Stochastic Orthogonalization and Its Application to Machine Learning

Yu Hong

Southern Methodist University, hongy@smu.edu

Follow this and additional works at: https://scholar.smu.edu/engineering_electrical_etds

Part of the Artificial Intelligence and Robotics Commons, and the Theory and Algorithms Commons

Recommended Citation


https://scholar.smu.edu/engineering_electrical_etds/31

This Thesis is brought to you for free and open access by the Electrical Engineering at SMU Scholar. It has been accepted for inclusion in Electrical Engineering Theses and Dissertations by an authorized administrator of SMU Scholar. For more information, please visit http://digitalrepository.smu.edu.
STOCHASTIC ORTHOGONALIZATION

AND ITS

APPLICATION TO MACHINE LEARNING

A Master Thesis Presented to the Graduate Faculty of the

Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Master of Science

in Electrical Engineering

by

Yu Hong

B.S., Electrical Engineering, Tianjin University 2017

December 21, 2019
ACKNOWLEDGMENTS

First, I want to thank Dr. Douglas. During this year of working with him, I have learned a lot about how to become a good researcher. He is creative and hardworking, which sets an excellent example for me. Moreover, he has always been patient with me and has spent a lot of time discuss this thesis with me. I would not have finished this thesis without him.

Then, thanks to Dr. Larson for giving me the inspiration to do this thesis. Last year, I had just started with machine learning and was doing some preliminary simulation. On research day, Dr. Larson enlightened me with his talk and sent me a paper which inspired me to write this thesis.

Finally, thanks to Dr. Rohrer, who has given me lots of freedom to work on this thesis. I have had bad health condition this year and really appreciate the understanding. I am looking forward to learn more from him.
Orthogonal transformations have driven many great achievements in signal processing. They simplify computation and stabilize convergence during parameter training. Researchers have introduced orthogonality to machine learning recently and have obtained some encouraging results. In this thesis, three new orthogonal constraint algorithms based on a stochastic version of an SVD-based cost are proposed, which are suited to training large-scale matrices in convolutional neural networks. We have observed better performance in comparison with other orthogonal algorithms for convolutional neural networks.
# TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION ......................................................... 1  
1.1. Introduction to Machine Learning ........................................ 1  
1.2. Convolutional Neural Networks .......................................... 2  
1.3. Challenges in CNN System Design ....................................... 6  
1.4. Contribution of Thesis .................................................... 8  
1.5. Outline of Thesis .......................................................... 9  

CHAPTER 2: PRINCIPLES OF ORTHOGONALITY ................................. 10  
2.1. Overview of Orthogonality ............................................... 10  
2.2. Orthogonality Applied to CNNs ......................................... 14  
2.3. Applications of Orthogonality in Machine Learning ................ 16  
2.4. Other Orthogonal Regularization ....................................... 18  

CHAPTER 3: STOCHASTIC ORTHOGONALIZATION .............................. 21  
3.1. Introduction ...................................................................... 21  
3.2. Derivations and Analysis .................................................. 21  

CHAPTER 4: APPLICATION TO DEEP LEARNING ............................... 30  
4.1. Description of Experiments .............................................. 30  
4.2. Results ........................................................................... 34  

CHAPTER 5: CONCLUSION AND FUTURE WORK ............................... 37  
5.1. Conclusion ...................................................................... 37  
5.2. Future Work ..................................................................... 37  

REFERENCES ............................................................................ 38
CHAPTER 1
INTRODUCTION TO MACHINE LEARNING

1.1. Introduction to Machine Learning

We are living in a world of data and information. The books you read, the photos of journalists and the sounds of broadcasts are all data. A good understanding of the information around us enables us to make better choices in response to our surroundings and brings us better, more comfortable lives. It is impossible, however, for a human to memorize everything around them and take action based on the information. So, people are considering the use of computers to do it for us. This portends the rise of machine learning.

Machine learning (ML) is a branch of Artificial Intelligence (AI) and includes some of the most vital methods to implement AI. Through ML, computers can process, learn from, and draw insights of big data. For really big data, computers already are more effective than humans. When one opens a website to search for something, massive supercomputers almost instantly provide the information that is most interesting. When one talks to a live AI in computers and other devices, it can recognize what is said and execute commands. When one looks at an expensive smartphone, it can recognize the user and unlock itself. Already, highly developed AI can do simple works in place of a human.

Research in ML has experienced a transformation from focusing on the inference process, the knowledge to make choices, and finally the learning method. It has become a combination of probability theory, statistics and approximation theory. Generally, the goal of machine learning is to find the strong relationship data-sets and make predictions based on them.

Data and learning methods are two critical components of ML. According to the type of available data, ML could be classified as either supervised learning or unsupervised learning. How do we know a cat is a cat? First, we make a definition based on the features of a cat:
small animal, fuzzy, meows, and catches mice. The more features we know about cats, the more accurately we can recognize them. For supervised learning, a computer learns on the features of the data already known and later uses this knowledge to classify the unknown. For unsupervised learning, unknown features are classified based on how different they are. ML can also be parsed by architecture, such as neural network, decision tree, dimensionality reduction, metric learning, clustering, and Bayesian classifier as some examples, based on the learning method and the algorithm used to analyze the data.

Artificial neural networks were developed decades ago and they have become the most popular method of supervised learning recently due to their freely designed architectures, high prediction accuracy, and speed. Years ago, computers had neither large memories nor high-speed processors, so neural networks were hard to test and evaluate. Most recently there have been rapid advances in computer architecture and associated hardware that provides the possibility to run neural networks on many devices.

The concept of a neural network was inspired by the structure of the human brain. A common neural network has the following components:

**Neurons**: The computational units in a neural network are imitations of a biological neuron, which contains pieces of information. Like a human brain, lower layer neurons receive raw information and process it, then pass the information to subsequent layers of neurons.

**Weights and Connection**: Neurons connect to adjacent neurons, where each connection provides the output of one neuron to be an input to another neuron. Each connection is assigned a weight that represents its relative importance. A given neuron can have multiple input and output connections.

**Propagation Function**: The propagation function computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. A bias term, which is used to adjust the output along with the weighted sum of the inputs to the neuron, is added to the result of the propagation.

### 1.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a popular class of neural networks but they
are not new. Their concept of them was proposed nearly 30 years ago [1].

The basic idea of CNNs was biologically-inspired by Hubel and Wiesel’s work on the cat’s visual cortex [1]. The visual cortex contains cells with a complex arrangement that are sensitive to small regions of visual field called the receptive field. Those receptive fields are tiled to cover the entire visual field. These cells are thought to be acting as local filters over different input spaces and can provide enough information to encompass the entire input image. This relationship between cells is called sparse connection.

Inspired by sparse connections, CNNs locally connect the pattern between neurons of adjacent layers. That is, the inputs of hidden units in layer $m$ are from a subset of neighbor units rather than all units in layer $m - 1$.

