Understanding symmetries in deep networks

Vijay Badrinarayanan
Department of Engineering
Cambridge University, UK
vb292@cam.ac.uk

Bamdev Mishra
Amazon Development Centre India
Bangalore, India
bamdevm@amazon.com

Roberto Cipolla
Department of Engineering
Cambridge University, UK
cipolla@cam.ac.uk

Abstract

Recent works have highlighted scale invariance or symmetry present in the weight space of a typical deep network and the adverse effect it has on the Euclidean gradient based stochastic gradient descent optimization. In this work, we show that a commonly used deep network, which uses convolution, batch normalization, reLU, max-pooling, and sub-sampling pipeline, possess more complex forms of symmetry arising from scaling-based reparameterization of the network weights. We propose to tackle the issue of the weight space symmetry by constraining the filters to lie on the unit-norm manifold. Consequently, training the network boils down to using stochastic gradient descent updates on the unit-norm manifold. Our empirical evidence based on the MNIST dataset shows that the proposed updates improve the test performance beyond what is achieved with batch normalization and without sacrificing the computational efficiency of the weight updates.

1 Introduction

Stochastic gradient descent (SGD) has been the workhorse for optimization of deep networks [1]. The most well-known form uses the Euclidean gradients with a varying learning rate to optimize the weights. In this regard, the recent work [2] has brought to light scale invariance properties in the weight space which commonly used deep networks possess. These symmetries or invariance to reparameterizations of the weights imply that even though the loss function remains unchanged, the Euclidean gradient varies based on the chosen parameterization. Consequently, optimization trajectories can vary significantly for different reparameterizations [2].

Although these issues have been raised recently, the precursor to these methods is the early work of Amari [3], who proposed the use of *natural gradients* to tackle weight space symmetries in neural networks. The idea is to compute the steepest descent direction on the manifold defined by these symmetries and use this direction to update the weights [4, 5, 6, 7]. Most of theses proposals are either computationally expensive to implement or they need modifications to the architecture. On the other hand, optimization over a manifold with symmetries or invariances has been a topic of much research and provides guidance to other simpler metric constructions [8, 9, 10, 11, 12, 13, 14].

Our analysis into a commonly used network shows that there exists more complex forms of symmetries which can affect optimization, and hence there is a need to define weight updates that resolve these symmetries. Accordingly, we look at one particular way of resolving the symmetries by constraining the filters to lie on the unit-norm manifold. This results from a geometric viewpoint on the manifold search space. The proposed updates, shown later in Table 1, are symmetry-invariant and are numerically efficient to implement. The updates are implemented in Matlab and Manopt [15]. The codes are available at `http://bamdevmishra.com/codes/deepnetworks`.

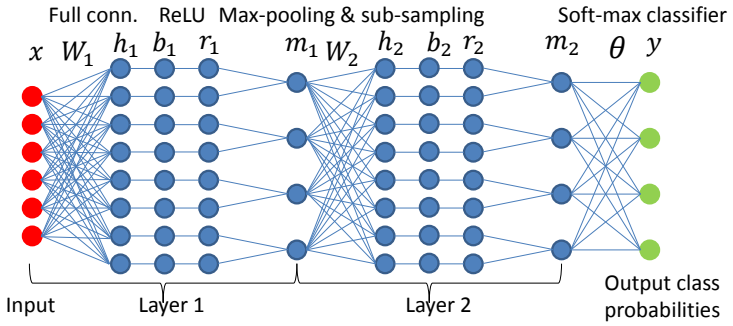

Figure 1: ArchBN: a two layer deep architecture for classification with batch normalization [17].

2 Architecture and symmetry analysis

A two layer deep architecture, ArchBN, is shown in Figure 1. Each layer in ArchBN has typical components commonly found in convolutional neural networks [16] such as multiplication with a trainable weight matrix (W_1, W_2), a batch normalization layer (b_1, b_2) [17], element-wise rectification ReLU, 2×1 max-pooling with stride 2, and sub-sampling. The final layer is a K-way soft-max classifier θ. The network is trained with a cross-entropy loss. The rows of the weight matrices W_1 and W_2 correspond to filters in layers 1 and 2, respectively. The dimension of each row corresponds to the input dimension of the layer. For the MNIST digits dataset, the input is a 784 dimensional vector. With 64 filters in each of the layers, the dimensionality of W_1 is 64×784 and of W_2 is 32×64. The dimension of θ is 10×32, where each row corresponds to a trainable class vector.

