mathbf{x}, \mathbf{p})$ and $p(\mathbf{x}, \mathcal{T}(\mathbf{r}))$ to each other. These have the exact same architecture and landscape, but their parameter vectors are initialized with different weights, and end up converging to different local minima of the same risk function.

A first insight into this comparison can be gleaned by comparing the distributions of network weights (excluding biases) for $p(\mathbf{x}, \mathbf{p}_e)$ and $p(\mathbf{x}, \mathcal{T}(\mathbf{r}_e))$ at epoch e of training. For notational simplicity, we define the *transferred parameter vector* $\mathbf{t} = \mathcal{T}(\mathbf{r})$. We initialize the two networks $\mathbf{p}_0$ and $\mathbf{t}_0$ with Kaiming Weight Initialization (KWI) method (He et al., 2015) and Hartz-Ma Weight Initialization (HMWI) method (Hardt & Ma, 2017) respectively (Details in SM Section A). Since network equivalence is preserved during training, we can also write $\mathbf{t}_e = \mathcal{T}(\mathbf{r}_e)$ for every epoch e, so we compare plain networks $p(\mathbf{x}, \mathbf{p}_e)$ and $p(\mathbf{x}, \mathbf{t}_e)$.

Initially ($e = 0$), the distribution $\chi(z, \mathbf{p}_0)$ of $\mathbf{p}_0$ is a zero-mean Gaussian distribution with variance σ_p^2, for the weights. The distribution $\chi(z, \mathbf{t}_0)$ of the initial transferred weights $\mathbf{t}_0$ is a mixture of two components: A Gaussian with zero mean and variance σ_r^2 for weights other than the ID entries, and a Gaussian with mean 1 and variance σ_r^2 for the ID entries. These distributions change during training. Incidentally, and interestingly, we have observed empirically that these changes are small relative to 1 for the example of networks $p(\mathbf{x}, \mathbf{p}_e)$ and $p(\mathbf{x}, \mathbf{t}_e)$ of depth 32, so training does not move weights very much within the optimization landscape (more details in SM Section C).

Stated differently: *The training histories for (i) a plain network initialized by KWI and (ii) a plain network of equal architecture but equivalent to a residual network initialized* *by HMWI evolve in relatively small neighborhoods of their starting points in parameter space. The two neighborhoods are far apart, separated in large measure by the size of the ID entries in the two networks.*

5. Large ID Entries Improve Noise Stability

The noise stability of a network (Arora et al., 2018) is measured by how much the output changes when noise is added to the weights of some layer. More specifically, the *cushion* for layer ℓ is an increasing function of a layer's noise stability, and is defined as the largest number μ_ℓ such that for any data point $\mathbf{x}$ in the training set,

$$\mu_\ell \left\|A^\ell\right\|_F \left\|\mathbf{x}^{\ell-1}\right\| \leq \left\|A^\ell \mathbf{x}^{\ell-1}\right\| \ .$$

In this expression, A^ℓ represents the weights in the layer, $\mathbf{x}^{\ell-1}$ is the output of previous layer (after the ReLU), and $\|\cdot\|$ and $\|\cdot\|_F$ are the 2-norm and Frobenius norm.

Because plain networks equivalent to residual networks contain large weights, the same amount of noise added to such a network has a smaller effect on the output than for a plain network initialized in the traditional way. Thus, residual networks have greater layer cushions and greater noise stability overall, which leads to better generalization when achieving the same loss. As shown in Figure 2, with same training loss, the ResNets achieve smaller error.

Empirical measurements support this view. Specifically, Figure 3 shows the distribution of the per-sample cushion $\mu_\ell(\mathbf{x}^{\ell-1}) = \left\|A^\ell \mathbf{x}^{\ell-1}\right\| / \left\|A^\ell\right\|_F / \left\|\mathbf{x}^{\ell-1}\right\|$ for the 10^{th} layer of a pre-trained 20-layer and 32-layer plain and residual network. To make this Figure, we measure the Frobenius norm of the weights of ResNets after removing 1 from the ID entries, because the intent of the term $\left\|A^\ell\right\|_F$ is to measure *change* in the coefficients. The plots in the figure show that ResNets have better noise stability than PlnNets do, as the red distribution is to the right of the blue distribution. This pattern holds for all the layers, with the exception of 2 layers (in both the 20-layer and 32-layer networks). This result implies (Arora et al., 2018) that ResNets are more likely to generalize better than PlnNets do. In summary, the large ID entries make ResNets less vulnerable to noise in its weights, leading to better generalization.