Convolution is a better imitation of these receptive fields. In a CNN, normal multiplication is replaced by a convolution operation in order to reduce the number of parameters to be trained within the neural network, which gives rise to the their name.

In additional to their structure, CNNs use shared weights further to simplify models. The same filters are used for different receptive fields, and their outputs form a feature map. This replication of filters allows features to be analyzed regardless of their position and further increases the efficiency by greatly reducing the number of free parameters to be
Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically in a 2 x 2 region. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. [2] Average pooling uses the average value from each of a cluster of neurons at the prior layer.

CNNs are trained using a process called back-propagation. [3] The parameters of a CNN are fine-tuned based on the loss from previous training epochs. In order to introduce back-propagation, we have to understand forward-propagation first. In CNNs, forward propagation is a strategy that passes information from the first layer to the last layer. The output of each layer is called the activation and becomes the input of the next layer. Forward-propagation is a feed-forward propagation, which means any changes in input will result in a change in output. The goal of ML is to have the system adjust itself based on its current output in comparison to what has resulted from training examples.
Back-propagation is a process whereby the parameters of a CNN are updated according to the difference between the network’s output labels and the labels it is supposed to produce. The difference is the error and the measure of the total error is the loss. We take the derivative of the loss with respect to the parameters and subtract a portion of that gradient from the old parameters to obtain the new parameters in an iterative process.

The underlying approach to training is called stochastic gradient descent, often abbreviated as SGD. It is an iterative method for optimizing an objective function with suitable smoothness properties. It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient calculated from the entire data set by an estimate thereof calculated from a randomly selected subset of the data. Especially in big data applications, this reduces the computational burden, achieving faster computational iterations.

The mathematical expression of SGD is:

$$W_{new} = W_{old} - \mu \frac{\partial Loss(W_{old})}{\partial W_{old}}$$  \hspace{1cm} (1.1)

where $W$ are the weight matrices, and the parameter $\mu$ is the step size. Then, as the loss
is minimized, the parameters approach optimized values for the given recognition problem. Back-propagation is a key method for auto-adaptive CNNs.

As knowledge of CNNs accumulates, CNNs are applied to sound recognition, languages translation, and other interesting problems [4], [5].

In addition to the application of CNNs, the architectures and core algorithms have evolved as the tasks for CNNs have changed from dealing with small data-sets to large data-sets. Deeper neural network from simple five layers LeNet and AlexNet to hundreds of layers ResNet, which was designed to improve the accuracy of prediction [6], [7], [8]. This explosion in size engenders a big problem: if each layer has individual parameters, there will be large numbers of parameters during training that creates a huge burden for computers. Even with modern electronic fabrication, the complexity is large, leading to challenges in coding and training time. Thus, there has been an effort to optimize the learning parameters due to the limitation of hardware and improve the performance of CNNs.

Over years, several CNN architectures and algorithms have made great progress to solve those problems. CNNs started from the simple 5 layer LeNet [7], which was built to deal with the simple MNIST problem and only attained 54 percent recognition accuracy in 1986. Twenty-six years later, as 8 layers AlexNet with 74 percent accuracy won the ImageNet Large Scale Visual Recognition Challenge(ILSVRC) in 2012, a much more difficult task [6]. Hence, the potential of CNNs was realized, and further developments ensured over these past seven years. In the ILSVRC 2014 challenge, a large-scale visual recognition challenge, almost every highly ranked team used a CNN as their basic framework, and the winner, GoogLeNet [9], increased the mean average precision of object detection to 0.439329 and reduced classification error to 0.06656 (top-5 error), the best result to date. In 2015, with competitive performance, a many-layered CNN demonstrated the ability to spot faces from a wide range of angles, including upside down, even when partially occluded. The number of layers steadily is increasing and new structures have been created such as residual blocks [8]. Now, 100-layer systems are common.

1.3. Challenges in CNN System Design

There are several problems that all CNN designers face:
Stability: The gradient represents the sensitivity from the output of a CNN to its input. The stability of the gradient allows a CNN to ignore small changes/distortions in input while providing an accurate result. Gradient vanishing and exploding problems arise during training of a deep network when the gradients are being propagated back to the initial layer. The gradients coming from the deeper layers have to go through continuous matrix multiplications because of the chain rule, and as they approach the earlier layers, if these weights have small values ($|.| < 1$), they shrink exponentially until they nearly vanish and make it impossible for the model to learn. This is the vanishing gradient problem. On the other hand if they have large values ($|.| > 1$) they get larger and eventually blow up and cause training to fail. This is the exploding gradient problem.

One way to address this issue is to use well-chosen initialization strategies as shown by Kaiming He [10]. Previously, Researchers used zero initialization, random initialization, but they are not good solutions to this problem. In 2015, He proposed a new initialization which is well suited for CNNs with ReLU as activation functions. The goal of He’s initialization is to keep the standard deviation of layers’ activations near a magnitude of one, which will allow the stacking of several more layers in a deep neural network without gradients exploding or vanishing. It turns out that when a ReLU activation is used, the average standard deviation of a single layer will be very close to the square root of the number of input connections, so we compute the following:

$$W = W_0/\sqrt{l/\sigma}$$  

where $W_0$ is the weights initialized using random unit-variance Gaussian signals, $l$ is the number of input nodes for a layer and $\sigma$ is a constant for different activations(in this case, 2). He demonstrated that deep networks (e.g. a 22-layer CNN) would converge much faster when using this initialization.

Overfitting: Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model when applied to new data. This means that the noise or random fluctuations in the training data is learned as examples
by the model. The problem is that these examples do not apply to new data and negatively impact the model’s ability to generalize.

In order to address this problem, several approaches have been developed:

1). Using a very large input data-set. A large data-set includes more details to be learned, which enables the network to better ignore the noise and better to recognize all the features. However, training a neural network with a large data-set as input greatly increases training time.

2). Introduce dropout [11]. Dropout, which refers to randomly ignoring neurons in neural networks during training, prevents units from co-adapting too much. This significantly reduces overfitting and provides some improvement. Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.

3). Using a well-designed weight regularization. A weight regularization will add a cost to the loss function of the network for large weights. As a result, the model is simplified and will be forced to learn only the relevant patterns in the train data.

Orthogonality preserves the energy for a filter bank in signal processing. One idea is to impose orthogonality on a CNN to stabilize the gradient. Actually, a orthogonal initialization of each layer’s parameters already has made a breakthrough for this problem [12].

1.4. Contribution of Thesis

This thesis proposes a new class of iterative orthogonalization methods called stochastic orthogonalization, which are designed to make the columns of matrices to be orthogonal. It is designed for large-scale matrices orthogonalization. When applied to large matrices, our method saves computations at each iteration.

Unlike common iterative orthogonal algorithms that change the gradient update function directly, we impose orthogonality by using a stochastic version of a loss function. Based on our first algorithm, we also derive two similar algorithms, one of which has even less computational complexity.

