Deep Orthogonal Neural Networks

Benjamin Nosarzewski

March 21, 2018

Abstract

Both empirical and theoretical results suggest that deep architectures are needed in order for artificial intelligence models to learn complex high-level abstractions without requiring an intractable number of parameters. However, instability of the gradient during back propagation has been a fundamental obstacle for deep artificial neural networks trained via gradient descent based methods. We demonstrate that the variance of the gradient and layer activations can be controlled or altogether held constant by using networks with hidden layers which apply orthogonal matrices followed by a normalization preserving nonlinear operation. Our results indicate these deep orthogonal neural networks can be trained at larger depths than standard feedforward neural networks with ReLU activations.

1 Introduction

According to a famous series of papers in the 1980's, circuit theory demonstrates that for certain functions such as computing parity a shallow circuit requires exponentially many more gates than a deep circuit [1]. This suggests that deep artificial neural networks may be able to efficiently learn a complicated mapping. Evolutionary insights further suggest the need for deep neural networks; the mammal brain which serves as inspiration for neural networks in general is organized in a deep architecture with a given input percept represented at multiple levels of abstraction each corresponding to a different area of the cortex [2]. Moreover the utility of depth in neural networks for many learning tasks is also evident from breakthroughs in artificial intelligence such as Deep Belief Networks in 2006 and more recently deep residual networks (ResNets) and densely connected networks (DenseNets) [2, 3, 4]. Residual networks produced state of the art performance on computer vision tasks and were trained with up to 1202 layers in the original paper [3]. One of the reasons that ResNets and DenseNets are so successful is that they mitigate an intrinsic instability of gradient descent based learning for deep neural network by allowing the gradient to skip intermediate layers. Without skip connections, the gradient is likely to exponentially decay across the network as each layer is not guaranteed to preserve the gradient norm, a problem which makes plain neural networks untrainable at large depths.

Nevertheless, recent results suggest that because of the nature of skip connections, ResNets and DenseNets may simply be an ensemble of shallower networks, widening of these networks is more effective than deepening, and their actual depth is unclear [5]. In this work we propose a novel type of neural network which avoids the exponentially vanishing gradient problem by using layer weights corresponding to orthogonal matrices and a norm-preserving nonlinear operation. Restricting layer weights to the manifold of orthogonal matrices restricts the available parameter space but still allows for an arbitrary degree of complexity if multiple orthogonal layers are stacked in a deep architecture. Training is performed via stochastic gradient descent with momentum by following geodesics along the manifold of orthogonal matrices. We find that our deep orthogonal neural networks are able to train at much larger depth than standard feed-forward networks with ReLU activations.

2 Definitions

Both typical layer weights and standard activation functions do not preserve norm of the signal or gradient during forward and back propagation. This problem is alleviated by choosing good initializations such as Xavier initialization, however this does not guarantee that the norm of the signal or gradient does not

exponentially decay across the layers after many iterations of training. We therefore propose to use weight matrices which correspond to an orthogonal matrix followed by a norm-preserving nonlinear operation. We will refer to layers performing these operations as orthogonal layers and a standard layer performing a linear operation followed by a ReLU as a ReLU layer. We note that orthogonal layers must have the same input and output dimensions because orthogonal matrices are necessarily square matrices. We initially tried generalizing orthogonal layers to unitary layers with weights which are complex numbers. This in fact inspired our non-linear operation which can be interpreted as taking the complex conjugate of those entries which have a positive real part. However, we found that restricting our weights to be real orthogonal matrices led to easier training compared to using weights corresponding to complex unitary matrices.

Forward propagation from the activations in layer $l-1$ to the activations in layer l through an orthogonal layer is defined as:

$$\begin{aligned} z^{[l]} &= Q^{[l]} a^{[l-1]} + b^{[l]} \\ a^{[l]} &= g(z^{[l]}) \end{aligned} \tag{1}$$

Where $Q^{[l]}$ is an orthogonal matrix, $b^{[l]}$ is a bias term, and g is the nonlinear operation. We define the nonlinear operation g as follows. If $a = g(z)$ where a and z are vectors of length n indexed from 1 to n, then:

$$z_i = \begin{cases} +a_i & \text{if } i \leq n/2, \\ -a_i & \text{if } i > n/2 \text{ and } a_{i-n/2} < 0, \\ +a_i & \text{otherwise.} \end{cases} \tag{2}$$

This operation flips the sign of elements in the second half of the vector a if the corresponding element in the first half of the vector is negative. Since $\partial z_i / \partial a_i = \pm 1$, this operation conserves the L2 norm of the activations and gradients during forward and backward propagation.