The batch normalization [17] layer normalizes each feature (element) in the h_1 and h_2 layers to have zero-mean unit variance over each mini-batch. Then a separate and trainable scale and shift is applied to the resulting features to obtain b_1 and b_2, respectively. This effectively models the distribution of the features in h_1 and h_2 as Gaussian whose mean and variance are learnt during training. Empirical results show this significantly improves convergence and our experiments also support this claim [17]. A key observation is that the normalization of h_1 and h_2 allows for complex symmetries to exist in the network. To this end, consider the reparameterizations

$$\tilde{W}_1 = \alpha W_1 \quad \text{and} \quad \tilde{W}_2 = \beta W_2, \tag{1}$$

where $\alpha = \text{Diag}(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8)$ and $\beta = \text{Diag}(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8)$ and the elements of α and β can be any real number. $\text{Diag}(\cdot)$ is an operator which creates a diagonal matrix with its argument placed along the diagonal. Due to batch normalization which makes h_1 and h_2 unit-variance, y is unchanged, and hence the loss is invariant to reparameterizations (1) of the weights. Equivalently, there exists continuous symmetries or reparameterizations of W_1 and W_2, which leave the loss function unchanged. It should be stressed that our analysis differs from [2], where the authors deal with a simpler case of $\alpha = \alpha_0$, $\beta = 1/\alpha_0$, and α_0 is a non-zero scalar.

Unfortunately, the Euclidean gradient of the weights (used in standard SGD) is not invariant to reparameterizations of the weights [2]. Consequently, optimization trajectories can vary significantly based on the chosen parameterizations. This issue can be resolved either by defining a suitable non-Euclidean gradient which is invariant to reparameterizations (1) or by placing appropriate constraints on the filter weights as we show in the following section.

3 Resolving symmetry issues using manifold optimization

An efficient way to resolve the symmetries that exist in ArchBN is to constrain the weight vectors (filters) in W_1 and W_2 to lie on the *oblique manifold* [8, 15], i.e., each filter in the fully connected layers is constrained to have unit norm (abbreviated UN). Equivalently, we impose the constraints $\text{diag}(W_1 W_1^T) = 1$ and $\text{diag}(W_2 W_2^T) = 1$, where $\text{diag}(\cdot)$ is an operator which extracts the diagonal

$\tilde{\nabla}_{W_1} L(W_1^t, W_2^t, \theta^t) = \Pi_{W_1^t}(\nabla_{W_1} L(W_1^t, W_2^t, \theta^t))$	$\tilde{\nabla}_{W_2} L(W_1^t, W_2^t, \theta^t) = \Pi_{W_2^t}(\nabla_{W_2} L(W_1^t, W_2^t, \theta^t))$
$W_1^{t+1} = \mathrm{Orth}(W_1^t - \lambda \tilde{\nabla}_{W_1} L(W_1^t, W_2^t, \theta^t))$	$W_2^{t+1} = \mathrm{Orth}(W_2^t - \lambda \tilde{\nabla}_{W_1} L(W_1^t, W_2^t, \theta^t))$
$\theta^{t+1} = \theta^t - \lambda \nabla_\theta L(W_1^t, W_2^t, \theta^t)$	

Table 1: The proposed UN symmetry-invariant updates for a loss function $L(W_1, W_2, \theta)$ in ArchBN. Here (W_1^t, W_2^t, θ^t) is the current weight, $(W_1^{t+1}, W_2^{t+1}, \theta^{t+1})$ is the updated weight, λ is the learning rate, and $\nabla_{W_1} L(W_1^t, W_2^t, \theta^t)$, $\nabla_{W_2} L(W_1^t, W_2^t, \theta^t)$, and $\nabla_\theta L(W_1^t, W_2^t, \theta^t)$ are the partial derivatives of the loss L with respect to W_1, W_2, and θ, respectively at (W_1^t, W_2^t, θ^t). The operator $\mathrm{Orth}(\cdot)$ normalizes the rows of the input argument. $\Pi_W(\cdot)$ is the linear projection operation that projects an arbitrary matrix onto the tangent space of the oblique manifold at an element W. It is defined as $\Pi_W(Z) = Z - \mathrm{Diag}(\mathrm{diag}((ZW^T))W$ [15].