Then we apply our stochastic orthogonalization to Machine Learning training, specifically the training of CNNs, and have obtained encouraging results. As designed, our
method runs faster than a well-performing algorithm in [13], which imposes orthogonality by introducing an additional cost to cost function, and also exhibits better accuracy.

1.5. Outline of Thesis

Chapter 2 introduces the theory of orthogonal matrices and reviews its application in Principal Component Analysis (PCA), Independent Component Analysis (ICA), and neural networks. Then we review the use of advanced orthogonalization algorithms applied to CNNs.

Chapter 3 presents the derivation of the stochastic orthogonalization methods and derives three new algorithms based on it. We show those algorithm’s strengths and weaknesses.

In Chapter 4, stochastic orthogonalization is imposed to one of the modern convolutional neural networks, ResNet. We analyse the performance of the method and make comparisons to algorithms of others.

Chapter 5 presents the conclusions and briefly describes future work.
CHAPTER 2
PRINCIPLES OF ORTHOGONALITY

2.1. Overview of Orthogonality

The term orthogonal is derived from a Greek word 'orthogonios'——'ortho' means right and 'gon' means angle. Orthogonality, the synonym to perpendicularity, has been studied in linear algebra, Euclidean geometry, and spherical trigonometry for a long time. In Euclidean space, two lines are orthogonal when they are perpendicular at their intersection point. Two vectors are orthogonal if and only if the inner product is zero, or

\[ \mathbf{a}^T \mathbf{b} = 0, \]

where \( \mathbf{a} \) and \( \mathbf{b} \) are vectors in an \( n \)-dimensional Euclidean space. Orthogonality can be extended as a concept to sets of vectors. These concepts can be extended further. For example, for two matrices \( \mathbf{A} \) and \( \mathbf{B} \), if each vector in \( \mathbf{A} \) is orthogonal to each vector in \( \mathbf{B} \), we call them jointly orthogonal matrices; such that \( \mathbf{A}^T \mathbf{B} = 0 \)

In some cases, normal could also be used to augment orthogonal. The term, orthonormality, is an extension of orthogonality to unit length vectors. In linear algebra, two vectors in an inner product space are orthonormal if they are orthogonal and are each of unit length. This leads to a concept of an orthonormal matrix. A \( n \times m \) matrix \( \mathbf{A} \) is orthonormal if

\[ \mathbf{A}^T \mathbf{A} = \mathbf{I} \tag{2.1} \]

where \( \mathbf{A}^T \) is the transpose of matrix \( \mathbf{A} \) and \( \mathbf{I} \) is a \( m \times m \) identity matrix. That means, if \( \mathbf{A} \) is also square, \( m = n \), the matrix \( \mathbf{A} \) is always invertible such that

\[ \mathbf{A}^{-1} = \mathbf{A}^T \tag{2.2} \]

Thus, in this case,

\[ \mathbf{A} \mathbf{A}^T = \mathbf{I} \tag{2.3} \]
In addition, the columns of an orthogonal matrix come from an orthogonal subspace. That is, columns of an orthogonal matrix are mutually perpendicular.

Orthonormality also has a strong relationship with the singular value decomposition (SVD) in linear algebra. SVD is an approach to factorize a real or complex matrix, to obtain a specific decomposition. Suppose $W$ is a matrix in real space $\mathbb{R}^{n \times m}$. Then, the singular value decomposition of $W$ has the form:

$$ W = U \Sigma V^T $$

(2.4)

where $U$ is a unitary matrix, whose inverse equals it conjugate transpose, and the $n \times n$ orthogonal matrix, $V$ is also a unitary and $m \times m$ orthogonal matrix, and $\Sigma$ is a unique diagonal matrix with non-negative real singular values on the diagonal where the singular values are placed in descending order.

Orthogonality and orthonormality are important mathematical properties for vectors and matrices, and they have been explored in some detection and estimation tasks.

For example, PCA, a statistical factor analysis procedure, uses an orthogonal transformation to convert a large set of possibly correlated variables into a small set of linearly uncorrelated variables to maintain the important information among a large data set. The smaller-dimensional data-set are called principal components. Mathematically, the transformation could be defined by a set of $m$ $n$-dimensional vectors of weights $w_i$ in a matrix $W = [w_1 \ w_2 \ ... \ w_m]$. Given a set of vector $x(n), 1 \leq n \leq N$, PCA maps each vector to a new vector of principal components $y$ as:

$$ y(n) = W^T x(n) $$

(2.5)

Developed in 1901 by Karl Pearson [14], PCA has been refined over the last century. Now, PCA is applied in many fields as an exploratory data analysis method, and we can build predictive models based on it [15].

Orthogonality also plays an important role in independent component analysis (ICA). ICA is a computational method to separate a set of mixed signals and transform them to
many independent components. ICA is based on a linear mixing model of the form

\[ x(n) = Ws(n) \]  

(2.6)

where \( s(n) \) is an \( m \)-dimensional set of independent signals and \( W \) is an \( n \times m \) mixing matrix. The goal is to separate the mixed signals in \( x(n) \) using a linear transformation. When a prewhitening step is performed, it turns out that the matrix \( W \) has orthonormal columns, such that a separation step of the form

\[ y(n) = W^T x(n) \]  

(2.7)

can be used.

For both PCA and ICA, we use weight vectors \( W = [w_1 \ w_2 \ ... \ w_m] \) to represent the orthogonal transformation, and these weight vectors have the property:

\[ w_i^T w_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \]  

(2.8)

In the case of ICA, these constraints are imposed in certain algorithms, such as the well-known FastICA algorithm [16], to solve a joint diagonalization task. More recently, orthonormality has been proposed as a parameter constraint to improve the performance of both recurrent neural networks and deep learning systems for real-world classification tasks [17], an issue we will consider in depth in this thesis.

Since the orthogonality transformation is important, the ways of imposing orthogonality on vectors have been extensively explored. Orthogonal weight matrices render analysis easy and are an important property of these matrices, but we are usually given an non-orthogonal matrix. So, an algorithm is necessary to transform \( W \) to an orthogonal set, \( W_{new}^T W_{new} = I \), is required. The most well-known approach is the Gram-Schmidt procedure [18]. Given any vector set \( w \), the Gram-Schmidt orthogonalization process can form an orthogonal set \( w_{new} \) as

\[ \widetilde{w}_k = w_k - (\alpha_{1,k}w_{1,new} + \ldots + \alpha_{k-1,k}w_{k-1,new}), 1 \leq k \leq n \]  

(2.9)

\[ \alpha_{j,k} = w_{j,new}^T w_k \]  

(2.10)

12
Based on the knowledge of SVD, there is a better approach to imposing orthonormality on the matrix $W$. From Eqn(2.4), we know the left and right singular vectors of $U$ and $V$ are orthogonal vectors and have unit length. So if we want to make an arbitrary matrix $W$ orthogonal, we can do:

$$W_{\text{new}} = U_m V^T$$  \hspace{1cm} (2.12)

where $U$ and $V$ are computed from the original $W$, $U_m$ is composed of the first $m$ columns of $U$, and the new $W$ is also orthonormal.