3 Gradient Analysis

We now analyze the gradient flow through the network. Let J denote the cost which we take as softmax cross entropy. Because the nonlinear operation preserves norm during back propagation, we have that

$$\text{Var}(\partial J / \partial z^{[l]}) = \text{Var}(\partial J / \partial a^{[l]}) \tag{3}$$

We then find that

$$\begin{aligned} \text{Var}(\partial J / \partial a^{[l-1]}) &= \text{Var}(Q^{[l-1]} \partial J / \partial z^{[l]}) \\ &= \text{Var}(\partial J / \partial z^{[l]}) \\ &= \text{Var}(\partial J / \partial a^{[l]}) \end{aligned} \tag{4}$$

which proves that the magnitudes of the gradients of the layer activations are perfectly constant during backpropagation in an orthogonal network!

Because of the bias term, the magnitudes of the layer activations are not perfectly constant during forward propagation. However, the resulting changes from layer to layer are *additive*. Based on the equation below, this additive change is reflected in the backpropagation of the gradient for the orthogonal matrix weights (the gradient depends on the size of the layer activations). Similarly, the gradient of the bias involves an average over training examples in the batch, a process which also additively changes the size of the gradient. We conclude that the gradients of the parameters in the network cannot exponentially vanish, as we observe in our experiments.

$$\begin{aligned} \frac{\partial J}{\partial Q^{[l]}} &= \frac{1}{m} \frac{\partial J}{\partial z^{[l]}} (a^{[l-1]})^T \\ \frac{\partial J}{\partial b^{[l]}} &= \frac{1}{m} \sum_{j=1}^{m} \frac{\partial J}{\partial z^{[l](j)}} \end{aligned} \tag{5}$$

where m is the batch size and $z^{[l](j)}$ denotes the j^{th} training example in the l^{th} layer.

4 Training Algorithm

We use results from differential geometry to train deep orthogonal neural networks [7]. The training procedure involves performing gradient descent constrained to the manifold of orthogonal matrices, a special case of the Stiefel manifold. We use geodesics along this manifold to update the parameters of the orthogonal matrices and also parallel transport the tangent space projection of the gradient to the new updated location on the manifold when performing each iteration of gradient descent with momentum.

Suppose Z is a direction in which we want to change our orthogonal matrix Q. In other words, $Z = -\partial J/\partial Q$ which we obtain from back propagation. Projection of Z onto tangent space at Q is given by:

$$\pi_T(Z) = QA + (I - QQ^T)Z \tag{6}$$

where $A = \frac{1}{2}(Q^T Z - Z^T Q)$. The geodesic equation in direction $\pi_T(Z)$ from Q is then given by:

$$Q(t) = Q(0)e^{At} \tag{7}$$

where t sets the distance traveled along the geodesic curve. Parallel transport of a matrix $\Delta \equiv Q(t)B(t)$ in the tangent space of Q along the geodesic is then given by:

$$\Delta = Q(0)e^{At/2}B(0)e^{At/2} \tag{8}$$

However, a geodesic parallel transports its own tangent vector so for the case of parallel transporting the vector $\pi_T(Z) \equiv V$ we can simply use:

$$V(t) = V(0)e^{At} \tag{9}$$

For gradient descent with momentum we first calculate $Z' = \beta Z_0 + (1 - \beta)\pi_T(Z)$ where Z_0 is the parallel transported vector from the previous step of gradient descent. We then calculate the geodesic in the direction Z' and also parallel transport the matrix Z' to the new location on the manifold and set it equal to Z_0. The parallel transport step ensures that the old gradient is in the tangent space at the new location on the manifold. Matrix exponentiation when calculating the geodesic path is in principle costly because it will take $O(n_h^3 L)$ time where L is the number of layers and n_h is the number of hidden units. However, the advantage of deep networks is that they use fewer hidden units where n_h is relatively small and so calculating the geodesic may take less time than forward and back propagation.

5 Methods

We used gradient descent with momentum on the orthgonal manifold with $\beta = 0.9$ and a learning rate of around 10^{-3}. Traditional regularization methods such as L2 and dropout can be used on the ReLU layers in an alternating Orth/ReLU architecture (discussed below) and help to prevent over-fitting in the model as seen in Fig 3. These regularizations cannot be used on the orthogonal layers as they would change the norm during forward and back propagation.

