UNLV Theses, Dissertations, Professional Papers, and Capstones

May 2019

A Novel Feature Maps Covariance Minimization Approach for Advancing Convolutional Neural Network Performance

Bikram Basnet
basnetbik@gmail.com

Follow this and additional works at: https://digitalscholarship.unlv.edu/thesesdissertations

Part of the Computer Sciences Commons

Repository Citation

Basnet, Bikram, "A Novel Feature Maps Covariance Minimization Approach for Advancing Convolutional Neural Network Performance" (2019). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 3569.
https://digitalscholarship.unlv.edu/thesesdissertations/3569

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

A NOVEL FEATURE MAPS COVARIANCE MINIMIZATION APPROACH FOR ADVANCING CONVOLUTIONAL NEURAL NETWORK PERFORMANCE

By

Bikram Basnet

Bachelor in Computer Engineering (B.E.)
Institute of Engineering Central Campus, Tribhuvan University, Nepal
2014

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2019

© Bikram Basnet, 2019

All Rights Reserved

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 11, 2019

This thesis prepared by

Bikram Basnet

entitled

A Novel Feature Maps Covariance Minimization Approach for Advancing Convolutional Neural Network Performance

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Justin Zhan, Ph.D.
Examination Committee Chair

Laxmi Gewali, Ph.D.
Examination Committee Member

Wolfgang Bein, Ph.D.
Examination Committee Member

Ge Kan, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Dean

Abstract

We present a method for boosting the performance of the Convolutional Neural Network (CNN) by reducing the covariance between the feature maps of the convolutional layers.

In a CNN, the units of a hidden layer are segmented into the feature/activation maps. The units within a feature map share the weight matrix (filter), or in simple terms look for the same feature. A feature map is the output of one filter applied to the previous layer. CNN search for features such as straight lines, and as these features are spotted, they get reported to the feature map. During the learning process, the convolutional neural network defines what it perceives as important. Each feature map is looking for something else: one feature map could be looking for horizontal lines while the other for vertical lines or curves. Reducing the covariance between the feature maps of a convolutional layer maximizes the variance between the feature maps out of that layer. This supplements the decrement in the redundancy of the feature maps and consequently maximizes the information represented by the feature maps.

Acknowledgements

"I would like to extend my deep gratitude and humble sincerity towards the University of Nevada in Las Vegas, and the Department of Computer Science, along with the thesis supervisor Dr. Justin Zhan, and all the other committee members under the assistance of whom I got to further broaden my knowledge. It was my great pleasure to complete my graduate studies in my field of interest from The University of Nevada, Las Vegas. I am thankful for providing me research opportunities that elucidated my interest and career objectives. Special appreciation to Coursera, and professor Andrew Ng.

I would also like to place heartful thanks to all my well wishers."