The concept of this SVD-based orthonormalization method is easy though it is hard to do the computation. Fortunately, there is an iterative way to implement it. One simple Newton-based approach is [19]

$$W_{\text{new}} = \frac{3}{2} W - \frac{1}{2} W W^T W$$  \hspace{1cm} (2.13)

The above iterative equation converges fast and finally fulfill $W_{\text{final}} = U_m V^T$.

Joint orthogonality of a set of vectors plays an important role in many estimation, detection, and classification tasks. In signal processing, orthogonality is employed in dimensionality reduction when searching for signal or noise subspaces, in the design of filter banks for signal coding tasks [20], [21], and in the specification of beamformers for array processing systems [22]. In these applications, classic techniques for orthogonalization are often used.

Recently, coefficient orthogonality has been leveraged to improve the performance of deep learning architectures when applied to classification tasks. In this application, orthogonality is used to influence the adaptation of each weight matrix $W$ associated with a specific layer within the architecture. In [23], random orthogonal weight matrix initialization is indicated to provide depth-independent learning dynamics; additional studies on orthogonal matrix initialization are in [12] [24]. Numerous works have sought to impose some measure of orthogonality on the weight matrices during adaptation as well, usually by augmenting the cost function at each layer with an additional term $\lambda J(W)$, where $J(W)$ is the orthogonality regularization cost and $\lambda$ is its associated weight for the given layer at
the given iteration [25] [26] [27]. The most well-known of these costs is

$$J(W) = \|W^T W - I\|^2_F = \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} (w_{ki}w_{kj} - \delta_{ij})^2,$$

although a number of variations on this cost have been considered. The augmented cost is then minimized via gradient descent. The regularization parameter $\lambda$ is often carefully tuned, and is typically reduced during adaptation.

One issue in the implementation of these regularization methods is computational cost. Evaluating $J(W)$ requires $0.5m^2n + \mathcal{O}(mn)$ multiplies, which can become burdensome when done as part of each (mini)batch update. The gradient of this cost is

$$\frac{\partial J(W)}{\partial W} = 4W(W^T W - I),$$

which requires $1.5m^2n + \mathcal{O}(mn)$ multiplies. A desirable approach to this task would reduce the complexity of the cost or gradient calculation to one that is $\mathcal{O}(mn)$, i.e. linear in the number of parameters. This method was discussed in [13].

### 2.2. Orthogonality Applied to CNNs

Recently, a work has focused on imposing orthogonality in CNNs and evaluated its effect by making the filters, the matrices of hidden convolution layers, orthogonal.

In this case, consider that we have a $(m \times m \times p \times q)$ convolutional weight matrix $W_0$. Due to its multi-dimensions and non-square size, the straightforward method is to reshape $W_0$ to an $((m \times m \times p) \times q)$ matrix $W$ and attempt to impose

$$W^T W = I$$

In this case, $W$ is a undercomplete matrix, such that $(q \leq (m \times m \times p))$. It is impossible for an overcomplete($(m \times m \times p) \leq q$) $W$ to be close to identity because of a lack of degrees of freedom.

Inspired by SVD discussed in the last section, setting a boundary on singular value of convolution filters is one of possible way to achieve orthogonality [28].

Harand [29] proposed a method to impose orthogonality only on the fully connected layer of CNNs and tested it on Visual Geometry Group network(VGGnet), but the result
is not very good. Inspired by the property of Stiefel manifold, Huang [27] formulated an Optimization over Multiple Dependent Stiefel Manifolds (OMDSM) and enforced the weight filters across channels in convolutional layers to be orthogonal. However, this approach did not improve the performance of the CNN and lead to a slow training resulting from many SVD computations.

In [30], an approximation of the reshaped weight $W$ is used through a singular value study. The original 4-d weight matrices were reshaped to 2-d weight matrices and the largest singular value was used for their orthogonal regularization.

The above method for specifying the matrix $W$ for CNNs does not take into account the operator norm of the linear transformation associated with the filters in the matrix $W$ in each layer. In [28], the authors consider this linear transformation, expanding each filter to the same size as the that of the input features via Fourier transformation. Then, they compute the singular values of the filters. Their work shows that a well-performing CNN does not impose orthogonality on the filter operations. Instead, singular values are bounded but not restricted to values near unity.

Figure 2.1. Plot of the singular values of the linear operators associated with the convolutional layers of the pretrained “ResNetV2” from the TensorFlow website.

Figure 2.1 shows the singular values of 4-d weight filters from some chosen convolutional
Figure 2.2. Plot of the ratio of singular values to the maximum singular value of the linear operators associated with the convolution layers of the pretrained “ResNet V2” normalized by size of the convolution.

layers (i.e. conv2d-6 refers to the sixth convolutional layer), which were computed using a Fourier transform technique for a particular CNN architecture trained to a high accuracy. In Figure 2.2, the singular values are replaced by the ratio of $\sigma_i$ to the maximum singular value $\sigma_{max}$. These figures show that the operations of each layer are close to orthogonal but are not strictly orthogonal. Except for the first convolution layer, singular values from the remaining layers form a cluster that is smoothly distributed between zero and one.

Thus, the matrix $W$ has singular values that do not impose strict orthogonality. Except for the first convolution layer, distribution curves of singular values of every convolution layer are smooth. So [28] built an algorithm to bound the singular values and got an inspiring result. That means, all we need may not be a strict orthogonality but rather an adaptive singular value regularization.

2.3. Applications of Orthogonality in Machine Learning

For neural networks, orthogonal transformations enforce the singular spectrum to be unitary which is norm-preserving [31]. Some orthogonal initialization strategies already
showed advantages in neural network training [32]. The benefit of imposing orthogonality have been proved on RNN and DNN [33]. Recently, researchers have been trying to improve CNN by imposing orthogonal constraints, normalization and regularization.

Orthogonal matrices have been actively explored in Recurrent Neural Networks (RNNs). [33], [17], [34]. Orthogonality helps to avoid the gradient vanishing and explosion problems in RNNs due to its energy preservation property [35]. However, the orthogonal matrix here is limited to be square for the hidden-to-hidden transformations in RNNs. The more general setting of the learning of orthogonal rectangular matrix is not well studied in DNNs [29], especially in deep Convolutional Neural Networks [26].

Recently several works [29] [25] [13] have shown that imposing orthogonal regularization or constraints throughout training lead to an encouraging result. In [8], an initialization method which pre-initialize weights of each convolution layer with orthonormal matrices is proposed. In [25], an algorithm constraining the weight matrices in the orthogonal feasible set during the entire whole process of network training is proposed, which is achieved by singular value bounding.

