Exploiting Invariance in Training Deep Neural Networks

Chengxi Ye, Xiong Zhou, Tristan McKinney, Yanfeng Liu, Qinggang Zhou, Fedor Zhdanov

Amazon Web Services
chengxye,xiongzho,tristamc,liuyanfe,qingganz@amazon.com, fedor.zhdanov@rhul.ac.uk

Abstract

Inspired by two basic mechanisms in animal visual systems, we introduce a feature transform technique that imposes invariance properties in the training of deep neural networks. The resulting algorithm requires less parameter tuning, trains well with an initial learning rate 1.0, and easily generalizes to different tasks. We enforce scale invariance with local statistics in the data to align similar samples at diverse scales. To accelerate convergence, we enforce a $GL(n)$-invariance property with global statistics extracted from a batch such that the gradient descent solution should remain invariant under basis change. Profiling analysis shows our proposed modifications takes $\sim$ 5% of the computations of the underlying convolution layer. Tested on convolutional networks and transformer networks, our proposed technique requires fewer iterations to train, surpasses all baselines by a large margin, seamlessly works on both small and large batch size training, and applies to different computer vision and language tasks.

Introduction

The pupillary light reflex constricts the pupil in bright light and dilates the pupil in dim light (Bear, Connors, and Paradiso 2020). This mechanism controls the amount of light passing into the eye, allowing common features to be extracted from signals of different scales. In addition, retinal receptive fields use center-surround structures to filter and sharpen images (Hubel and Wiesel 1962) (Supp. Fig. 2 a, b) , removing a bell-shaped autocorrelation present in real-world visual signals. Intriguingly, all other receptive field configurations discovered in Hubel and Wiesel's seminal research (Supp. Fig. 2 c, d, e) have found artificial analogs in the first layer filters learned by modern convolutional neural networks (Zeiler and Fergus 2014) (Fig. 5 a,b).

While convolutional networks continue to push the envelope in computer vision tasks, state-of-the-art training recipes are still limited by scope and scale. Specifically, when moving from image classification to object detection, different normalization techniques need to be used. Most algorithms perform well at a specific scale, and there is usually a significant drop in accuracy when the training batch size is too large or too small.

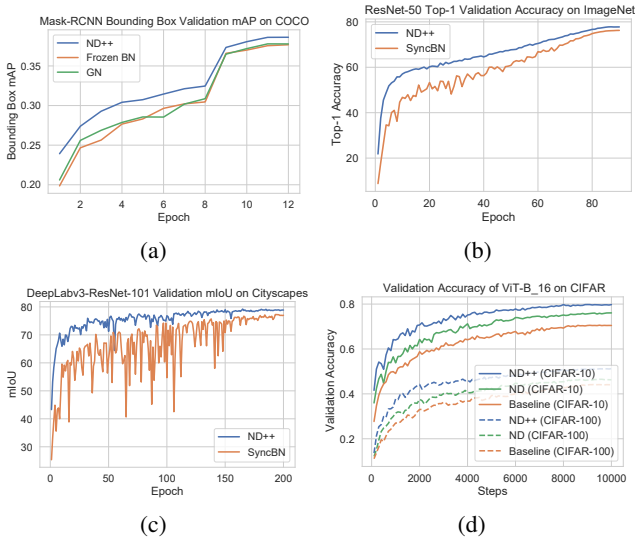


Figure 1: (a) Validation AP curves of a Mask R-CNN network when trained with network deconvolution++ (ND++), frozen batch normalization and group normalization. (b) Top-1 validation accuracy of ResNet-50 when trained on ImageNet with ND++ and SyncBN using batch size of 2048, the training finishes better and faster than using batch size 256 using the same hardware. (c) mIoU curves of DeepLabv3 with a ResNet-101 backbone trained on the Cityscapes dataset using ND++ and SyncBN. ND++ consistently outperforms baselines and produces more stable mIoU curves. (d) ND++ significantly improves upon ND when training ViT-B_16 on the CIFAR datasets.

In this paper, we conduct a study of a one-layer linear network to understand the origin of these limitations. Drawing inspiration from this study, the structure of animal visual systems mentioned above, and recent related work (Ye et al. 2020), we derive two invariance properties that enhance the training of deep neural networks. We implement cross-GPU synchronization to aggregate the computation required to enforce the invariance, surpassing the widely-used synchronized batch normalization (Peng et al. 2017) method significantly. This implementation supports both small batch and

large batch training without algorithm change. By enforcing the invariance properties at every layer of the network, we accelerate training convergence and surpass baseline accuracy by a large margin on the ImageNet (Deng et al. 2009), MS COCO (Lin et al. 2014), and Cityscapes (Cordts et al. 2016) datasets for image classification, object detection, and semantic segmentation, respectively (Fig. 1). In our supplementary materials, we also show promising results for training transformers on multiple vision and language tasks.