elements of the argument matrix. To this end, consider a weight vector $w \in \mathbb{R}^n$ with the constraint $w^T w = 1$. (For example, w^T is a row of W_1.) The steepest descent direction for a loss $\ell(w)$ with w on the unit-norm manifold is computed $\tilde{\nabla}\ell(w) = \nabla\ell(w) - (w^T \nabla\ell(w))w$, where $\nabla\ell(w)$ is the Euclidean gradient and $\tilde{\nabla}\ell(w)$ is the Riemannian gradient on the unit-norm manifold [8, Chapter 3]. Effectively, the normal component of the Euclidean gradient, i.e., $(w^T \nabla\ell(w))w$, is subtracted to result in the tangential (to the unit-norm manifold) component. Following the tangential direction takes the update out of the manifold, which is then pulled back to the manifold with a *retraction* operation [8, Example 4.1.1]. Finally, an update of w on the unit-norm manifold is of the form

$$\begin{aligned} \tilde{\nabla}\ell(w) &= \nabla\ell(w) - (w^T \nabla\ell(w))w \\ \tilde{w}_+ &= w - \lambda \tilde{\nabla}\ell(w) \\ w_+ &= \tilde{w}_+ / \|\tilde{w}_+\|, \end{aligned} \tag{2}$$

where w is the current weight, w_+ is the updated weight, $\nabla\ell(w)$ is the Euclidean gradient, and λ is the learning rate. It should be noted that when W_1 and W_2 are constrained, θ is *unconstrained*.

The proposed weight update (2) can be used in a stochastic gradient descent (SGD) setting, which we use in our experiments in the following section. It should be emphasized that the proposed updates are numerically efficient to implement. The formulas are shown in Table 1. The convergence analysis of SGD on manifolds follows the developments in [1, 18].

4 Experiments and results

We train both two and four layer deep ArchBN to perform digit classification on the MNIST dataset (60K training and 10K testing images). We use 64 features per layer. The digit images are rasterized into a 784 dimensional vector as input to the network(s). No input pre-processing is performed. The weights in each layer are drawn from a standard Gaussian and each filter is unit-normalized. The class vectors are also drawn from a standard Gaussian and unit-normalized.

We use SGD-based optimization and choose the base learning rate from the set 10^{-p} for $p \in \{2, 3, 4, 5\}$ for each training run. For finding the base learning rate, we create a validation set of 500 images from the training set. We then train the network with a fixed learning rate using a randomly chosen set of 1000 images for 50 epochs. At the start of each epoch, the training set is randomly permuted and mini-batches are sampled in a sequence ensuring each training sample is used only once within an epoch. We record the validation error measured as the error per training sample for each candidate base learning rate. We then choose the candidate rate which corresponds to the lowest validation error and use this for training the network on the full training set. We repeat this whole process for 10 training runs for each network to measure the mean and variance of the test error. We ignore the runs where the validation error diverged. For each full dataset training run, we use the bold-driver protocol [19] to anneal the learning rate. We choose 50000 randomly chosen samples as the training set and the remaining 10000 samples for validation. We train for a minimum of 25 epochs and a maximum of 60 epochs. Training is terminated if either the training error is less than 10^{-5} or the validation error increases with respect to the one measured before 5 epochs or successive validation error measurements differ less than 10^{-5}.

Depth	B-SGD	UN
2	0.0206 $\pm$ 0.0024	**0.0199** $\pm$ 0.0046
4	0.0204 $\pm$ 0.0027	**0.0179** $\pm$ 0.0025

Table 2: The test error after the last training epoch measured over 10 runs on the MNIST dataset. The proposed UN update shows a competitive performance while resolving the symmetries.

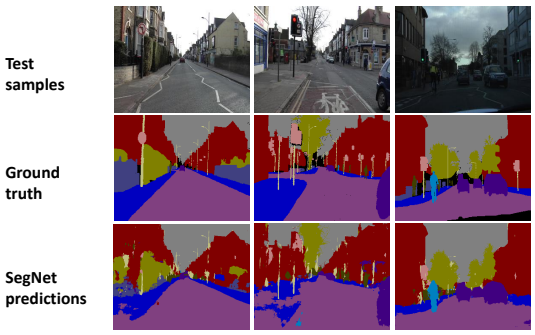

Figure 2: SGD with the proposed UN weight updates, shown in Table 1, for training SegNet [21]. The quality of the predictions as compared to the ground truth indicates a successful training of SegNet.