The better noise stability holds not only for simplified networks, but also for the fully-fledged ones. To show this, we measure the per-sample interlayer cushion $\mu_{i,j}(\mathbf{x}^i)$ of plain and residual blocks in commonly used networks. Given layers $i \leq j$,

$$\mu_{i,j}(\mathbf{x}^i) = \sqrt{n^{(i)}} \left\|J^{i,j}_{\mathbf{x}^i}\mathbf{x}^i\right\| / \left\|J^{i,j}_{\mathbf{x}^i}\right\|_F / \left\|\mathbf{x}^i\right\|$$

where $n^{(i)}$ is the size of the input $\mathbf{x}^i$ to layer i, and $J^{i,j}_{\mathbf{x}^i}$ is the Jacobian of the part of the network from layer i to layer j. Experimental results show that $\mu_{i,j}(\mathbf{x}^i)$ for a residual block is much bigger than that of the corresponding plain block. For example, the interlayer cushion of the 5^{th} out of 9 residual block of a fully-fledged residual network of depth 20 has a mean 0.20 and standard deviation 0.0024. In comparison, the interlayer cushion of the corresponding plain block has a mean 2.7×10^{-7} and standard deviation 2.4×10^{-8}. The plain block in this case is the concatenation of two groups of convolutional layer, Batch Normalization layer, and ReLU; the residual block adds the input of the first convolutional layer to the output of second Batch Normalization layer of the plain block.

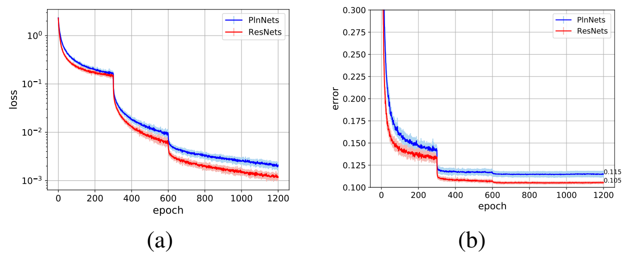


Figure 2. Plots of risk $\mathcal{L}$ (left) and error $\mathcal{E}$ (right) as a function of training time for plain (blue) and residual (red) networks of depth 20 after training for 1200 epochs on the CIFAR-10 dataset. The results are based on training 13 plain networks and 13 residual networks initialized at random (but equivalently to each other), to estimate variance in the results. The sequence of training samples are also randomized. Dark color refers to the mean and the half length of the error bar represents the standard deviation.

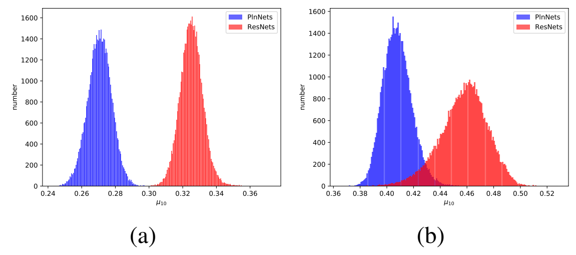


Figure 3. Histogram of the per-sample layer cushion in the 10^{th} layer of 20-layer (left) and 32-layer (right) PlnNets (blue) and ResNets (red). All the four networks are trained for 200 epochs on CIFAR-10 dataset. The test errors for 20-layer PlnNet and ResNet are 0.1263 and 0.1122 respectively, and those for 32-layer networks are 0.1349 and 0.1121 respectively.

6. Conclusions

We propose a one-to-one map between simplified ResNets and PlnNets, and we show how to train corresponding networks equivalently. We conjecture that ResNets achieve lower generalization error than PlnNets because the large ID entries make the former more stable against noise added to the weights. Our experiments support our conjectures for both simplified and fully-fledged networks.