Our main findings are the following:

- We propose a drop-in modification *before* the linear layers in a network to explicitly enforce these two invariance properties, significantly reducing training iterations and surpassing baseline accuracy by a large margin.
- Training with this modification is robust to different optimizer configurations; using the theoretically optimal learning rate for the linear case of 1.0 can successfully train a wide range of deep networks.
- Our implementation takes $\sim 5\%$ (Supp. Table 1) of the cost of the underlying convolution layer and incorporates a cross-GPU synchronization which surpasses synchronized batch normalization by a large margin in both small and large batch regime.
- The benefits of the proposed modification generalize to emerging architectures in vision and language.

Background

Backward Correction Methods

The complicated loss landscapes (Li et al. 2017) of neural networks create numerous challenges for training. Deep neural networks are generally over-parameterized. The presence of strong correlation between features induces areas of pathological curvature in the landscape and inhibits effective training. Small gradients are common in these pathological regions, and the problem is exacerbated by small linear layers and common activation functions. These issues were traditionally addressed by correcting the gradients (Nocedal and Wright 2006; Martens et al. 2010; Ye et al. 2017), that is, by modifying the backward pass. The most popular methods normalize the gradient scale to avoid the vanishing gradient problem and smooth the direction by using previous gradients as a momentum term (Kingma and Ba 2014). More advanced methods, such as Newton's method, use approximate curvature information to modify the gradient direction (Fig. 2). However, high computational costs limit these algorithms to small-scale problems (Martens and Grosse 2015; Desjardins et al. 2015). They have not been widely shown to surpass first-order gradient descent methods.

Forward Correction Methods

Forward transforms provide an alternative approach to address the challenges in training deep neural networks (Huang et al. 2020). Batch normalization (Ioffe and Szegedy 2015) is a common and powerful example that standardizes the distribution of features in each dimension. This stretching with a diagonal matrix works perfectly only for uncorrelated, axis-aligned features, suggesting that the optimality of standardization *depends on the choice of basis*. Given the inherent correlation in real-world data, the feature covariance matrix is generally ill-conditioned and not diagonal. As a result, gradient descent training usually takes unnecessary steps towards the solution (Fig. 2(b)). Recently, more accurate transforms have been proposed to utilize the covariance matrix and remove the pairwise correlation between features (Huang et al. 2018; Ye et al. 2018; Huang et al. 2019; Pan et al. 2019; Ye et al. 2020). If the features are transformed to be axis-aligned and have unit-variance, the loss landscape is more isotropic. Gradient descent converges more quickly and accurately (Fig. 2(c)).

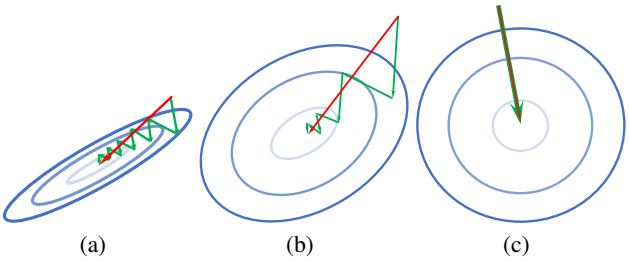


Figure 2: A toy picture of loss landscapes and gradient steps. The red arrows correspond to the gradients found through backward correction or Newton's method and the green arrows correspond to the gradients used in gradient descent iterations. (a) Highly correlated, non-standardized data. (b) Correlated but standardized data. (c) With standardized and uncorrelated data, vanilla gradient descent coincides with Newton's method (Proposition 1).

Motivations

We start our investigation with a toy example of a one-layer network (Eq. 1). Rigorous analysis of this simple setup reveals a surprising depth of insight into the fundamental training problem. The analysis explains the utility of various modern techniques to facilitate training and leads us to a more advantageous design.

Assume we are given a linear regression problem with a mean squared error loss (Eq. 1). $\hat{y}$ is the continuous or discrete response data to be regressed. This formulation can be used for prediction or classification. In the typical setting, the output $y = Xw$ is given by multiplying the inputs X with an unknown weight vector w for which we are solving. Let N be the number of samples, d the feature dimension, and X either the $N \times d$ or augmented $N \times (d+1)$ data matrix $(X|\mathbf{1})$, if we include a bias.

$$Loss_{MSE} = \frac{1}{2}E(|y - \hat{y}|^2) = \frac{1}{2N}\|Xw - \hat{y}\|^2. \quad (1)$$

Local Statistics vs Global Statistics

Given a $N \times d$ data matrix X, we refer to the column statistics as the *global statistics* and the row statistics as *local statistics*. Mini-batch statistics represent an approximation

to the global statistics of the whole dataset. Batch normalization (Ioffe and Szegedy 2015) standardizes the d column vectors. This can be visualized as a coordinate transform that stretches the data along each axis based on global statistics (Fig. 2 (a) to (b)). Training then solves for *a new set of weights* w_{BN} in this transformed space.

On the other hand, sample-based normalization (Ba, Kiros, and Hinton 2016; Wu and He 2018; Singh and Krishnan 2020) stretches each sample by removing the scale and bias in each row of X according to the local statistics. Viewed from the original space, training corresponds to finding N *sets of sample-variant weights* $\{w_i\}$ that both stretch the samples and fit the model.