Bansal et al. [13] proposed an orthonormal regularization to improve the accuracy and training speed of deep ResNet. The researchers added a regularization to the loss function of CNNs and through training the loss function would be minimized such that orthogonality is imposed to the system. They tested their orthogonality regularization on ImageNet, CIFAR-10, CIFAR-100 and observe the constraints and remarkable accuracy boost (e.g.: 2.31 percent for top-1 accuracy on CIFAR).

Most recently, [36] introduced the algorithms of Orthogonal Deep Neural Networks (OrthDNNs) to connect with recent interest of spectrally regularized deep learning methods. They prove that the optimal bound with respect to the degree of isometry is attained when each weight matrix has a spectrum of equal singular values. OrthDNNs with weight matrices of orthonormal rows or columns are thus the most straightforward choice. Based on such analysis, they presented algorithms of strict and approximate OrthDNNs, and propose a simple yet effective algorithm called Singular Value Bounding. They also proposed Bounded Batch Normalization, which is a technique to improve accuracy and speed up training, to
make compatible use of batch normalization with OrthDNNs. Experiments on benchmark image classification show the efficiency and robustness of OrthDNNs.

However, a completed examination on the effect of orthogonality to CNNs and reasonable theory remains missing.

This thesis shows a new adaptive orthogonal regularization method, stochastic orthogonalization, which is inspired by a well-know orthogonalization algorithm in signal processing, could improve the performance of CNNs. It maintains the high accuracy while requiring less training time each iteration.

2.4. Other Orthogonal Regularization

Bansal et al [13] proposed a good orthogonality implementation method which can be incorporated with other advanced strategies such as He-normalization or batch-normalization, in almost any architecture of CNNs. In their paper, three means of implementing orthogonality were discussed and tested on three state-of-the-art of CNNs: ResNet, ResNeXt and WideResNet. Remarkable accuracy improvement on accuracy was observed with a fast and stable convergence. This method can enforce orthogonality on non-squared weight matrices without changing the architecture of the original CNN. Their work show that orthogonality still provides significant potential to improve the performance of CNNs.

Among three type of orthogonal constraints they discussed, the Spectrum Restricted Isometry Property (SRIP) based method works best for large and deeper CNNs. The mechanism [13] uses to impose orthogonality on the weight matrix of a convolutional layer is to add a term \( \sigma(W^T W - I) \) to the original loss function of the CNN, where \( W \) is the reshaped weight matrix. Assume the weight matrix \( M \in R^{N \times N \times I \times O} \) is a 4-dimensional matrix, where \( N, I, O \) denotes the weights length( weights heights usually equals weight length in a convolutional layer), number of input channels, number of output channels.

Then \( M \) is reshaped to \( W \) with shape \( m \) by \( n \), where \( m = N \times N \times I \) and \( n = O \).

The RIP condition assumes the following:

For all vector \( z \in R^n \) that is \( k \)-sparse, there exists a small \( \delta w \in (0, 1) \) s.t \( (1 - \delta w) \leq \frac{\|Wz\|}{\|z\|} \)

With every set of columns of weight matrices \( W \) no larger than \( k \), it shall behave like a orthogonal system.
Especially when \( k = n \), the entire \( \mathbf{W} \) would be close to orthogonal according to RIP. In this case, the RIP equation we mentioned above could be rewritten as

\[
\left| \frac{\| \mathbf{W} \mathbf{z} \|}{\| \mathbf{z} \|} - 1 \right| \leq \delta \mathbf{w}
\]  

(2.16)

They defined the spectral norm of weight matrices \( \mathbf{W} \) as \( \sigma(\mathbf{W}) = \frac{\| \mathbf{W} \mathbf{z} \|}{\| \mathbf{z} \|} \). So the spectral norm of the term \( \mathbf{W}^T \mathbf{W} - \mathbf{I} \) is \( \sigma(\mathbf{W}^T \mathbf{W} - \mathbf{I}) = |\frac{\| \mathbf{W} \mathbf{z} \|}{\| \mathbf{z} \|} - 1| \). Notice that when \( \mathbf{W} \) is approaching orthogonality, \( \sigma(\mathbf{W}^T \mathbf{W} - \mathbf{I}) \) is also approaching zero, so it is claimed that enforcing orthogonality mathematically is equivalent to minimizing the spectral norm of \( \mathbf{W}^T \mathbf{W} - \mathbf{I} \):

\[
(SRIP) : \sigma(\mathbf{W}^T \mathbf{W} - \mathbf{I}) \leq \delta \mathbf{w}
\]  

(2.17)

The above term is added to the original loss function of CNN and through the training the loss of entire CNN along with \( \sigma(\mathbf{W}^T \mathbf{W} - \mathbf{I}) \) is minimized, so the system approaches orthogonality.

They also introduced a mechanism to reduce the computation of the gradient of SRIP by initializing a random uniform distributed vector \( \mathbf{v} \in \mathbb{R}^n \) and then use the following procedure:

\[
\mathbf{u}_0 = (\mathbf{W}^T \mathbf{W} - \mathbf{I}) \mathbf{v}
\]  

(2.18)

\[
\mathbf{v}_1 = (\mathbf{W}^T \mathbf{W} - \mathbf{I}) \mathbf{u}_0
\]  

(2.19)

\[
\sigma(\mathbf{W}^T \mathbf{W} - \mathbf{I}) \approx \frac{\| \mathbf{v}_1 \|}{\| \mathbf{u}_0 \|} = \frac{\| (\mathbf{W}^T \mathbf{W} - \mathbf{I}) (\mathbf{W}^T \mathbf{W} - \mathbf{I}) \mathbf{v} \|}{\| (\mathbf{W}^T \mathbf{W} - \mathbf{I}) \mathbf{v} \|}
\]  

(2.20)

Then, an additional term is added to the loss function of CNN:

\[
\sigma(\mathbf{W}^T \mathbf{W} - \mathbf{I}) \approx \frac{\| \mathbf{v}_1 \|}{\| \mathbf{u}_0 \|} = \frac{\| (\mathbf{W}^T \mathbf{W} - \mathbf{I}) (\mathbf{W}^T \mathbf{W} - \mathbf{I}) \mathbf{v} \|}{\| (\mathbf{W}^T \mathbf{W} - \mathbf{I}) \mathbf{v} \|} = \frac{g(\mathbf{W})^{0.5}}{h(\mathbf{W})^{0.5}}
\]  

(2.21)

The gradient of their additional cost is given by:
\[
\frac{\partial g(W)}{\partial W} = (2WW^TWW^TWv - 8WW^TWv + 12WW^Tv)v^T + (2WW^TWW^Tv - 8WW^Tv + 12Wv)v^TWW^T + 2Wvv^TWW^TW - 8Wvv^T
\]

\[
\frac{\partial h(W)}{\partial W} = 2(WW^TWvv^T + Wvv^TWW^T - 4Wvv^T)
\] (2.23)

\[
\frac{\partial \sigma(W^T - I)}{\partial W} \approx 0.5h(W)\left(\frac{\partial g(W)}{\partial W}/g(W) - \frac{\partial h(W)}{\partial W}g(W)/h(W)^2\right)
\] (2.24)

The gradient computational cost is at least $12mn$ per iteration, where $m$ and $n$ are the number of rows and columns of each weight matrix ($n \leq m$).