References

Abdi, M. and Nahavandi, S. Multi-residual networks: Improving the speed and accuracy of residual networks. *arXiv preprint arXiv:1609.05672*, 2016.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. Stronger generalization bounds for deep nets via a compression approach. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 254–263, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018a.

Chang, B., Meng, L., Haber, E., Tung, F., and Begert, D. Multi-level residual networks from dynamical systems view. In *International Conference on Learning Representations*, 2018b.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pp. 249–256, May 2010.

Greff, K., Srivastava, R. K., and Schmidhuberr, J. Highway and residual networks learn unrolled iterative estimation. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Hardt, M. and Ma, T. Identity matters in deep learning. *In the International Conference on Learning Representations (ICLR)*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

Hinton, G. E. Learning translation invariant recognition in a massively parallel networks. In *PARLE Parallel Architectures and Languages Europe: Volume I: Parallel Architectures*, pp. 1–13, 1987.

Huang, F., Ash, J., Langford, J., and Schapire, R. Learning deep ResNet blocks sequentially using boosting theory. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2058–2067, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pp. 646–661. Springer, 2016.

Jastrzebski, S., Arpit, D., Ballas, N., Verma, V., Che, T., and Bengio, Y. Residual connections encourage iterative inference. In *International Conference on Learning Representations*, 2018.

Kawaguchi, K. Deep learning without poor local minima. In *Advances In Neural Information Processing Systems*, pp. 586–594, 2016.

Krähenbühl, P., Doersch, C., Donahue, J., and Darrell, T. Data-dependent initializations of convolutional neural networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, May 2016.

Krogh, A. and Hertz, J. A. A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippmann, R. P. (eds.), *Advances in Neural Information Processing Systems 4*, pp. 950–957. Morgan-Kaufmann, 1992.

Li, S., Jiao, J., Han, Y., and Weissman, T. Demystifying resnet. *arXiv preprint arXiv:1611.01186*, 2016.

Liao, Q. and Poggio, T. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

Littwin, E. and Wolf, L. The loss surface of residual networks: Ensembles and the role of batch normalization. *arXiv preprint arXiv:1611.02525*, 2016.

Mishkin, D. and Matas, J. All you need is a good init. In *Proceedings of International Conference on Learning Representations (ICLR)*, May 2016.

Orhan, A. E. and Pitkow, X. Skip connections eliminate singularities. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

Veit, A., Wilber, M., and Belongie, S. Residual networks behave like ensembles of relatively shallow networks. *Conference on Neural Information Processing Systems*, pp. 550–558, 2016.

Zaeemzadeh, A., Rahnavard, N., and Shah, M. Norm-preservation: Why residual networks can become extremely deep? *arXiv preprint arXiv:1805.07477*, 2018.

A. Empirical Baseline

Our experiments are conducted on CIFAR-10 data set that contains 50k training samples and 10k testing samples from 10 classes. Each image in the data set is RGB image of size 32×32. The simplified network architectures are shown in Figure 4. PlnNets (or ResNets) of L layers are for us the concatenation of a convolutional layer with ReLU, three groups, namely G_0, G_1, and G_2, with N plain (or residual) blocks each, an average pooling layer and a final, fully-connected layer followed by a soft-max layer. There are $L = 3N + 2$ trainable layers in total. All the filters have size 3×3 in our architecture. Each of the first $N + 1$ convolutional layers has 16 filters. The first convolutional layer in both G_1 and G_2 doubles the number of output channels but halves the input width and height; the rest of the convolutional layers keep the size of input and output the same. The output of the last convolutional layer has 64 channels with each channel of size 8×8. The average pooling layer has window size of 8×8 so that the result after pooling layer is a vector of 64 entries. The final output of the networks is a vector of size 10×1 that represents soft-max scores for the 10 classes.