Bikram Basnet

University of Nevada, Las Vegas
May 2019

Table of Contents

Copyright i

Abstract iii

Acknowledgements iv

Table of Contents v

List of Tables viii

List of Figures x

List of Algorithms xii

Chapter 1 Introduction 1

1.1 Background 1

1.2 Objectives and Summary of Approach 4

1.3 Organization of the Thesis 5

Chapter 2 Related Work 6

2.1 First Wave: Cybernetics 8

2.2 Second Wave: Connectionism 9

2.3 Third Wave: Deep Learning 10

Chapter 3 Proposed Approach 13

3.1 Covariance and Correlation 13

3.2 Correlation Matrix and Convolution Matrix 15

3.3 Proof of Principal Component Analysis 16

3.3.1 Introduction 16

3.3.2 Statement 17

3.3.3 Proof by Induction 17

3.4 Feature Maps Covariance Minimization 20

3.5 Algorithm/Proposed Model 22

Chapter 4 Experimental Evaluation For Image Classification Task 23

4.1 Overall Settings 23

4.2 Data Description 23

4.3 Experiments 24

4.3.1 MNIST 24

4.3.2 Fashion MNIST 26

4.3.3 CIFAR-10 27

4.3.4 CIFAR-100 29

4.4 Discussion 30

4.5 Further Improvements 30

4.5.1 ZCA Whitening/Sphering 30

4.5.2 Pair Formation (for Covar Minimization) with Replacement 30

4.5.3 Using Additional Pair-wise Networks 30

4.5.4 Voting Among the Networks 31

Chapter 5 Experimental Evaluation For Enhancing Image Resolution 32

5.1 Overall Settings 32

5.2 Data Description 32

5.3 Experiments 33

5.3.1 MNIST 33

5.3.2 Fashion MNIST 34

5.3.3 CIFAR-10 35

5.3.4 CIFAR-100 36

5.4 Further Improvements 37

5.4.1 ZCA Whitening/Sphering 37

5.4.2 Increasing the Number of Networks 37

Chapter 6 Conclusion **38**

Appendix A Artificial Neural Network **40**

A.1 Backpropagation Algorithm 53

A.1.1 Forward Propagation 53

A.1.2 Backward Propagation 53

A.2 Proof by Induction 54

A.2.1 Output/Final layer 54

A.2.2 Base Step (last hidden layer) 57

A.2.3 Inductive Hypothesis ($(k+1)^{th}$ layer) 58

A.2.4 Inductive Step ($(k)^{th}$ layer) 58

A.2.5 Summary 59

Appendix B Representation Learning **60**

B.1 Neural Network as a 1-vs-all Model 60

B.2 Neural Network as Image Generator 61

Bibliography **62**

Curriculum Vitae **67**

List of Tables

4.1 The top-1 error rate comparison of different models on the MNIST dataset for first 35 epochs. 24
4.2 Confusion matrix of baseline 25
4.3 Confusion matrix of best model (dropout+covar) 25
4.4 The top-1 error rate comparison of different models on the fashion MNIST dataset for first 35 epochs. 26
4.5 Confusion matrix of baseline 27
4.6 Confusion matrix of best model (dropconnect+covar) 27
4.7 The top-1 error rate comparison of different models on the CIFAR-10 dataset for the first 35 epochs. 28
4.8 The top-1 error rate comparison of different models on the CIFAR-100 dataset for the first 35 epochs. 29

5.1 Mean Squared Error (MSE) comparison of different models on the MNIST dataset for first 35 epochs. 33
5.2 Mean Squared Error (MSE) comparison of different models on the fashion MNIST dataset for first 35 epochs. 34
5.3 Mean Squared Error (MSE) comparison of different models on the CIFAR-10 dataset for first 35 epochs. 35
5.4 Mean Squared Error (MSE) comparison of different models on the CIFAR-100 dataset for first 35 epochs. 36

A.1 Fisher Iris Setosa Dataset. 43
A.2 Fisher Iris: Versicolor/Virginica dataset 46
A.3 Dataset for boolean 'XOR' 47

A.4 Dataset for boolean 'XOR' and 'AND' 50

List of Figures

1.1 With the assessment of the sequence of transformations, each manifold becomes compact, while different manifolds are distant. For example, chair manifold (blue) and non-chair manifold (black)[25] 2
1.2 Two spheres viewed from the front and top view; same colored (1 and 3), and different colored (2 and 4) spheres. 3
1.3 Example of different representations: cartesian coordinate system vs polar coordinate system. 3

3.1 Backpropagation with feature maps covariance minimization. 21

4.1 Performance comparison of different models on the MNIST test dataset. 24
4.2 Performance comparison of different models on the fashion MNIST test dataset. 26
4.3 Performance comparison of different models on the CIFAR-10 test dataset. 28
4.4 Performance comparison of different models on the CIFAR-100 test dataset. 29

5.1 Performance comparison of different models on the MNIST test dataset. 33
5.2 Performance comparison of different models on the fashion MNIST test dataset. 34
5.3 Performance comparison of different models on the fashion CIFAR-10 test dataset. . . . 35
5.4 Performance comparison of different models on the fashion CIFAR-100 test dataset. . . 36

A.1 Working of single neuron/perceptron, or a logit unit with D=3 40
A.2 Linear regression with fisher iris setosa dataset. 43
A.3 Fisher Iris: Versicolor/Virginica dataset. 46
A.4 Logistic Regression, and AND, OR, XOR boolean functions 47
A.5 Simple Neural Network for XOR 48
A.6 Simple Neural Network for XOR: showing the first neuron of first layer 50

A.7 Simple Neural Network for XOR and AND 51
A.8 Initial Layers of ANN containing L layers (L-1 hidden layers and 1 output layer) 55
A.9 Final Layers of ANN containing L layers (L-1 hidden layers and 1 output layer) 56

List of Algorithms

1 Proposed Algorithm 22

Chapter 1

Introduction

1.1 Background

Convolutional neural network (CNN, or ConvNet) comprising of a variation of multilayer perceptrons, falls under the category of deep feed-forward artificial neural networks [1]. Rooted in parameter/weights sharing and translation invariance behavior, and burrowing the concept of translational symmetry in Physics and Mathematics, it is also called shift/space invariant artificial neural network (SIANN). When looking for a particular feature, even if the image is shifted, CNN will still be able to detect that feature: the same weight will act upon even if the spatial position of the feature changes. Shifting the image presents with the translated but same output. CNNs are designed to employ minimal preprocessing. In comparison to the traditional algorithms where the filters (shared weights) were hand-engineered, CNN learns them during the training.

Convolutional networks are biologically-inspired in the sense that the connections of the neurons bear a resemblance to the structure of the visual cortex. Each cortical neurons are sensitive to small sub-regions of the visual field, called a receptive field. In order to incorporate or reach the entire visual field, the receptive fields are tiled and they partially overlap. In traditional fully-connected Artificial Neural Network (ANN), the entire visual field is the receptive field at each level of the network.