The insertion of an additional term $J(W) = \sigma(W^TW - I)$ into the loss function aims at making the reshaped weights $W$ orthogonal ($W^TW = I$). This algorithm does improve the performance of CNNs to some degree, but we need to compute many higher-order terms of a large matrix $W$ which is inefficient in comparison with our stochastic orthogonalization.
CHAPTER 3
STOCHASTIC ORTHOGONALIZATION

3.1. Introduction

In this chapter, we describe a set of algorithms that perform stochastic orthogonalization of the columns of a matrix \( W \). These algorithms minimize a stochastic version of \( J(W) \) using \( m \)-dimensional uncorrelated signals as part of the processing. In these algorithms, the inner products of the columns of \( W \) are never computed, the lengths of each of the columns are never computed, and weights are never scaled by any multiplicative factor. In addition, the computational complexity of each algorithm is linear in the number of entries of \( W \) and is between \( 3mn + \mathcal{O}(m) \) and \( 5mn + \mathcal{O}(m) \) multiplications at each iteration. The theory of stochastic orthogonalization is carefully described, and stability issues regarding the choice of step size or regularization parameter are considered. Performance comparison of our proposed method with competing approaches show the viability of the techniques in deep learning classification tasks.

3.2. Derivations and Analysis

Three proposed algorithms for stochastic orthogonalization are derived in this section. We employ a simplified notation whereby all time indices are suppressed. Furthermore, gradient descent of the matrix \( W \) on a specified cost \( J(W) \) is represented as

\[
W_{\text{new}} = W - \gamma \frac{\partial J(W)}{\partial W},
\]

(3.1)

where \( W_{\text{new}} \) are the updated parameters and \( \gamma \) is a chosen step size. With the above notation, gradient descent on (2.14) with \( \mu = \gamma/4 \) yields the well-known iterative Newton-based algorithm

\[
W_{\text{new}} = W + \mu (W - WW^T W).
\]

(3.2)
We summarize three existing results about (3.2) below.

1. For \( \mu = 0.5 \), the algorithm was originally described in [37]. It is an approximate Newton’s method for this parameter choice and has fast convergence.

2. This algorithm only changes the singular values of \( W \); the left- and right-singular vectors of \( W \) remain unchanged throughout adaptation [19].

3. Let the maximum singular value of the initial value of \( W \) be \( \sigma_{\text{max}} \). If
\[
\sigma_{\text{max}} < \sqrt{\frac{2 + \mu}{\mu}},
\]
this algorithm causes \( W^T W \) to converge to \( I \) according to the decoupled nonlinear equations for each singular value of \( W \) given by
\[
\sigma_{i,\text{new}} = \left| 1 + \mu (1 - \sigma_i^2) \right| \sigma_i.
\] (3.4)

All algorithms in this chapter approximate (3.2) without direct evaluation of \( J(W) \) or its gradient. These algorithms employ an \( m \)-dimensional vector signal sequence \( x \) with uncorrelated elements \( x_i \), such that
\[
E\{x_i\} = 0 \quad \text{and} \quad E\{xx^T\} = I,
\] (3.5)
where \( E\{\cdot\} \) denotes statistical expectation. Now, for any \((m \times m)\) matrix \( A \) with diagonal entries \( a_{ii} \), \( 1 \leq i \leq m \),
\[
E\{x^T A x\} = \text{tr}[A] = \sum_{i=1}^{m} a_{ii}.
\] (3.6)
Therefore, let us define
\[
A = (W^T W - I)(W^T W - I),
\] (3.7)
such that
\[
E\{x^T A x\} = \text{tr}[(W^T W - I)(W^T W - I)] = J(W)
\] (3.8)
Therefore, an equivalent cost to (2.14) above is \( J(W) = E\{x^T A x\} \). Expanding the expression, we have
\[
x^T A x = x^T W^T W x - 2x^T W W^T W x + x^T x
\]
\[
= \|u\|^2 - 2\|y\|^2 + \|x\|^2
\]
where we have defined

$$y = Wx,$$  \hspace{1cm} (3.9)

$$u = W^T y = W^T Wx,$$  \hspace{1cm} (3.10)

$$v = Wu = W W^T Wx.$$  \hspace{1cm} (3.11)

Then, we have that

$$\mathcal{J}(W) = E\{||u||^2 - 2||y||^2 + ||x||^2\}.$$  \hspace{1cm} (3.12)

Eqn. (3.12) is a cost function identical to that in (2.14) that depends on the first- and second-order statistics of $x$.

A gradient descent algorithm minimizing the cost in (3.12) can be derived by noting that

$$\frac{1}{2} \frac{\partial \{||u||^2 - 2||y||^2 + ||x||^2\}}{\partial W} = W xx^T W^T W + WW^T W xx^T - 2 W xx^T$$

$$= y[u - x]^T + [v - y]x^T.$$  

Therefore, this gradient algorithm is

$$W_{new} = W + \frac{\mu}{2} E\{y[x - u]^T + [y - v]x^T\}.$$  \hspace{1cm} (3.13)

This algorithm depends on the statistics of $x$, not on the sequence $x$ itself.

To obtain data-dependent algorithms, we employ the classic approximation used in stochastic gradient adaptation by dropping the expectation from the above relation. Before giving the algorithm forms, we also note via a simple analysis that

$$E\{y[x - u]^T\} = W E\{xx^T\}(I - W^T W)$$

$$= W(I - W^T W).$$  \hspace{1cm} (3.14)

$$E\{[y - v]x^T\} = W(I - W^T W)E\{xx^T\}$$

$$= W(I - W^T W).$$  \hspace{1cm} (3.15)

Thus, we can describe three algorithms that approximately minimize $\mathcal{J}(W)$ in (3.12), each with slightly different algorithmic complexities (denoted in brackets) and behaviors. Note that $y$, $u$, and $v$ are computed via (3.9)–(3.11) above, where needed.
Algorithm 1 \([5mn + O(m)]\):

\[
W_{new} = W + \frac{\mu}{2} (y[x - u]^T + [y - v]x^T).
\]

Algorithm 2 \([3mn + O(m)]\):

\[
W_{new} = W + \mu (y[x - u]^T).
\]

Algorithm 3 \([4mn + O(m)]\):

\[
W_{new} = W + \mu ([y - v]x^T).
\]

Remark #1: The three algorithms above preserve the linear span of \(W\) in \(W_{new}\). However, unlike (3.2), these algorithms do not maintain the left- and right-singular vectors of \(W\). As \(\mu\) is reduced, the deviations between the behaviors of the above algorithms and (3.2) are reduced.