As an initial empirical baseline, we trained simplified PlnNets and ResNets (whose building blocks are shown in Figure 5 (a) and (b) respectively) of depths 20, 32, and 44. Plain networks are typically initialized by the Kaiming Weight Initialization (KWI) method (He et al., 2015). In this method, biases are set to zero, and the weights of a $C \times U \times V$ convolutional kernel are set to samples from a Gaussian distribution with mean 0 and standard deviation

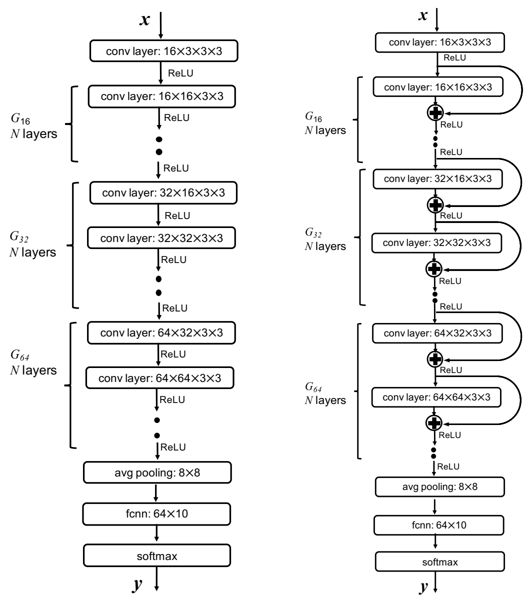
$$\sigma_p = \sqrt{\frac{2}{S}} \quad \text{where} \quad S = CUV \ . \tag{3}$$

Initialization of the parameters of a residual network is similar, except that σ_p is replaced by

$$\sigma_r = \frac{1}{S} \ , \tag{4}$$

which is typically much smaller than σ_p. This prescription, called Hartz-Ma Weight Initialization (HMWI) method (Hardt & Ma, 2017), reflects the observation that residual weights are generally smaller than plain weights. Table 1 shows training loss $\mathcal{L}$ and test error $\mathcal{E}$ after 200 epochs of training, when the loss stabilizes.

We observe that 1) ResNets have smaller $\mathcal{L}$ and $\mathcal{E}$ than the PlnNets of equal depth, and the performance discrepancy between ResNets and PlnNets increases with the depth of the network. 2) PlnNets exhibit an obvious degradation problem, as $\mathcal{L}$ increases 6.85 (0.4950/0.07227) times from depth 20 to depth 44. This problem is much smaller for ResNets, as $\mathcal{L}$ increases 1.73 (0.07397/0.04287) times from depths 20 to 44. 3) The test error $\mathcal{E}$ for PlnNets increases



(a) (b)

Figure 4. The architecture of (a) plain network and (b) the corresponding residual network. There are three groups of N layers each: G_{16}, G_{32}, and G_{64}. The entire network has $L = 3N + 2$ layers. The number of weights of both networks of same depth equals. The inputs from CIFAR-10 data set are $32 \times 32 \times 3$ RGB images and the output is a vector of size 10×1 that represents soft-max scores for the 10 classes.

significantly with depth, but remains roughly constant for ResNets.

In summary, our simplified ResNets outperform our simplified PlnNets in terms of both training loss and test error, and ResNets suffer much less from the degradation problem. Thus, *our network simplification preserves the phenomena we wish to study.*

B. Equivalent Training of Equivalent Nets

PlnNets and ResNets can only be trained equivalently in ideal case. Empirically, we observed some small discrepancies as a result of rounding errors. Specifically, Table 2 shows the results of the same experiment performed for Table 1, except that the two networks are initialized to be equivalent. We used double precision in the experiments, and performed all calculations on the same CPU to avoid possible hardware differences between different GPUs or CPUs.