From Table 2 we see that for both two and four layer deep networks, the mean test error is lower for UN as compared to balanced SGD (B-SGD) which is simply the Euclidean update, but where the starting values of filters and class vectors are unit-normalized. The lowest mean and variance in the test error is obtained with UN weight updates on a four layer deep network. The difference between B-SGD and UN updates is more significant for the four layer deep network, thereby highlighting the performance improvement over standard batch normalization in deeper networks. The proposed UN weight updates can also be seen as a way to regularize the weights during training without introducing any hyper-parameters, e.g., a weight decay term. It should also be stressed that the performance difference between the two and four layer networks is not very large. This raises the question for future research as to whether some deep networks necessarily have to be that deep or it can be made shallower (and efficient) by better optimization [20].

5 Application to image segmentation

We apply SGD with the proposed UN weight updates in Table 1 for training SegNet, a deep convolutional network proposed for road scene image segmentation into multiple classes [21]. This network, although convolutional, possesses the same symmetries as those analyzed for ArchBN in (1). The network is trained for 100 epochs on the CamVid [22] training set of 367 images. The predictions on some sample test images from CamVid are shown in Figure 2. These qualitative results indicate the usefulness of symmetry-invariant weight updates for larger networks that arise in practice.

6 Conclusion

We have highlighted the symmetries that exist in the weight space of currently popular deep neural network architectures. These symmetries can be absorbed into gradient descent by applying a unit-norm constraint on the filter weights. This takes into account the manifold structure on which the weights of the network reside. The empirical results show that the test performance can be improved using our proposed weight updates on a modern architecture. As a future research direction, we would like to explore other efficient symmetry-invariant weight updates and exploit them for deep convolutional neural network used in practical applications.

Acknowledgments

Vijay Badrinarayanan and Roberto Cipolla were supported by a sponsorship from Toyota Motor Europe, Belgium. This work was initiated while Bamdev Mishra was with the Department of Electrical Engineering and Computer Science, University of Liège, 4000 Liège, Belgium and was visiting the Department of Engineering (Control Group), University of Cambridge, Cambridge, UK. He was supported as an FNRS research fellow (Belgian Fund for Scientific Research). The scientific responsibility rests with its authors.

References

[1] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics (COMPSTAT)*, pages 177–186, 2010.

[2] B. Neyshabur, R. Salakhutdinov, and N. Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2015. Accepted for publication.

[3] S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[4] R. Pascanu and Y. Bengio. Revisiting natural gradient for deep networks. Technical report, arXiv:1301.3584, 2013.

[5] G. Desjardins, K. Simonyan, R. Pascanu, and K. Kavukcuoglu. Natural neural networks. Technical report, arXiv:1507.00210, 2015.

[6] Y. Ollivier. Riemannian metrics for neural networks I: Feedforward networks. *Information and Inference*, 4(2):108–153, 2015.

[7] Y. Ollivier. Riemannian metrics for neural networks II: Recurrent networks and learning symbolic data sequences. *Information and Inference*, 4(2):154–193, 2015.

[8] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008.

[9] B. Mishra and R. Sepulchre. Riemannian preconditioning. Technical report, arXiv:1405.6055, 2014.

[10] N. Boumal and P.-A. Absil. Low-rank matrix completion via preconditioned optimization on the Grassmann manifold. *Linear Algebra and its Applications*, 475:200–239, 2015.

[11] M. Journée, F. Bach, P.-A. Absil, and R. Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization*, 20(5):2327–2351, 2010.

[12] P.-A. Absil, R. Mahony, and R. Sepulchre. Riemannian geometry of Grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematicae*, 80(2):199–220, 2004.

[13] A. Edelman, T.A. Arias, and S.T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[14] J.H Manton. Optimization algorithms exploiting unitary constraints. *IEEE Transactions on Signal Processing*, 50(3):635–650, 2002.

[15] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a matlab toolbox for optimization on manifolds. *The Journal of Machine Learning Research*, 15(1):1455–1459, 2014.

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine learning (ICML)*, 2015.

[18] S. Bonnabel. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013.

[19] G. Hinton. Lecture notes. Technical report, University of Toronto, 2008.

[20] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 2654–2662, 2014.

[21] V. Badrinarayanan, A. Handa, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. Technical report, arXiv:1505.07293, 2015.

[22] G. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.