Remark #2: It is possible to perform an analysis of each update to determine step size bounds that maintain stability. We shall focus on Algorithm 2 due to its reduced complexity.

We can show after some algebra that Algorithm 2 admits the following update relation on \(J(W)\).

First, we take Algorithm 2 and define \(y\) and \(u\) as

\[
W_{new} = W + \mu y[x - u]^T
\]

\[
y = Wx
\]

\[
u = W^Ty = W^TWx
\]

Then, we square the \(W_{new}^TW_{new} - I\) matrix and take its trace.

\[
W_{new}^TW_{new} - I = W^TW - I + \mu (u[x - u]^T + [x - u]u^T) + \mu^2||y||^2[x - u][x - u]^T
\]

(3.20)
\[
\text{tr}(W_{\text{new}}^{T}W_{\text{new}} - I)^2 = \text{tr}(W^{T}W - I)^2 \\
\quad + \mu^2 \text{tr}([u[x - u]^T + [x - u]u^T] (u[x - u]^T + [x - u]u^T)) \\
\quad + \mu^4 \|y\|^4 \text{tr}([x - u][x - u]^T[x - u][x - u]^T) \\
\quad + 2\mu \text{tr}([W^T W - I] (u[x - u]^T + [x - u]u^T)) \\
\quad + 2\mu^2 \|y\|^2 \text{tr}([W^T W - I][x - u][x - u]^T) \\
\quad + 2\mu^3 \|y\|^2 \text{tr}([u[x - u]^T + [x - u]u^T] [x - u][x - u]^T)
\]

We define:
\[
e = x - u.
\]

Then, the trace becomes:
\[
\text{tr}(W_{\text{new}}^{T}W_{\text{new}} - I)^2 = \text{tr}(W^{T}W - I)^2 \\
\quad + \mu^2 \text{tr}([ue^T + eu^T] (ue^T + eu^T)) \\
\quad + \mu^4 \|y\|^4 \text{tr}[ee^T ee^T] \\
\quad + 2\mu \text{tr}([W^T W - I] (ue^T + eu^T)) \\
\quad + 2\mu^2 \|y\|^2 \text{tr}([W^T W - I] ee^T) \\
\quad + 2\mu^3 \|y\|^2 \text{tr}([ue^T + eu^T] ee^T)
\]
Now, since the common cost function uses squared error we set:

\[ ||e||^2 = J(W) \]
\[ u^T e = u^T [x - u] \]
\[ = ||y||^2 - ||u||^2 \]
\[ u^T W^T W [x - u] \]
\[ = ||u||^2 - ||v||^2 \]
\[ e^T W^T W [x - u] \]
\[ = ||y||^2 - 2||u||^2 + ||v||^2. \]
Then, we simplify the trace equation and replace $||e||^2$ with $J(W)$

$$\text{tr}[(W_{\text{new}}^T W_{\text{new}} - I)^2] = \text{tr}[(W^T W - I)^2]$$

$$+ 2\mu^2 \left[ ||u||^2 ||e|| + (||y||^2 - ||u||^2) \right]$$

$$+ \mu^4 ||y||^4 ||e||^4$$

$$- 4\mu \left[ ||y||^2 - 2||u||^2 + ||v||^2 \right]$$

$$+ 2\mu^2 ||y||^2 \left[ ||y||^2 - 2||u||^2 + ||v||^2 - ||e||^2 \right]$$

$$+ 4\mu^4 ||y||^2 ||e||^2 \left[ ||y||^2 - ||u||^2 \right]$$

Notice that:

$$||y||^2 - 2||u||^2 + ||v||^2 = x^T(WW^T - I)(WW^T - I)x \geq 0 \quad (3.23)$$

which means the square trace is decreasing since $\mu$ is positive.

Consider that the step size $\mu$ used in this iterative algorithm is always chosen to be small, so the higher order terms $\mu$ can be ignored.

$$J(W_{\text{new}}) = J(W) - 4\mu ||W(W^T W - I)x||^2 + O(\mu^2), \quad (3.24)$$

where the full relation involves a fourth-order polynomial in $\mu$. While the exact form of this polynomial does not provide much insight, we can see for small $\mu > 0$ and $x$ satisfying $||x - u||^2 > 0$ that $J(W)$ decreases at each iteration. Moreover, to maintain similar scaling of the second term in (3.24) to that of $J(W)$, it is beneficial to choose a normalized step size of the form

$$\mu = \frac{\hat{\mu}}{||y||^2} \quad (3.25)$$
To show convergence similarities and differences between the proposed stochastic gradient algorithms and the original Newton-based approach, Figure 3.1 shows the evolutions of the average values of $J(W)$ as computed over 100 simulations with $m = 32$ and $n = 288$, where five different algorithm setups are compared: the three stochastic gradient algorithms, SRIP algorithm [13] introduced in Chapter 2, and two versions of (3.2) with differing step sizes. In this case, a normalized step size with $\hat{\mu} = 0.1$ was chosen for the proposed methods. As can be seen, all three proposed algorithms converge similarly and much faster than SRIP, and thus Algorithm 2 is to be preferred due to its reduced complexity. Moreover, the stochastic gradient algorithms provide similar convergence to that of (3.2) when a small step size for (3.2) is selected. Eqn. (3.2) achieves much faster convergence than can be attained by any one of (3.16), (3.17), (3.18) when $\mu$ is large, as the stochastic methods require data averaging through a small step size. A similar performance relationship
can be found between least-squares methods and the least-mean-square algorithm in linear adaptive filtering [38] and is to be expected. The overall advantage of the stochastic gradient methods is the combination of computational simplicity and adequate adaptation capabilities to achieve the desired performance characteristics.
4.1. Description of Experiments

To demonstrate the capabilities of stochastic orthogonalization in deep learning tasks, we apply the method to ResNet, a popular convolutional neural network architecture, to evaluate its performance on the CIFAR-10 and CIFAR-100 image classification tasks.

**CIFAR-10**: The CIFAR-10 dataset consists of 60,000 $32 \times 32$ color images of 10 object categories (50,000 training and 10,000 testing ones). We use raw images without preprocessing. Data augmentation follows the standard manner in [39]: during training, we zero-pad 4 pixels along each image side, and sample a $32 \times 32$ region crop from the padded image or its horizontal flip; during testing, we use the original non-padded image.

**CIFAR-100**: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a ”fine” label (the class to which it belongs) and a ”coarse” label (the superclass to which it belongs).

In each case, we employ the original training protocols in all pre-processing steps, including data augmentation, training/validation/testing data splitting, and common step size schedules.