$GL(n)$-invariance

One may ask whether the solution of Eq. 1 found using a given algorithm changes under a change of basis. That is, if we use any invertible linear transform, i.e., a member of the general linear group $GL(n)$ (Artin 2011), to transform the features, will we simply reach an equivalent solution in a different coordinate system? If the solution is invariant under the operation of $GL(n)$, we will call the training algorithm $GL(n)$-invariant.

Assuming X^tX is invertible for our toy problem, the unique solution can be found by setting the gradient to 0, $\frac{\partial Loss}{\partial w} = \frac{1}{N}X^t(Xw - \hat{y}) = 0$,

$$w = (X^tX)^{-1}X^t\hat{y}. \tag{2}$$

Let us consider the impact of different correction methods on the ability of gradient descent-based algorithms to find this solution.

Case 1 (backward correction/Newton's method): For simplicity, suppose we start from $w_0 = 0$, then $\frac{\partial Loss}{\partial w}|_{w_0} = -\frac{1}{N}X^t\hat{y}$. Letting $H = \frac{1}{N}X^tX = \nabla^2_w Loss$, we see from Eq. 2 that $w = -H^{-1}\frac{\partial Loss}{\partial w}$. This derivation shows us how the gradients can be manipulated in the backward fashion to accelerate convergence (Martens and Grosse 2015; Desjardins et al. 2015; Ye et al. 2017): (1) approximate the curvature with H and apply an inverse correction to *decorrelate the gradient*, $H^{-1}\frac{\partial Loss}{\partial w}$, then (2) take a descent step using a learning rate of 1.0: $w = w_0 - 1.0 \cdot H^{-1}\frac{\partial Loss}{\partial w}$. $GL(n)$ invariance can be achieved because the optimal solution can be found in one step by following the corrected negative gradient in *any basis* (found as the red arrows in Fig. 2).

Case 2 (forward correction): Instead of the Newton's method, we adopt a forward correction point of view. Simple forward corrections include standardization using global statistics, which results in standardized columns, while using local statistics results in standardized rows. In terms of convergence rate, using global statistics is slightly superior, as after correction $\frac{1}{N}H = \frac{1}{N}X^tX$ is guaranteed to have unit diagonal. Therefore, the loss landscape has better statistical properties and convergence is accelerated.[1] However, standardization with either local or global statistics does not remove correlations between features. On an elongated energy landscape, gradient descent algorithms generally do not converge in one step and the optimal solution cannot be found under an arbitrary basis change. The following proposition demonstrates that forward correction can achieve the power of Newton's method if we (1) *decorrelate the feature columns with global statistics* and (2) use a learning rate 1.0.

Proposition 1. *The optimal solution can be found in one step with the forward correction if and only if the feature columns are standardized and uncorrelated.*

Proof. With these features we have $\frac{1}{N}X^tX = I$, and the optimal explicit solution (Eq. 2) simplifies to $w = \frac{1}{N}X^t\hat{y}$. After one iteration of gradient descent, $w_{new} = w_{old} - \eta\frac{1}{N}(X^tXw_{old} - X^t\hat{y})$. By substituting $\frac{1}{N}X^tX = I$, we find that $\eta = 1.0$ is optimal and yields convergence in a single iteration. □

Scale Invariance

In our toy problem, the output scales linearly with the input, $(aX)w = a(Xw)$. In fact, this property generalizes to common networks with linear layers and ReLU-like activation functions. If we consider a task such as bounding box prediction, this suggests that we can scale the box size by simply changing the input brightness, which should clearly be avoided. Through the pupillary light reflex, animal visual systems introduce a scale invariance property so that differently scaled inputs generate similarly scaled features. Although transforming using global statistics can guarantee $GL(n)$-invariance as discussed above, it does not result in scale invariance. Since scale invariance is a concern in some tasks, we utilize both global and local statistics (where samples are transformed individually) in our algorithm design, and we defer the explanation of details to the next section.

Optimizing the Formulation

Influenced by the batch normalization, most normalization algorithms (Ba, Kiros, and Hinton 2016; Wu and He 2018; Singh and Krishnan 2020; Huang et al. 2018, 2019; Pan et al. 2019) adopt a post-normalization design, normalizing the output $y = Xw$ of the linear transform rather than normalizing the input X. Since post-normalization restricts the representation power of y, two extra parameters are introduced to remove this limitation: $Loss_{MSE} = \frac{1}{2N}||(Xw - \mu)\Sigma^{-1}\gamma + \beta - \hat{y}||^2$. As a result, the toy problem contains more than necessary parameters (w, γ, β instead of w) to be optimized. However, the optimal convergence property that we find on the toy problem no longer holds with the redundancy. This motivates us to optimize our design to remove the redundancy while maintaining full representation power.

Enforcing Invariance in Training Neural Networks

The motivations naturally lead us to a new formulation by applying the principles derived from the toy problem. We start from identifying the data matrices X, in three common linear layers. Then we remove the correlation between the

[1] Normalization with local statistics works on XX^t, which is less related to the convergence.

features to accelerate the convergence and insert scale invariance to pre-align features at different scales.