The invariant features play a vital role during object recognition. An object can have infinite projections depending upon the angle of view and distance of the viewer from the object. The feature vector describing an object changes with the change in the angle of view, position, entire shape, or any sort of transformations that undergoes preserving the identity of the object. The projection of these feature vectors into a high-dimensional feature space yields a relatively

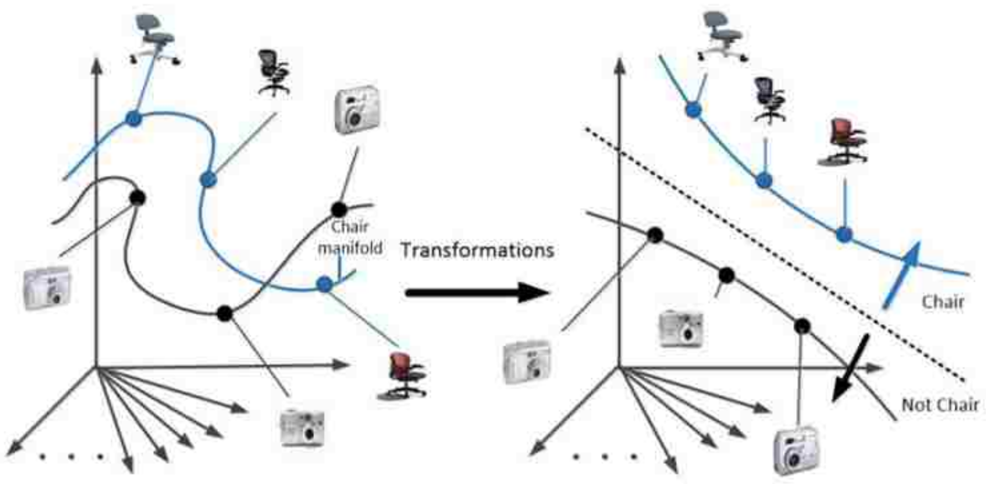

Figure 1.1: With the assessment of the sequence of transformations, each manifold becomes compact, while different manifolds are distant. For example, chair manifold (blue) and non-chair manifold (black)[25]

low-dimensional manifold; more dimension means more information/features and information is knowledge, and more knowledge drives towards more clarity. The human visual cortex achieves invariance in feature extraction in a similar manner [5]−[7]. Ventral stream is responsible for the ability of human brains to realize object recognition. The four layers V1, V2, V4, and IT compose the ventral stream. The manifolds from different object categories are very tangled at V1 layer. Neurons become able to recognize different object classes gradually from V1 layer to the IT layer, implying that different manifolds will be progressively untangled. On reaching the IT layer, the within-manifold distance becomes minimal, and the between-manifold distance is maximal. (see Fig. 1.1).

In figure 1.2.A, from the front view it is perceived that there is only one green colored sphere, but when observed from the top view (or in three-dimensional space) it is seen that there are two distant different green colored spheres. In figure 1.2.B, from the front view it is perceived that there is only one green colored sphere, but when observed from the top view (or in three-dimensional space) it is seen that there are two distant different colored spheres (green and orange). In figure 1.2.C, from the front view it is perceived that there is only one green colored object, but when observed from the top view (or in three-dimensional space) it is seen that there are two distant different green colored spheres. In figure 1.2.D, from the front view it is perceived that there is only one partial green/orange colored object, but when observed from the top view (or in three-dimensional space) it is seen that there are two distant different colored spheres (green and orange).

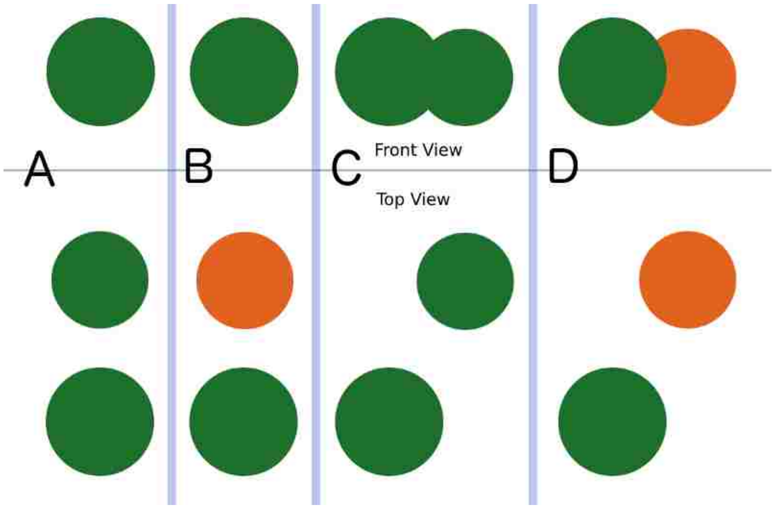

Figure 1.2: Two spheres viewed from the front and top view; same colored (1 and 3), and different colored (2 and 4) spheres.