We explain these discrepancies based on the following observations. After one iteration of training, some outputs of the second convolutional layer differ by about 10^{-16} between the two networks, as a result of rounding errors (the relative accuracy of double-precision floating-point arithmetic is

	PlnNet			ResNet		
depth	$\mathcal{L}$	$\mathcal{C}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{C}$	$\mathcal{E}$
20	0.07227	0.07304	0.1263	0.04287	0.04338	0.1131
32	0.1225	0.1234	0.1349	0.04847	0.04903	0.1143
44	0.4950	0.4962	0.2213	0.07397	0.07459	0.1182

Table 1. Training loss $\mathcal{L}$, training cost $\mathcal{C}$ and test error $\mathcal{E}$ of PlnNets and ResNets for increasing depth and after training for 200 epochs on the CIFAR-10 dataset. The initial learning rate was 10^{-3} for the depth-44 plain network, and 10^{-2} for the other 5 networks. The reason for the exception is that the training loss for the depth-44 plain network does not decrease with larger learning rates. The learning rate was divided by 10 after epochs 120 and 160 for all networks. The parameter λ of weight decay is 10^{-4}. Loss $\mathcal{L}$ is generally over 80 times more than the weight decay term. Figures are reported with four significant decimal digits.

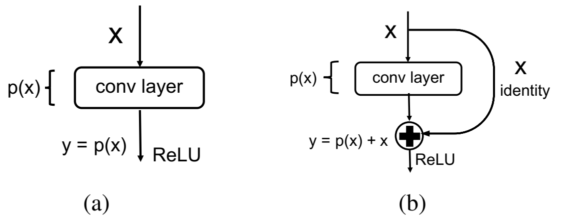


Figure 5. Simplified (a) plain block and (b) corresponding residual block.

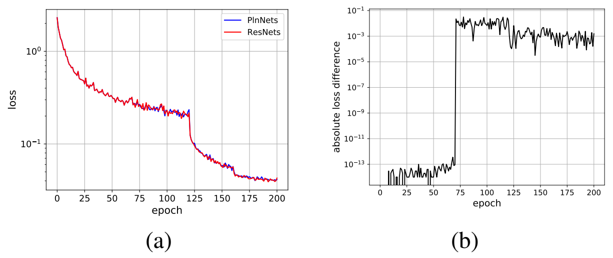


Figure 6. Left: Semi-logarithmic plots of $\mathcal{L}$ for plain (blue) and residual (red) networks of depths 20, initialized with equivalent parameter values. Right: Semi-logarithmic plots of the absolute differences.

10^{-17}) and of the different orders of magnitude of the ID weights between plain and residual network (ID weights are of the order of 10^{-3} for residual networks, and the corresponding plain ID weights are close to 1). As numerical errors propagate through ReLU functions, slightly negative numbers are truncated to zero, while slightly positive ones are left unchanged. These small deviations sometimes compound through a sort of ratcheting effect, in which they accumulate in one of the two networks but not the other. For instance, we observed a value -2.4980×10^{-13} being truncated to zero by a ReLU, while the corresponding value 2.6903×10^{-10} in the other network remained unchanged as it passed through the ReLU, and repeated operations increased the divergence. Figure 6 supports this explanation by showing a very slow divergence over training time.

C. Evolution of Weight Distribution

The left column in Figure 7 shows two views of the distributions $\chi(z, \mathbf{p}_0)$ (blue) and $\chi(z, \mathbf{t}_0)$ (red) of PlnNets and ResNets of depth 32 just after initialization. The plot in the top row shows these two distributions in full with the y-axis clipped to 800. The plot in the bottom row only shows the histogram for the ID entries in the transferred parameter vector $\mathbf{t}_0$.

The distribution of the "plain" weights $\mathbf{p}_0$ is clearly visible in blue in the overall view at the top. The distribution of the component with zero-mean Gaussian of "transferred" weights is also visible in red, but the variance is very small compared to that of the "plain" weights. In addition, the distribution of the component with one-mean Gaussian for the transferred weights $\mathbf{t}_0$ (red) is also visible in the overall view, since the y-axis is clipped to a small number.

After 200 training epochs (right column in Figure 7), the two distributions have not changed much, relative to 1: The blue plots are qualitatively similar to what they were before, and the narrow red peaks, both at 0 and 1, have spread out. However, the second row of the Figure, which displays histograms for the ID entries only, shows that the mode for these entries is still high, around 0.9, in $\mathbf{t}_{200}$, while very few weights in $\mathbf{p}$ has magnitude above 0.7 (first row), either before (left: 1 entry bigger than 0.7) or after training (right: 26 entries bigger than 0.7).