Case 1. For a fully-connected layer, the data matrix X of N samples is constructed straightforwardly by stacking N rows of d or $d+1$ dimensional feature vectors. The training convergence is suboptimal with gradient descent approaches if the columns are correlated (Fig. 2 (a,b)).

Case 2. For a convolution layer, the computation can be expressed in three ways in the spatial domain, expressed as $y = x * w = Wx = Xw_{flipped}$. We call them the (a) direct convolution, (b) convolution matrix and (c) data matrix formulation respectively. For (b) we expand the kernel of size k into a $(n-k+1) \times n$ convolution matrix W. For (c) we unroll (known as $im2col$) the overlapping windows of x into a $(n-k+1) \times k$ data matrix X. We illustrate the matrices of a $1d$ convolution example of (b, c) using $k = 3$ (in the 'valid' mode according to Matlab terminology):

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_{n-k} \\ y_{n-k+1} \end{pmatrix} = \begin{pmatrix} w_3 & w_2 & w_1 & & & \\ & w_3 & w_2 & w_1 & & \\ & & \ddots & & & \\ & & & w_3 & w_2 & w_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{pmatrix}$$

$$= \begin{pmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \\ \dots & & \\ x_{n-3} & x_{n-2} & x_{n-1} \\ x_{n-2} & x_{n-1} & x_n \end{pmatrix} \begin{pmatrix} w_3 \\ w_2 \\ w_1 \end{pmatrix}$$

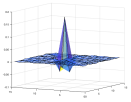
By closely looking at the different columns in the convolution data matrix X (in case 2 (c)), one can find a rarely discussed problem in the training of networks. Since real-world data exhibits strong autocorrelation, and the neighboring columns of the data matrix correspond to shifting the signal by one pixel, the feature dimensions in the convolution/-correlation data matrices are *heavily correlated*. Therefore gradient descent training cannot converge efficiently with existing standardization techniques (Fig. 2 (b)) or just by decorrelating the feature channels in a layer.

Case 3. The correlation operation (also called transposed convolution, or misnomered as deconvolution) is the adjoint operation of the convolution, and involves padding the data by $k-1$ on each side (corresponding to the 'full' mode in Matlab terminology) and using the unflipped kernel. The data matrix of correlation only differs from that in the convolution by the zero padding and is omitted here. The correlation operation shares the same convergence problem with the convolution case.

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 & & & \\ & w_1 & w_2 & w_3 & & \\ & & \ddots & & & \\ & & & w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ x_1 \\ x_2 \\ \dots \\ x_{n-k+1} \\ 0 \\ 0 \end{pmatrix}$$

Connection with Frequency Domain Normalization

For convolution and correlation, removing such pixel-wise autocorrelation has a long history and can be achieved through a process called whitening deconvolution (Gonzalez and Woods 2006). This correlation removal process can be applied to every layer of a convolution neural network and we call this procedure network deconvolution. Usually this is achieved in the frequency domain through spectral whitening, i.e. a normalization in the frequency domain: $\frac{\mathcal{F}(x)}{|\mathcal{F}(x)|} = \frac{\mathcal{F}(x)}{\sqrt{\mathcal{F}^H(x) \cdot \mathcal{F}(x)}}$. This elegant normalization shows a profound insight: the optimal standardization for the convolution operation should in fact be carried out in the frequency domain. Note that frequency multiplication corresponds to a spatial domain convolution, and the frequency division corresponds to a spatial domain *deconvolution*. Moreover, the deconvolution kernel $\mathcal{F}^{-1}(\frac{1}{|\mathcal{F}(x)|})$ has been found to resemble the center-surround structures in animal visual systems (Supp. Fig. 2(a,b)) (Hubel and Wiesel 1962; Hyvrinen, Hurri, and Hoyer 2009; Ye et al. 2020). Four lines of Matlab code are provided here for a quick verification (Fig. 3).



```
I=double(imread('cameraman.tif'))/255;
f=ifft2(1./(abs(fft2(I))+eps));
f=circshift(f,[7,7]);
figure;surf(f(1:15,1:15),'FaceAlpha',0.5)
```

(a) (b)

Figure 3: Visualizing a frequency domain deconvolution filter in the spatial domain leads to a structure that has positive center values and negative surround values, resembling an on-center cell in animal visual systems (Hubel and Wiesel 1962) (Supp. Fig. 2(a)).