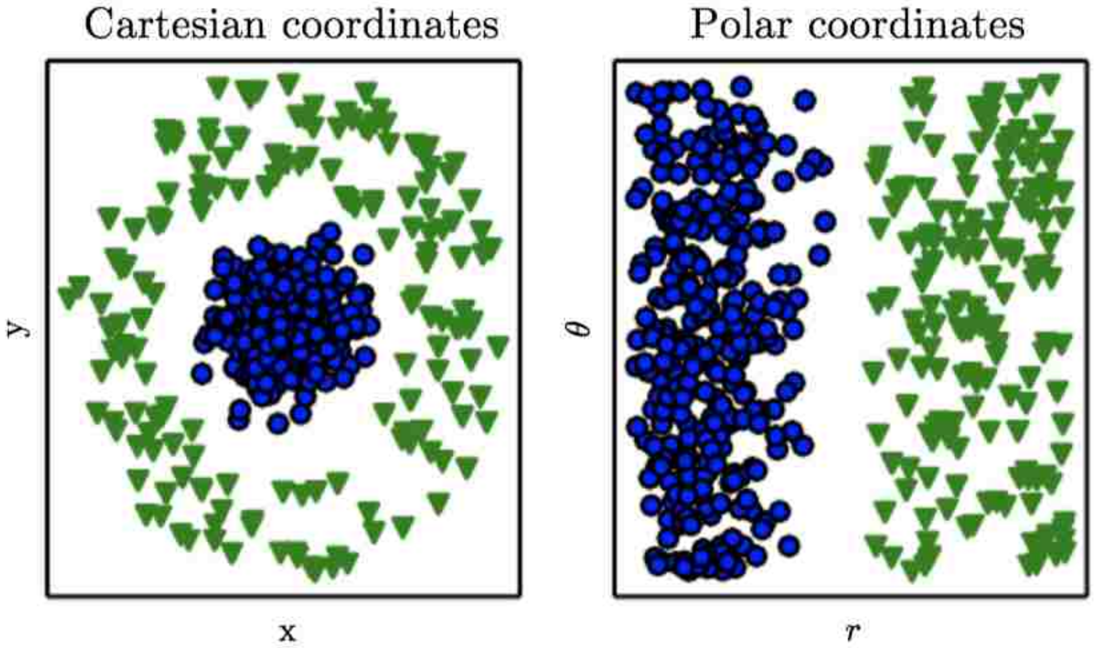

Figure 1.3: Example of different representations: cartesian coordinate system vs polar coordinate system.

If we have two classes of data in a scatterplot (Fig. 1.3), and the task is to draw a line as a boundary between them, representing the data in the cartesian coordinate system makes it impossible. Changing the same data points to the polar coordinate system turns the task into a feasible work, as a simple vertical line will be able to separate the data points into two corresponding classes. (Figure produced in collaboration with David Warde-Farley.) After projecting/representing the data into the polar coordinate system, this two-class classification problem can be solved using models that can work with linearly separable data (such as the Perceptron algorithm).

Deep networks have more representational power than shallow ones; they have greater opportunity to learn the internal structure of the actual process/model that generated the data instances in the assessment of which the network is learning. Training deeper and complex network structures using ultra large-scale training data [21], [22] has been the recent trend to improve the performance accuracies of CNN models. However, the aid of this to the network performance is limited as the model reaches a certain level of complexity. The complexity of a CNN model can be increased by the increment in the depth and/or the number of activation maps out of the convolutional layers.

1.2 Objectives and Summary of Approach

With the above-mentioned problems in mind, we propose a novel feature maps covariance minimization approach for boosting the performance of the Convolutional Neural Networks.

The visualization of the feature maps within the convolutional neural network models reveals the features to be far from random uninterpretable patterns. Rather, they show many intuitively desirable properties such as compositionality, increasing invariance and class discrimination as we ascend the layers. The model, while trained for classification, is highly sensitive to local structure in the image and is not just using broad scene context.

Each neuron in a layer of a CNN is linked to a subspace of the previous layer, and the neurons of the following layer get their input from the subspace of the current layer. Each new layer is subsequently built out of the previous layer. This behavior of CNN leads the initial layer of CNN to be able to capture the slightest part of the image, while the following layers successively become able to capture larger portions of the image, finally leading the final layer to be able to recognize the entire image.

In CNN, the projections from each layer show the hierarchical nature of features in the network. The first layer represent the lines and curves. The second layer responds to corners and edge/color conjunctions. Following layers capture textures, mesh patterns, where the final layers represent the objects. A certain combination of features in a certain area can signal a larger, more complex feature exists there. The initial layers of CNN learn the slowest, however, the overall performance of the network is largely impacted by the rate of learning of these layers.

We want to let the shared weights learn such that the feature maps generated by projecting the previous layer with the shared weights contain maximum information. The idea is based on the fact that better pattern recognition systems can be built by relying more on automatic learning. This will reduce the redundancy of feature maps, ultimately leading the CNN towards faster convergence. If the feature maps of a convolutional layer of a CNN are orthogonal, the information carried by them is not redundant. We want variance of projected feature maps X to be maximized. So, the optimization problem is to minimize the covariance between the pair of feature maps.