Table 3 shows some additional statistics for the weight distributions, including the total weight mass, defined as the L_1 norm of a parameter vector, for (i) all the parameters; (ii) the parameters with magnitude greater than 0.25, and (iii) all ID entries. This table shows in particular that even after training, the total mass of the ID entries in the residual-equivalent parameter vector $\mathbf{t}_{200}$ still accounts for more than 76 percent ($1000/1300$) of the mass of all the entries greater than 0.25. In contrast, the overall mass of the ID entries in $\mathbf{p}_0$, initially very small (67), barely changes in $\mathbf{p}_{200}$, where it increases to 72, a mere 10.29 percent (72/700) of the mass of all the entries greater than 0.25.

In summary, for this experiment, training modifies the pa-

	PlnNet			ResNet		
depth	$\mathcal{L}$	$\mathcal{C}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{C}$	$\mathcal{E}$
20	0.04132	0.04183	0.1121	0.04287	0.04338	0.1131
32	0.04880	0.04936	0.1085	0.04847	0.04903	0.1143
44	0.07345	0.07407	0.1159	0.07397	0.07459	0.1182

Table 2. Training loss $\mathcal{L}$, training cost $\mathcal{C}$ and test error $\mathcal{E}$ of equivalently initialized ResNets and PlnNets of depth 20, 32, and 44 after training for 200 epochs on the CIFAR-10 dataset. The initial learning rate was 10^{-2}, and the rate was divided by 10 after epoch 120 and 160. The parameter λ of weight decay is 10^{-4}. Loss $\mathcal{L}$ is generally over 50 times more than weight decay term. All the experiments are conducted on the same CPU to guarantee the two networks have exactly the same training process. Figures are reported with four significant decimal digits.

		$\mathbf{p}_0$	$\mathbf{p}_{200}$	$\mathbf{t}_0$	$\mathbf{t}_{200}$
Mean		0.000 062	−0.0061	0.0023	−0.0056
Standard Deviation		0.070	0.071	0.048	0.071
Mass	All Weights	25 000	25 000	1900	20 000
	$\lvert$Weights$\rvert \geq 0.25$	420	700	1100	1300
	Weights of ID Entries	67	72	1100	1000

Table 3. Statistics of the parameter vectors $\mathbf{p}$ and $\mathbf{t}$ before and after training for a network of depth 32. Figures are reported with two significant decimal digits.

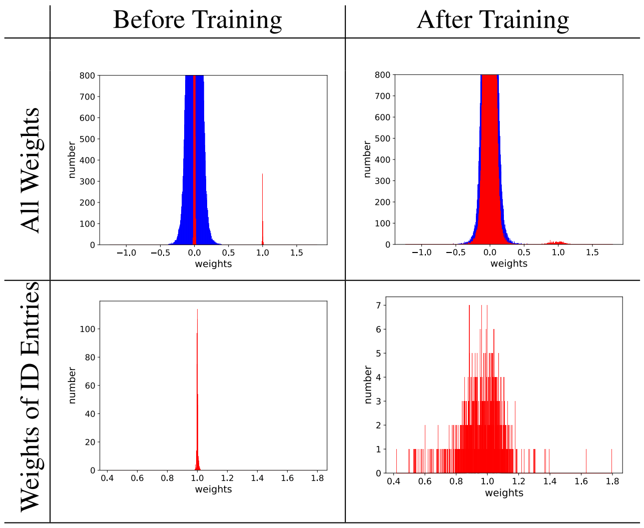


Figure 7. Distribution of all the weights (top row) for depth-32 plain network $p_{32}(\mathbf{x}, \mathbf{p}_e)$ (blue) and residual-network-equivalent plain network $p_{32}(\mathbf{x}, \mathbf{t}_e)$ (red), and the weights of the ID entries (bottom row) for $p_{32}(\mathbf{x}, \mathbf{t}_e)$ before training ($e = 0$, left column) and after 200 epochs ($e = 200$, right column).

rameters of a network by a small extent, when compared to the differences between the initialization vectors $\mathbf{p}_0$ and $\mathbf{t}_0$.