Spatial Domain Implementation

Algorithm 1: Proposed Computation for a Convolution Layer

Input: Feature tensor $X_{N \times C \times H \times W}$
Scale features along first dimension $X_{scaled} = SX$
Convert feature channels to data matrices along second dimension
for $i \in \{1, ..., C\}$ **do**
  $X_i = im2col(X_{scaled}[:, i, :, :])$
end for
$X = [X_1, ..., X_C]$ %Horizontally Concatenate
$Cov = \frac{1}{N} X^t X$ %Cross GPU Sync
$D \approx (Cov + \epsilon \cdot I)^{-\frac{1}{2}}$ %via Inverse Newton Iterations
Output: $Y = Conv(SX, DW)$

For a practical implementation for convolutional networks, it is better to carry out the deconvolution computation in the spatial domain using the data matrix X with the consideration of kernel size, stride, and boundary condition that are found in practice. All three linear transforms share the following correlation removal process: we calculate the

covariance matrix $Cov = \frac{1}{N}X^tX$, find its unique principal inverse square root $D = Cov^{-\frac{1}{2}}$, and then use this matrix to transform the data matrix: $X_0 = X \cdot D$. Note that $\frac{1}{N}X_0^tX_0$=$Cov^{-\frac{1}{2}} \cdot Cov \cdot Cov^{-\frac{1}{2}} = I$. As the transformed features are uncorrelated and standardized, faster convergence is achieved in this corrected space (Fig. 2(c)). We track the correction D during training and use the running average during testing.

For a convolution/correlation layer with C channels, we construct the data matrix for each channel then horizontally concatenate the C data matrices into a wider matrix (with Ck/Ck^2 columns for $1d$/$2d$). Deconvolution with this wider matrix removes the correlation of k/k^2 *nearby pixels* and C *feature channels* at once (see Algorithm 1).

Our experiments show that the above formulation works well for simple classification tasks that do not require scale invariance. However, it may not work well for some other tasks such as object detection and instance segmentation where scale invariance plays an important role. To generalize to these tasks, we incorporate local statistics to remove a scale factor in each individual sample. This simple strategy is inspired by the biological observation that *animal visual systems also use two sets of statistics for visual analysis* (Hubel and Wiesel 1962; Bear, Connors, and Paradiso 2020). The retinal light reflex (Bear, Connors, and Paradiso 2020) suggests local statistics is used to adjust the scale of the signal. For a fully-connected layer, local statistics are calculated from each row of the data matrix X. For a convolution/correlation layer, local statistics can be calculated for one or more rows of the data matrix or even the full feature tensor at each layer. We have found empirically that the last option works well and does not introduce an excessive computational cost. At each layer we standardize each sample by its sample statistics, interchangeably with $LayerNorm$ (Ba, Kiros, and Hinton 2016), or with l_1 norm $E(|x|)$ if we just consider the scale. Combined with the linear transform weights, our proposed feature transform is:

$$y = S \cdot X \cdot D \cdot w. \tag{3}$$

Here S is the standardization operating on the *rows* of X, a diagonal matrix ($\frac{1}{\sigma_i}$) if we only consider the scaling or augmented with an extra column ($-\frac{\mu_i}{\sigma_i}$) if we also consider the bias, and D is the decorrelation operating on the *columns* of $S \cdot X$. The weights are trained in the transformed space based on the uncorrelated features $S \cdot X \cdot D$.

Our formulation transforms both the rows and columns of X without introducing redundancy. Normalization using local statistics enforces scale invariance, aligning features of different scales. Normalization with global statistics enforces $GL(n)$-invariance, leading to faster convergence.

Implementation Details

Cross-GPU Synchronization

To leverage the latest advancements in hardware development, we implement cross-GPU synchronization to improve the quality of the estimates (see (Pan et al. 2019) for another independent implementation). At each layer, the covariance matrix is computed on each GPU and synchronized across all the GPUs. On an 8-GPU machine, the synchronization cost is negligible. This implementation allows us to collect reliable statistics throughout the training for all practical batch sizes. In our development, we have consistently achieved satisfactory results using per-GPU batch sizes ranging from from 2 to 1024.

Acceleration Techniques

We use several techniques to simplify computation and reduce the complexity of the proposed algorithm to a small fraction ($\sim 5\%$) of the cost of a convolution layer (Supp. Table 1). The memory accessing time of extracting the data matrix from a convolution/correlation layer in existing packages takes significantly more time than the underlying convolution operation. We adopt a $3\times$-subsampling for ImageNet scaled images and $5\times$-subsampling for MS COCO scaled images. The data and covariance matrices are computed using tensor patches sampled at the strided locations, reducing the construction cost by $9\times$ and $25\times$ respectively. Since the involved number of pixels is usually more than enough, this strategy maintains accuracy while keeping the computational footprint low. When the covariance matrix becomes too large, we divide the columns into blocks and decorrelate between the blocks (Ye et al. 2017; Huang et al. 2018). In our experiments, we set the number of blocks $B = 256$ for fully-connected layers and $B = 64 \times k^2$ for convolution/correlation layers.

There are multiple ways (Huang et al. 2018; Ye et al. 2018; Huang et al. 2019; Ye et al. 2020) to calculate the principal inverse square root of a matrix but algorithms that are both efficient and stable are scarce. Since explicit eigenvalue decomposition is slow, Newton-Schultz iteration becomes a faster alternative. However, the vanilla Newton-Schultz method is unstable under finite arithmetic (Ye et al. 2020)(Fig. 7), so we adopt a coupled inverse Newton iteration method that is numerically stable and works well under finite precision arithmetic (Guo and Higham 2006). Starting with $X_0 = I$, $M_0 = Cov$, the coupled inverse Newton iteration calculates $X_{k+1} = X_k\frac{(3I-M_k)}{2}$, $M_{k+1} = (\frac{3I-M_k}{2})^2M_k$ and produces $X_k \to Cov^{-\frac{1}{2}}$. Empirically, using 5 iterations yields good results. We also add a small diagonal matrix ϵI to the covariance matrix to avoid rank-deficiency.