1.3 Organization of the Thesis

This thesis starts with the abstract with the basic idea of what it is about. Apart from prologue and epilogue, the thesis is divided into six chapters.

The first chapter contains the general introduction. It provides a brief background on the topic and provides an outline of the thesis. It discusses about the scope of the thesis and the objectives to be fulfilled on the completion of the thesis.

The second chapter provides the literature review about the project. It also contains information about the previously performed tasks on the related field.

The third chapter consists of the core part of the thesis. It contains the design details of the method.

The fourth chapter includes the experimental evaluation for the image classification task. It enlists and explains the experimental evaluation of the approach on four standard datasets: MNIST, Fashion MNIST, CIFAR-10, and CIFAR-100.

The fifth chapter explains the experimental evaluation of 1-CNN vs 4-CNN on enhancing the image resolution.

The sixth chapter concludes the thesis.

Chapter 2

Related Work

With the first proposal of programmable computers, before a hundred years one was actually constructed, it was a matter of wonder to think if the machines will gain intelligence. Augusta Ada King, Countess of Lovelace wrote first ever computer program in 1842 for the conceptual computer conceived by Charles Babbage. In 1844, Ada expressed her wish of creating a mathematical model that can describe the occurrence of ”the rise to thoughts” and ”nerves to feelings” in the brain (a calculus of the nervous system).

Artificial intelligence (AI) refers to the ability of the machines to perform tasks that the intelligent life forms are capable of doing. During its beginning time period, AI speedily became able to solve the problems to which human beings find intellectual difficulties. It was because of the reason that computers are very good at solving problems which are describable with the help of formal, and mathematical rules. The real challenge that lied in front of AI was when it was exposed to the problems which people solve naturally, with the help of intuition. For example, recognizing sounds, and identifying images [27]. It is very difficult to write formal rules for them. Learning by experience, along with understanding the concept with regard to hierarchical concepts: building complicated ones gradually out of simpler ones, is the answer. The knowledge gathered from experience circumvents the requirement of formal specification of the entire knowledge needed by the computers. Technological singularity, which is not a good event to occur, is plausible (not in the near future). However, here, we are not talking about the technological singularity.

To deal with everyday life, we require a deep understanding of the world we reside in. We are constantly exposed and bombarded to infinite variables by the universe, 90% of which we are not continuously aware of; it does not mean we are not reacting toward them. The subconscious mind, with the continuous experience from exposures and learning as a reaction to these, naturally, without our complete conscious awareness is taking care of them. For example; breathing, responding to gravity, reacting to light, and so on. 'Nature' and 'nurture' are two fundamental aspects that greatly influence our behavioral pattern; instances belonging to a common species reveal the greatest intersection of the traits because of their exposure to the common environment, and also largely because of the similar information perpetuated by genes. Genes evolve through generations of learning from their interaction with nature. Knowledge gained as such is the product of intuition. The subjective property of it places hindrance in their formal articulation. For computers to display intelligence, they have to find a way to capture the very knowledge. It is not that easy to make the computers understand this informal knowledge. For example, systems that rely upon the hard-coded knowledge. The solution is to provide the computing system the ability to extract the best patterns within the raw data and acquire the knowledge that fits them the best, via learning by experience. This approach to learning by machines is referred to as machine learning (ML).

Logistic regression, an ML algorithm, is capable of recommending cesarean delivery [28]. In case of a simple machine learning algorithm like this, the presentation of the data fed to them greatly influences their performance. Actually, the dependence with representation is a natural phenomenon. It is easy for people to perform arithmetic on Arabic numerals, and difficult to do the same on Roman numerals. People can easily perform arithmetic on a decimal number system, but find difficult with binary number system or hexadecimal number system; while computers use the binary number system. Similarly, the Fourier transform is done on signal to change it into the frequency domain from the time domain before doing any frequency analysis on the signal. For almost all the real-world tasks, it is not easy to determine the features to be extracted.

Representation learning emerged as a very popular terminology because of the above-mentioned reasons. It uses machine learning in order to extract the representation itself, along with the mapping of representation to output. Hand-designed representations are often inferior to the learned representations; one often times performance wise, and the other the model under learning is allowed to learn the representation that fits the domain of the problem it is currently trying to solve. It results in the ability of the Artificial Intelligence models to fleetingly adapt when exposed to novel tasks, without much human intervention. To be able to capture pragmatic features set, a representation learning algorithm takes minutes with regard to easy tasks, while hours to months with regard to complex tasks. It involves a tremendous amount of time and effort in the manual design of features when solving complicated tasks.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts. For example, simple concepts like corners and contours can combine to define edges.

The first wave (cybernetics; 1940s-1960s) of representation learning developed theories of biological learning [29][30], and implemented the first models, like the perceptron [31], which enabled a single neuron training. The second wave (connectionism; 1980-1995) introduced backpropagation [32] enabling the training of a neural network consisting of one or two hidden layers. Deep learning, which is capable to learn the better representation of data, the current and the third wave, began around 2006 [33][34][35].