6 Results

6.1 Purely Orthogonal Network

We first tested our networks on the simple classification task of predicting the class of 200 points distributed with various shapes in a two dimensional space. Because of the simplicity of the dataset, standard shallow neural networks can easily accomplish the classification task. We make the problem much more challenging by restricting the networks be very narrow (small number of hidden units) in order to force the networks to use their depth. In this case standard networks fail due to vanishing gradients.

Fig 1a demonstrates the exponentially vanishing gradient for a standard network with 10 layers. Gradients for networks with less than 10 layers are stable but as soon as the depth increase to 10 layers or more the

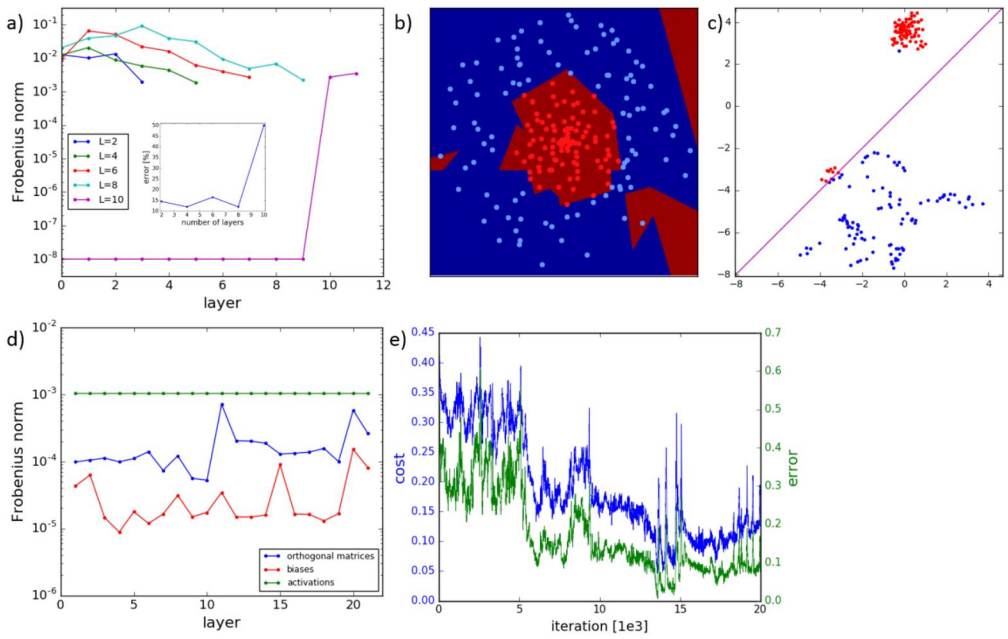


Figure 1: a) Plot of the gradients of the layer activations for standard feedforward neural networks of variable depths with ReLU activations and 2 hidden units trained on the dataset shown in the next panel. Inset shows the classification error for each of the network sizes. b) Classification boundaries after training deep orthogonal network with 20 layers and 2 hidden units in each layer. c) Activations of the last layer before softmax cross entropy. The line x=y should ideally separate the red and blue classes. d) Magnitude of the gradients for the activations, orthogonal parameters, and bias parameters. e) Cost function and error during training.

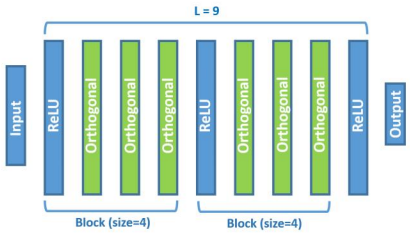


Figure 2: Example Orth/ReLU architecture. All layers are fully-connected.

gradient completely vanishes after back propagation over two layers. In contrast, as shown in Fig 1(b,c,d,e), a deep orthogonal network with 20 layers and the same number of hidden units is able to achieve an error of 1% when trained on the same dataset. The gradients of the activations are constant and the gradients of the neural network parameters do not exhibit exponential decay. The 1% error is better performance than that of the standard feedforward networks at any depth for the same number of hidden units which have errors greater than 10%. All layers in this particular orthogonal network are orthogonal layers since the input and all activations are two dimensional. For this case we can also exactly visualize what each layer in the network does because it is possible to plot the activations in two dimensions. The video at this link is created by taking each of the layer activations as a video frame and linearly interpolating between the frames. The activations of the final layer successfully segregate 99% of the points across the softmax decision boundary $x = y$ as shown in Fig 1c.