We use ResNet as our testing architecture. Each network starts with a convolutional layer of 16 $(3 \times 3)$ filters, and then sequentially stacks three types of convolutional layers of $(3 \times 3)$ filters, each of which has the feature map sizes of 32, 16, and 8, and filter numbers of 16, 32, and 64, respectively; spatial sub-sampling of feature maps is achieved by convolutional layers of stride 2; the network ends with a global average pooling and a fully-connected layer [8].

ResNet has a different structure in comparison with common CNNs, such as:

**Shortcut**: A shortcut connection is inserted which turn the network into its counterpart
residual version. The identity shortcuts $F(x) + x$ can be directly used when the input and output are of the same dimensions. When the dimensions increase, it considers two options:

1. The shortcut performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no additional parameter.
2. The projection shortcut is used to match dimensions (done by $1 \times 1$ convolutions).

Figure 4.1. Residual learning: a building block [8].

**Bottleneck block:** We define a bottleneck architecture as the type found in the ResNet paper where two ($3 \times 3$) convolutional layers are replaced by one ($1 \times 1$) convolutional, one ($3 \times 3$) convolutional, and another ($1 \times 1$) convolutional layer. The ($1 \times 1$) convolutional layers are used as a form of dimension reduction and restoration.

There are two sub-versions of Inception ResNet, namely v1 and v2.

For residual addition to work, the input and output after convolution must have the same dimensions. Hence, we use ($1 \times 1$) convolutions after the original convolutions, to match the depth sizes, where depth is increased after convolution. The pooling operation inside the main inception modules were replaced in favor of the residual connections. The prominent changes between ResNet v1 and v2 are: (a) The use of a stack of ($1 \times 1$) - ($3 \times 3$) - ($1 \times 1$) BN-ReLU-Conv2D (b) Batch normalization and ReLU activation come before 2D convolution.

We choose the ResNet-110 v2 architecture with bottleneck residual units [8] [40], where we have selected $p = 9n + 2$ with $n = 12$ so $p = 110$. The Adam optimizer [41] is used for training the network over 200 epochs with a step size schedule of 0.01 decreasing to 0.001, $10^{-5}$, and $10^{-6}$ at 80, 120, and 160 epochs, respectively. Dropout was not used.
We compare the performance of stochastic orthogonalization to the SRIP regularization method described in [13] along with the original unregularized network. The regularization parameter $\lambda$ for both the proposed and SRIP methods was the same and selected as $0.01$ decreasing to $10^{-5}$, $10^{-6}$, $10^{-8}$, and 0 after 20, 50, 70, and 120 epochs, respectively. In this way, the methods influence the convergence behavior while not limiting the final performance of the architecture at convergence.

Applying **Batch normalization(BN)**: Batch normalization is a technique for improving the speed, performance, and stability of artificial neural networks. Batch normalization was introduced in [42]. It is used to normalize the input layer by adjusting and scaling the activations. Once implemented, batch normalization has the effect of dramatically accelerating the training process of a neural network, and in some cases improves the performance of the model via a modest regularization effect [42].

Applying **Adam Optimizer**: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. Adam realizes the benefits of both AdaGrad and RM-
SProp [43] [44]. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance) [41].

**Applying Mini-batch gradient descent**: Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients. Implementations may choose to sum the gradient over the mini-batch which further reduces the variance of the gradient. Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima. The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

![Figure 4.3](image1)

![Figure 4.3](image2)

Figure 4.3. Convergence curves of ResNet-110 on CIFAR-10, averaged over 5 runs with the same initialization. See the text for the learning rate schedule.
Figure 4.4. Convergence of ResNet-110 on CIFAR-100, averaged over 5 runs with the same initialization. See the text for the learning rate schedule.

4.2. Results

Shown in Figure 4.3 and 4.4 are the convergence of the accuracies of the various classifiers on CIFAR-10 and CIFAR-100, respectively. As can be seen, stochastic orthogonalization provides similar performance as SRIP, and both outperform the architecture with no regularization. Table 1 shows the average error rates for the three architectures, where it is seen that stochastic orthogonalization’s performance is competitive with that of the SRIP method. Also shown in the table is the percentage overhead (measured by executing times) associated with the regularization procedure on the SMU M2 GPU computer cluster, where it is seen that stochastic orthogonalization involves less overhead as compared to SRIP due to the former’s straightforward implementation. Thus, our proposed techniques are simpler to implement and provide competitive performance to other state-of-the-art schemes.

Figures 4.5 and 4.6 show the overall benefit of adaptive regularization via stochastic
<table>
<thead>
<tr>
<th>Regularization</th>
<th>Error Rate (%)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>CIFAR-100</td>
</tr>
<tr>
<td>None</td>
<td>7.10</td>
<td>26.40</td>
</tr>
<tr>
<td>SRIP</td>
<td>5.76</td>
<td>25.60</td>
</tr>
<tr>
<td>Stoch. Orth.</td>
<td>5.74</td>
<td>25.50</td>
</tr>
</tbody>
</table>

Table 4.1. Error rates and overhead (%) for ResNet-110 architectures with and without regularization.

Figure 4.5. Singular value distributions for ResNet-110 with no regularization applied to CIFAR-10.

orthogonalization. These plots show the singular value ratio defined as $\sigma_i/\sigma_{\text{max}}$ for selected weight matrices $W$ throughout the ResNet-110 architecture applied to CIFAR-10, where layers closer to the input are plotted with colors with a greater share of blue. As can be seen, the layers nearest the output in the unregularized system have weight matrices that are rank-deficient, whereas the system with stochastic orthogonalization regularization has weight matrices that do not suffer this rank deficiency throughout the architecture. It has been observed that singular value ratio distributions similar to those shown in Figure 4.6 are an indicator of improved performance in deep learning classification tasks [28].
Figure 4.6. Singular value distributions for ResNet-110 with stochastic orthogonalization regularization applied to CIFAR-10.
CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1. Conclusion

In this thesis, we have introduced stochastic orthogonalization as a computationally-
simple way to impose regularization on the distributions of the singular values of weight
matrices in convolutional neural networks. The technique appears to work like singular-
value bounding in that it raises low singular values and decreases high singular values.
Unlike other approaches which focus on imposing orthogonality, the methods in this thesis
cause the weight matrices to have a smooth singular value distribution but are not all equal
to unity.

In addition, the methods proposed in this thesis have a low computational cost. Thus,
they are well-suited to deep neural network architectures.

5.2. Future Work

Future work in this area could include the following:

The methods could be tested on a wider range of CNN architectures, such as Wide-
ResNet and ResNext [13]. In some cases, these architectures have a large increase in channels
from layer to layer, causing the weight matrices to be overcomplete when considered in
traditional form. Thus, a modification to stochastic orthogonalization would be needed.

The methods for directly updating the $W$ matrices within each structure could be used
in place of the cost function approach, such that Algorithms 2 and 3 could be tested and
explored for use within CNNs.

Optimization of the weight decay coefficient to improve performance further could be
performed.
References