2.1 First Wave: Cybernetics

In the early model of brain function, McCulloch-Pitts [29] model of the neuron, the inputs from input neurons combine linearly whereby the synaptic weights rescale the inputs before combining; the result oftentimes passed through an activation function. This linear model is able to decide the class of provided inputs into one of the given (two) classes depending upon the sign of the f(x, w). The weights must be set correctly for the model to make correct class predictions; maybe by a human operator. The first model to learn the weights with the aid of labeled inputs/samples/examples was the perceptron algorithm [31]. It could learn the weights defining the categories of input examples. The adaptive linear element(ADALINE), on the other hand, could learn from data and predict a real number value [36].

The simple learning algorithms as such made a great impact on the field of machine learning. The weights of ADALINE were learned in progression using the concept of the training algorithm called stochastic gradient descent. A little modified variant of stochastic gradient descent dominates the training method of deep learning models.

The Perceptron, and the ADALINE are the instances of parametric approach to supervised learning. Supervised learning is a type of learning which learns $t : \boldsymbol{X} \to \boldsymbol{T}$, where t is a function which maps the elements of $\boldsymbol{X}, \boldsymbol{X} = \{\boldsymbol{x_i}\}$; i = 1, 2, ..., N to $t(\boldsymbol{x_i}) \in \boldsymbol{T}$, only with the help of given labeled training (noisy) data; $(\boldsymbol{x_1}, t_1)$, $(\boldsymbol{x_2}, t_2)$, ... , $(\boldsymbol{x_N}, t_N)$. The parametric approaches to supervised learning learns a function h such that h is a close approximation of t, and $h = h_w(\boldsymbol{x}) = h(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{x}, \boldsymbol{w})$; that is, the function h is a representation of parameter $\boldsymbol{w}$. The construction error can be used to train the model.

Gradient descent is a method to find the optimal weights. The residual error is computed and in order to minimize the error, the sharp forward direction is taken in each step on the basis of the cost function. For example:

$$E = \sum_{i=1}^{N} (h(\boldsymbol{x_i}, y_i) - t_i)^2$$

∇w is computed for each weight, and then the weights are updated with some learning rate.

$$w_{k+1} = w_k - learning\_rate * \nabla w$$

The cost function (J) has to be convex and differentiable. For example, J = $\frac{1}{2}$ E can be used.

2.2 Second Wave: Connectionism

The brain is the major influence in the conceptualization of numerous computing units gaining intelligence via their interactions. Neocognitron [37] introduced a strong model to process images bringing forth the inspiration from the architecture of the visual system in mammals. Modern convolutional network [38] is based on this.

A complicated version of the neocognitron with inspiration from the knowledge of the function of the brain was put forward by the original cognitron [39] [40]. The simplified modern version is the result of ideas gathered from multiple views, with Nair and Hinton [41] and Glorot et al. [42] having influence from neuroscience, and Jarrett et al. [43] with more engineering-oriented influences.

The main achievement of the connectionist movement was the successful training of deep neural networks (with internal representations) using backpropagation [44][38]. It is the dominant method to train deep neural network models. Backpropagation is a method to perform gradient descent on neural networks where the error signal for the last layer is computed, let us say δ^L, and then all the error signals for the remaining layers, or δs (δ^{L-1}, δ^{L-2}, ..., δ^2, δ^1) are computed in a backward in a similar manner as that of the forward propagation, but in a reverse direction. With the help of these error signals, gradients for the weights and biases are computed. And, the gradient descent is performed.

Universal approximation theorem states that | f($\boldsymbol{x}$) - F($\boldsymbol{x}$) | $< \epsilon$ for $\epsilon > 0$ where

$$F(\boldsymbol{x}) = \sum_{i=1}^{k} \alpha_i \sigma((\boldsymbol{w_i})^{\boldsymbol{T}} \boldsymbol{x} + b_i)$$

where, $\alpha_i, \boldsymbol{w_i}, b_i$ constitute of real numbers, and σ is a monotically increasing, continuous, nonconstant, bounded function.

This means that with some ϵ we can represent the function producing the results.

Though a huge number of possible instances/outcomes, the number of favorable instances/outcomes is surprisingly very less. For example, if we consider a 1000x1000 pixel image, each pixel can take a value from the pool of 256 (0-255) possible values. So, $255^{1000000}$ possible instances are there, which is way larger than the number of atoms in our known, observable universe (10^{78} to 10^{82}). The images we deal with barely make up to that number. It is because of the laws of nature having the property to be describable using physical equations. Physical properties governing the laws of nature limit the number of favorable instances. This extremely awe-inspiring and magnificent behavior of nature is the foremost factor that makes the machine learning algorithms work, given good enough data and required computing power.

ϵ or residual error is mostly the remnant of noise. There is always noise in the data we record from nature; partly due to the inability of the sensors we use, and partly because due to other reasons. For instance, in the case of the image, a huge deal of noise comes from the inability to take the same picture twice with exactly the same features. One of the dominant factors for this is the continuous change in sunlight throughout the day, and throughout the life span of the sun.