Note that for a large data matrix X, decorrelating columns of X can require excessive computation. Previous applications are usually restricted to a small number of layers (Huang et al. 2018, 2019; Pan et al. 2019), which adds to the network design complexity. We adopt a universal design and avoid excessive computation by reordering the computation by transforming the model weights instead, $y = (S \cdot X) \cdot (D \cdot w)$ (see Algorithm 1).

Experiments

We refer to our improved implementation as Network Deconvolution++ (ND++). ND++ introduces scale invariance, generalizes the $GL(n)$-invariance to all three common linear transform layers (convolution/correlation/linear) in modern

Network	SyncBN	ND++
VGG-11	71.11	**72.24**
ResNet-50	76.25	**77.95**
ResNet-101	77.37	**79.40**
DenseNet-121	74.65	**76.11**

Table 1: Top-1 Validation Accuracy on the ImageNet Dataset. ND++ also surpasses the top-1 accuracy rates of the deeper networks in the model zoo: VGG-13: 71.55%,ResNet-101: 77.37%,ResNet-152: 78.32%,DenseNet-169: 76.00%.

	MB	MLPerf	Detectron2
ND++	**37.36**	**37.37**	**39.01**
BN	36.78	36.35	37.9
GN	36.04	35.9	38.54

Table 2: Bounding box AP of three Faster R-CNN implementations on the COCO 2017 dataset.

	MB		MLPerf		Detectron2	
	AP^{bbox}	AP^{mask}	AP^{bbox}	AP^{mask}	AP^{bbox}	AP^{mask}
ND++	**38.62**	**35.65**	**38.36**	**35.07**	**39.91**	**36.87**
BN	37.67	34.28	37.14	33.97	38.6	35.2
GN	37.82	34.75	36.32	33.52	38.96	35.93

Table 3: Bounding box and mask AP of three Mask R-CNN implementations on the COCO 2017 dataset.

	LR	AP^{bbox}	AP^{mask}
ND++	0.02	37.81	34.87
GN	0.02	37.76	34.8
ND++	0.1	**38.86**	**35.56**
GN	0.1	35.21	33.66

Table 4: Mask R-CNN trained from scratch on the COCO 2017 dataset.

architectures, and uses cross-GPU synchronization to allow reliable training at different scales. In the following experiments, all linear transform layers are enhanced with ND++. In fine-tuning experiments, the pretrained backbone network is replaced with a backbone pretrained with ND++. For a fair comparison, we have also experimented with training from scratch to verify the gain is not only from the improved backbone. Standard stochastic gradient descent is used for all experiments. We continue to use learning rate decay in the face of inherent non-linearity and mini-batch training.

Image Classification

We demonstrate improved training on three popular CNN architectures (VGG, ResNet, DenseNet) at three scales (10/50/100 layers). With ND++, we have seamlessly increased the training to batch size 2048, eight times larger than the model zoo default setting of 256. We train in one-eighth the number of iterations but produce superior models (Table 1). All three network architectures, VGG-11/ResNet-50/ResNet-101/DenseNet-121, surpass *their deeper counterparts* in the model zoo after standard 90-epoch training with cosine learning rate decay. Using the official PyTorch recipe, it takes less than one day to train on a machine with 8 Nvidia A100 GPUs. On the popular ResNet-50/101 network, the $\mathbf{77.95\%/79.40\%}$ top-1 accuracy we reach is among the highest numbers reported when training for 90 epochs (Fig. 1(c)).

Object Detection

We test ND++ on Faster R-CNN and Mask R-CNN, milestone object detectors from three major benchmarks, *maskrcnn-benchmark*(MB), *MLPerf*, and *Detectron2*. We use ResNet-50 with FPN as backbones and two fully-connected layers in the box heads. In Mask R-CNN, the mask head contains convolution and correlation layers. ND++ is used to enhance all layers. We report our numbers and evaluation curves on the COCO 2017 dataset (Tables 2, 3, Fig. 1(a,b)). Results with ResNet-101 can be found in Supp. Sec. , Supp. Tab. 2.

On all three benchmarks and with both Faster R-CNN (Table 2) and Mask R-CNN (Table 3), ND++ consistently outperforms baselines with frozen batch normalization and group normalization (Wu and He 2018) in terms of Average Precision(AP) (Fig. 1(a,b)). When training from scratch, ND++ benefits from increase the learning rate to 0.1 while group normalization ends up with worse results (Table 4, Supp. Fig. 4).