6.2 Orth/ReLU architecture

We experimented with networks combining standard layers with ReLU activation functions with orthogonal layers which we refer to as Orth/ReLU networks. Fig 2 shows a diagram of the architecture. We tested the networks on a dataset consisting of a spiral of points in two dimensions as shown in Fig 3. For alternating Orth/ReLU architectures, one should set the number of orthogonal matrices per block (see Fig 2) to the minimum number for which the gradient variation is no more than roughly two or three orders of magnitude across the layers. Larger networks will require a higher ratio of orthogonal to ReLU layers. More frequent ReLU's seem to lead to smooth cost functions during training, but too frequent ReLU's cause gradient instability. We did manage to train good models with fairly unstable gradients (variations over 3 orders of magnitude). However, in this case the gradients would occasionally become unstable and prevent training. On the other hand, networks with stable gradients and a high density of unitary layers demonstrated unpredictable training behavior. In this case the network would sometimes require multiple training attempts, sometimes gradient descent would not converge for certain random initializations, and other times the network would quickly converge after a long period during which the cost was nearly constant. In summary, the trade-off of for gradient stability with orthogonal layers is unpredictability and increased noise in the cost function during training. Nevertheless, orthogonal layers make training possible of an otherwise untrainable model by solving the vanishing/exploding gradient problem. In future work we hope to develop ways to improve the training behavior of orthogonal neural networks.

6.3 MNIST

We test orthogonal networks on the MNIST dataset of handwritten digits which is a standard benchmark for various algorithms. We chose this dataset because of the small size of the images and because we were surprised to learn that the best performing single feedforward network performs as well as the best single convolutional network (test error = 0.35%) [8]. Our primary goal was not to achieve good performance on MNIST, but to demonstrate once again that orthogonal networks can be trained at larger depths than standard networks. We find that for the case of 4 hidden units, standard networks cannot be trained at a depth of 50 units because of the vanishing gradient problem. In contrast, gradients of our orthogonal networks are stable at this depth and the cost function and classification error decrease during the training as seen in Fig 4b. Because the dimensions of the input and output are not equal to 4 for MNIST, the first

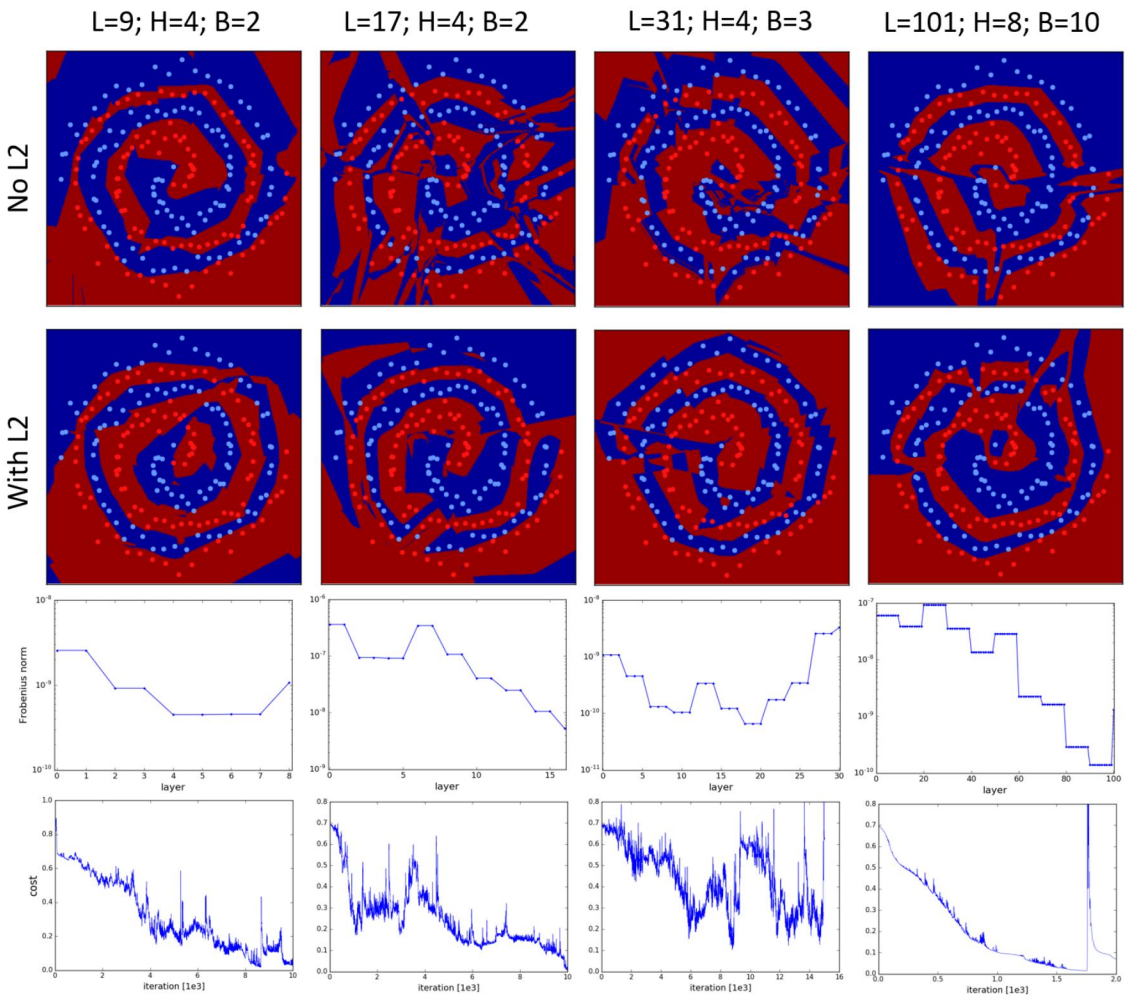
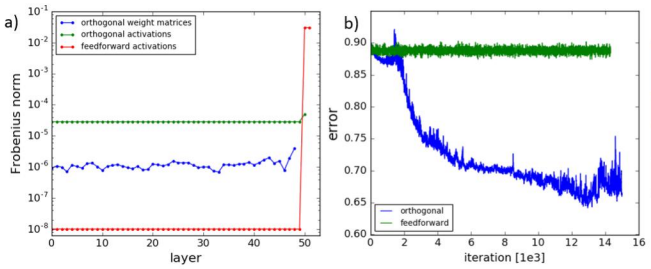