For confined or noiseless systems, most mathematically strong models perform really well, that is, ϵ is close to zero, or even equal to zero.

2.3 Third Wave: Deep Learning

Convolutional neural networks (CNNs) are admired choice in response to the state-of-the-art accomplishments they have exhibited with respect to practical image classification tasks. The rapid advancement of computing infrastructures have made the training of complex models pragmatic, and the rapid increase in the size of widely published datasets have dramatically reduced the obstacle of overfitting in the case of deep CNNs.

Lin et al. [24] presented the network-in-network (NIN). Unlike traditional CNNs, it uses micro multilayer perceptrons instead of convolution kernels/filters. VGG-19 [21], with the help of 16 convolutional and 3 fully connected layers, attained top-1 error rate of 24.4% on the ImageNet dataset. In comparison to Alexnet, it improved the performance by 16.3%. A more complex network known as GoogLeNet [22] consisting of 22 convolutional layers and no fully-connected layer, with 25.6% top-1 error rate is slightly inferior to VGG-19. A more complicated version of GoogLeNet is called BN-Inception [20]. BN-Inception comprises of 31 convolutional layers and reduces the top-1 error rate to 21.99%. The recent deep CNN MSRA ResNet [23] consists of 152 whopping layers. It further improved the performance with the top-1 error rate of 19.38%.

Also, despite all the aforementioned advancement, the convergence of very deep CNN models (e.g., GoogLeNet [22], BN-Inception [20], and ResNet [23]) is becoming more difficult, and computationally very expensive. The improvement of the performance of deep CNNs by the backpropagation algorithm is being supported by the development in the many strategies employed during the training process: different types of activation functions [9]-[11], variety of pooling methods [12]-[15], weight decay[18], pretraining [19], and batch normalization [20] and other methodologies.

The concepts like dropout[16] and dropconnect[17] regularize the network training and ultimately try to avoid the overfitting problem by discarding some information of the training data while training. DropConnect is a generalization of Dropout, for regularizing large fully-connected layers within neural networks. When training with Dropout, a randomly selected subset of activations are set to zero within each layer. DropConnect instead sets a randomly selected subset of weights within the network to zero. Each unit thus receives input from a random subset of units in the previous layer. LeCun et al. show state-of-the-art results on several image recognition benchmarks by aggregating multiple DropConnect-trained models [17].

Inspired by the manifold learning in the brain, the MinMax objective [25] was developed by Shi et al. for boosting the performance of CNNs. Rather than increasing the complexity of the network, this approach focuses on enforcing the features learned by CNN under training to be able to maintain compactness among manifolds belonging to an object category, while maintaining maximum possible separation among manifolds belonging to different object categories: maximum between-manifold distance and minimum within-manifold distance.

The two-streams hypothesis [8] proposes that humans have two distinguished visual systems. Recent evidence suggests that there are two distinguished auditory systems too. Visual information on leaving the occipital lobe and auditory information on leaving the phonological network transmits through two main pathways. The ventral stream (what pathway) is responsible for the object and visual identification and recognition. The dorsal stream (where pathway) is responsible for computing the objects spatial location relative to the viewer and with speech repetition.

Inspired by two-streams hypothesis, bilinear CNN was proposed by Lin et al. [26]. It consists of two CNN-based feature extractors; for obtaining image descriptor, the outputs of whose are multiplied using the outer product at each location of the image and pooled over entire locations. The outer product is capable of capturing pairwise correlations between the feature maps/channels. It can also model the part-feature interactions, that is if one of the networks was a part detector and the other a local feature extractor.

Chapter 3

Proposed Approach

3.1 Covariance and Correlation

Covariance and Correlation analysis is done very often in the field of statistics and probability theory. They assist us in determining the relation and extent of the dependency that exists between two random variables. Though they look similar, they represent different things. Covariance calculates the degree of change in two random variables. Correlation, on the other hand, shows the strength of the relationship between the two variables.

Covariance indicates the direction of the relationship between variables. Correlation on the other hand measures both the strength and direction of the relationship between two variables. Correlation is a function of the covariance. What sets them apart is the fact that correlation values are standardized whereas, covariance values are not. You can obtain the correlation coefficient of two variables by dividing the covariance of these variables by the product of the standard deviations of the same values. Standard Deviation essentially measures the absolute variability of a datasets distribution. When you divide the covariance values by the standard deviation, it essentially scales the value down to a limited range of -1 to +1. This is precisely the range of correlation values. Covariance values lie between $-\infty$ to $+\infty$. In simple terms, covariance measures correlation, and correlation is a scaled version of covariance. The change in scale of the variables affects the covariance value. If we multiply the values of a given variable with a constant and the values of the other variable with the same or another constant, we notice a change to the value of covariance as well. However, it is not the case with the correlation: the change in scale of the variables does not affect the correlation value. Both covariance and correlation are the measures of the relationship between two variables. If one is zero the other becomes zero as well. Moreover, the change in location affects neither covariance nor correlation.