Though researchers have encountered severe challenges in scaling up the training of Mask R-CNN (Peng et al. 2017; Wang et al. 2020), we have been able to seamlessly increase the batch size and achieve good results without ad hoc techniques. Here we take the MLPerf implementation, remove the warmup stage, and fix the learning rate to 0.1 with a momentum of 0.9. We notice that frozen/synchronized batch normalization explodes at all scales without warmup while ND++ produces superior results. ND++ significantly surpasses the MLPerf accuracy goal[2] within the standard 12-epoch fine-tuning with batch sizes up to 256, an order of magnitude larger than most existing baseline settings (Table 5, Supp. Fig. 5).

	LAMB	MegDet	ND++	
Batch	AP^{bbox}	AP^{bbox}	AP^{bbox}	AP^{mask}
128	36.7	37.7	**38.61**	35.54
256	36.7	37.7	**38.40**	35.26
512	36.5	-	**37.45**	34.5

Table 5: Large-scale training performance of Mask R-CNN using ND++. Our reported numbers are based on the NVIDIA implementation. ND++ significantly surpasses MegDet (Peng et al. 2017) and LAMB (Wang et al. 2020) that utilize SyncBN, warmup and optimizer change.

Network	SyncBN	ND++
DLv3-RN-50 (scratch, 200 ep.)	69.30	**72.69**
DLv3-RN-50 (finetune, 200 ep.)	75.70	**77.50**
DLv3-RN-101 (finetune, 200 ep.)	77.28	**79.23**
DLv3-RN-50 (finetune, 50 ep.)	-	76.47
DLv3-RN-50 (finetune, 500 ep.)	75.71	-

Table 6: Validation mIoU when fine-tuning and training from scratch on the Cityscapes dataset. DLv3 stands for DeepLabv3 and RN stands for ResNet.

ResNet-50	Mask R-CNN (MLPerf)		DeepLabv3-RN-50
top-1	AP bbox	AP mask	mIoU (Scratch)
77.66	38.26	35.41	71.64

Table 7: Experiments with an initial learning rate 1.0 and no momentum.

Semantic Segmentation

To demonstrate the usefulness of ND++ for semantic segmentation, we add ND++ layers to the DeepLabv3 architecture (Chen et al. 2017) with both ResNet-50 and ResNet-101 backbones and test the performance on the Cityscapes dataset (Cordts et al. 2016). We use a base resolution of 1024 for the images. We train for 200 epochs for all experiments unless stated otherwise. We perform fine-tuning experiments with a ResNet-50 backbone pretrained on ImageNet an initial learning rates of 0.01 and 0.1 and momentum 0.9 for both ND++ and the SyncBN baseline, and we report the results with learning rate 0.01 for SyncBN and 0.1 for ND++ in Table 6 since these produced the best results for each network configuration. Our ResNet-50 model with ND++ achieves an mIoU of 77.50 on Cityscapes, comparable to the ResNet-101 model from the original paper with 77.82 mIoU for single-scale evaluation (Chen et al. 2017).

ND++ substantially improves over the synchronized batch norm baseline, and the acceleration is especially apparent early in training. For example, it takes less than 50 epochs for a network equipped with ND++ to beat the performance of a network trained using SyncBN for 500 epochs (Supp. Fig. 7 and Table 6) if we fine-tune using a pretrained ResNet-50 backbone.

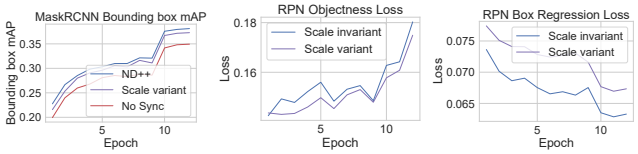
Experiments with Learning Rate 1.0

Existing deep neural network models are trained with stochastic gradient descent algorithm variants with momentum. Although the momentum term usually reduces noise and accelerates convergence, we notice that with ND++, many networks train well with a initial learning rate 1.0 and without the use of the momentum term, presumably thanks to the $GL(n)$-invariance property (Table 7).

Training a Vision Transformer Model

The techniques discussed in this paper naturally generalize to emerging architectures. To demonstrate, we compare a $10,000$-step training using ND++ and compare with ND and the standard training of a vision transformer (ViT-B_16) on the CIFAR-10/100 datasets. Mostly following the original setting, we resize images to 224, set the learning rate of SGD to 0.1 and the weight decay to 0.0001, and use a batch size of 512 when training from scratch[3]. Note that even though fine-tuning Transformers has shown promising results on various language and vision tasks, training transformers from scratch leaves large room for improvements. When training from scratch on the CIFAR-10 dataset, the baseline $10,000$-step (102 epochs) training yields an accuracy of only 70%. We drop in our modifications in every linear layer of the network and remove the original LayerNorms in the network. Interestingly, we observe $GL(n)$-invariance suffices and significantly improves the testing accuracy curve. Adding scale invariance further improve the results (Fig. 1(d)) to 80%. More training from scratch results for language tasks can be found in the supplementary materials. The wall-time of the baseline/ND/ND++ is 180/200/220 minutes, respectively, when run on 8 GPUs.

Ablation Studies



(a) (b) (c)

Figure 4: Ablation study with the Mask R-CNN architecture.