Figure 3: Results of alternating Orth/ReLU architectures on spiral dataset. L is the number of layers, H is the number of hidden units, and B is the block size as defined in Fig 2. The size of the gradients and the cost functions are shown for those networks where L2 regularization was applied ($\lambda = 0.001$).



c)

Error	Feedforward Network	Orthogonal Network
Train	89%	64%
Validation	89%	64%
Test	89%	64%

Figure 4: a) Gradients for the feedforward network and orthogonal networks trained on the MNIST dataset. Number of hidden units = 4. b) Error vs training iteration for feedforward and orthogonal networks. c) Training, validation, and test error for feedforward and orthogonal networks.

and last layer of the network consist of a matrix multiplication followed by an application of the norm-preserving nonlinear operation. The remaining 50 internal layers are orthogonal layers. We hope to achieve good performance on MNIST and other real world datasets with orthogonal networks in future work.

7 Conclusion

Orthogonal layers are useful for situations where increased neural network depth is important to the performance of the model and gradients vary by over two orders of magnitude between layers. In these cases, using orthogonal layers can successfully help control the stability of gradients in the network and allow the training of an otherwise non-trainable model, although convergence may become harder to attain. We hope that our work has made the possibility of training very deep networks without skip connections a reality and will inspire future advancements in this direction by demonstrating that gradients can be stabilized by using layers which preserve norm during both forward and back propagation.

8 code repository

https://github.com/bennosski/OrthAI

References

[1] M. Nielsen, Neural Networks and Deep Learning, 2017.

[2] Y. Bengio, Learning Deep Architectures for AI, Vol. 2, No. 1 (2009).

[3] K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, arXiv:1512.03385, 2015.

[4] G. Huang, Z. Liu, L. Maaten, and K. Weinberger, Densly Connected Convolutional Networks, arXiv:1608:06993v5.

[5] S. Zagoruyko and N. Komodakis, DiracNets: Training Very Deep Neural Networks Without Skip-Connections, arXiv:1706.00388, 2018.

[6] A. Viet, M. J. Wilber, and S. J. Belongie, Residual networks are exponemtial ensembles of relatively shallow networks. CoRR, abs/1605.06431, 2016.

[7] A. Edelman, T. A. Arias, and S. T. Smith, The geometry of algorithms with orthogonality constraints, Siam J. Matrix Anal. Appl., Vol. 20, No. 2, pp. 303-353.

[8] Y. LeCun, MNIST webpage, $http://yann.lecun.com/exdb/mnist/$.