Cross-GPU synchronization and scale invariance produce larger improvements in the more advanced two-stage detector Mask R-CNN as indicated by ablation experiments on the MLPerf benchmark (Fig. 4 (a)). To further study the effect of enforcing scale invariance, we separately plot the objectness classification and regression losses for the region proposal network (RPN) (Fig. 4 (b,c)). Without scale invariance, the object classification loss remains lower throughout the training, but the bounding box regression loss is higher. The inaccurate proposals likely pull down the final results (Fig. 4 (a) the purple curve). Further studies can be found in the experiment details in the supplementary materials.

Discussion

The transform we propose has simple and intuitive geometric meanings. To achieve scale invariance, each sample is stretched individually using local statistics. The global geometry of the loss landscape, however, depends on the full collection of data. We therefore utilize the global distribution of data to find a unique optimal feature transform and achieve $GL(n)$-invariance. The unique gradient direction in this space corresponds to the optimal direction in the linear case and leads to significantly accelerated training of deep neural networks.

[2] 37.7 for bounding box AP and 33.9 for mask AP.

[3] The Vision transformer is accessed from: https://github.com/jeonsworld/ViT-pytorch

References

Artin, M. 2011. *Algebra*. Pearson Prentice Hall. ISBN 9780132413770.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. arXiv:1607.06450.

Bear, M.; Connors, B.; and Paradiso, M. A. 2020. *Neuroscience: Exploring the brain*. Jones & Bartlett Learning, LLC.

Chen, L.-C.; Papandreou, G.; Schroff, F.; and Adam, H. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; and Schiele, B. 2016. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3213–3223.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Desjardins, G.; Simonyan, K.; Pascanu, R.; and kavukcuoglu, k. 2015. Natural Neural Networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*, 2071–2079. Curran Associates, Inc.

Gonzalez, R. C.; and Woods, R. E. 2006. *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc. ISBN 013168728X.

Guo, C.-H.; and Higham, N. J. 2006. A Schur–Newton Method for the Matrix p th Root and its Inverse. *SIAM Journal on Matrix Analysis and Applications*, 28(3): 788–804.

Huang, L.; Qin, J.; Zhou, Y.; Zhu, F.; Liu, L.; and Shao, L. 2020. Normalization Techniques in Training DNNs: Methodology, Analysis and Application. *arXiv preprint arXiv:2009.12836*.

Huang, L.; Yang, D.; Lang, B.; and Deng, J. 2018. Decorrelated batch normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 791–800.

Huang, L.; Zhou, Y.; Zhu, F.; Liu, L.; and Shao, L. 2019. Iterative normalization: Beyond standardization towards efficient whitening. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4874–4883.

Hubel, D. H.; and Wiesel, T. N. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1): 106–154.

Hyvrinen, A.; Hurri, J.; and Hoyer, P. O. 2009. *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.* Springer Publishing Company, Incorporated, 1st edition. ISBN 1848824904, 9781848824904.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

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

Li, H.; Xu, Z.; Taylor, G.; and Goldstein, T. 2017. Visualizing the Loss Landscape of Neural Nets. *CoRR*, abs/1712.09913.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*, 740–755. Springer.

Martens, J.; and Grosse, R. B. 2015. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *CoRR*, abs/1503.05671.

Martens, J.; et al. 2010. Deep learning via hessian-free optimization. In *ICML*, volume 27, 735–742.

Nocedal, J.; and Wright, S. 2006. *Numerical optimization*. Springer Science & Business Media.

Pan, X.; Zhan, X.; Shi, J.; Tang, X.; and Luo, P. 2019. Switchable whitening for deep representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1863–1871.

Peng, C.; Xiao, T.; Li, Z.; Jiang, Y.; Zhang, X.; Jia, K.; Yu, G.; and Sun, J. 2017. MegDet: A Large Mini-Batch Object Detector. *CoRR*, abs/1711.07240.

Singh, S.; and Krishnan, S. 2020. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11237–11246.

Wang, T.; Zhu, Y.; Zhao, C.; Zeng, W.; Wang, Y.; Wang, J.; and Tang, M. 2020. Large Batch Optimization for Object Detection: Training COCO in 12 minutes. In *European Conference on Computer Vision*, 481–496. Springer.

Wu, Y.; and He, K. 2018. Group Normalization. *CoRR*, abs/1803.08494.

Ye, C.; Evanusa, M.; He, H.; Mitrokhin, A.; Goldstein, T.; Yorke, J. A.; Fermuller, C.; and Aloimonos, Y. 2020. Network Deconvolution. In *International Conference on Learning Representations*.

Ye, C.; Mitrokhin, A.; Parameshwara, C.; Fermüller, C.; Yorke, J. A.; and Aloimonos, Y. 2018. Unsupervised Learning of Dense Optical Flow and Depth from Sparse Event Data. *CoRR*, abs/1809.08625.

Ye, C.; Yang, Y.; Fermüller, C.; and Aloimonos, Y. 2017. On the Importance of Consistency in Training Deep Neural Networks. *CoRR*, abs/1708.00631.

Zeiler, M. D.; and Fergus, R. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*, 818–833. Springer.