The formula for calculating covariance is shown below.

$$covariance(x, y) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n-1}$$

x = the independent variable

y = the dependent variable

n = number of data points in the sample

$\overline{x}$ = the mean of the independent variable x

$\overline{y}$ = the mean of the dependent variable y

Similarly, the formula for calculating correlation is shown below.

$$correlation(x, y) = \frac{cov(x, y)}{s_x s_y}$$

cov(x, y) = covariance of the variables x and y

s_x = sample standard deviation of the random variable x

s_y = sample standard deviation of the random variable y

With the use of standardized datasets, correlation turns out to be equivalent to covariance. If it comes to a matter of choice or natural selection, correlation is preferred method instead of covariance. The reason for this is due to the fact that correlation is not affected by the alteration in either the location or the scale.

Correlation analysis is a vital tool for feature selection and multivariate analysis in data preprocessing and exploration. Correlation helps us investigate and establish relationships between variables. This is employed in feature selection before any kind of statistical modeling or data analysis. PCA or Principal Component Analysis is one significant application of the same.

So how do we decide what to use? Correlation matrix or the covariance matrix? In simple words, it is advised to use the covariance matrix when the variables are on similar scales (or standardized) and the correlation matrix when the scales of the variables differ. Standardizing the dataset and then computing the covariance and correlation matrices will yield the same results. Covariance is computationally cheaper than correlation, as the tremendous amount of division operations combinedly is very expensive. Thus, we standardize the data and use covariance. Also, there is a significant benefit in the performance of the learning with standardized data; almost all machine learning models converge faster with the standardized data.

In this thesis, we propose a model that uses the pair-wise feature maps covariance minimization.

3.2 Correlation Matrix and Convolution Matrix

The neighboring pixels of an image are highly correlated as compared to the pixels that are lying further. So, in a CNN, we take size (n_k) of kernel or convolution matrix just enough to capture this. n_k=5 is a good option, but not the universal choice. As we move about 8 pixels away, the correlation becomes negligible. Here, we are talking about the correlation between the spatial pixels in an image.

A convolution matrix also known as a mask, kernel or filter is simply a little matrix of weights. During the convolution process each element of matrix weights the corresponding pixel of the current receptive field or window, and the thus weighted local neighbors' pixel values are summed up to obtain a new value.

The idea is inherited from mathematical convolution. Mathematically, convolution is simply an operation performed on two functions, say, f and g producing as an output a third function, whereby the result explains how the shape of one alters the other. It bears resemblance with cross-correlation. Considering the case of discrete/continuous variable, and real-valued functions, the difference lies only with the reversal of one of the given functions, that is, in this case, where f(x) and g(x) are the given functions, cross-correlation of the f(-x) and the g(x), or the f(x) and the g(-x) yields convolution.

In the case of CNN, the convolution matrix is obtained by rotating the correlation matrix by 180 degrees, or vice versa. Correlation measures the similitude between the two sequences. Convolution, on the other hand, measures the impact of sequences on one another. For standardized data, the covariance matrix and correlation matrix are the same. Similarly, in the case of the symmetric matrix, correlation matrix and convolution matrix are the same.

Let **H** be a 3x3 correlation kernel/matrix.

$$\boldsymbol{H} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix}$$

If $w_{11} = u_{33}$, $w_{12} = u_{32}$, $w_{13} = u_{31}$, $w_{21} = u_{23}$, $w_{22} = u_{22}$, $w_{23} = u_{21}$, $w_{31} = u_{13}$, $w_{32} = u_{12}$ and $w_{33} = u_{11}$, then

$$\mathbf{W} = \begin{bmatrix} u_{33} & u_{32} & u_{31} \\ u_{23} & u_{22} & u_{21} \\ u_{13} & u_{12} & u_{11} \end{bmatrix}$$

or,

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

W is the convolution kernel/matrix.

The choice of the word ”convolution” in the scenario of neural networks is a result of convention. The process of convolution in CNN is in a mathematical sense more closely a cross-correlation. But, it does not matter as the index of w_{ij} does not have much significance as these weights are learned by the network.

In a CNN, what we are actually doing is projecting the receptive field with a filter. Consider a single neuron. Its output is simply the dot product, the sum of the elementwise products of the two same-sized sequence: corresponding weights multiply corresponding features (bias being multiplied by 1) and the results are added to give a single scalar value. If the specific spatial receptive field or current window and also the filter are smashed/reshaped into the respective sequences, convolution is mathematically identical to dot product. Or, in simple words filters are just projecting the visual fields into a new topological dimension.

W is used as a filter. Filtering helps modify or enhance images; filtering an image is useful to spotlight specific features while removing other features. We use many but different convolution filters at each convolutional layer in our trial to capture as many distinct features as possible. When we slide and share the weights in a CNN layer, each filter looks for the multiple presences of the specific feature in an entirety of the image.

3.3 Proof of Principal Component Analysis

3.3.1 Introduction

Principal Component Analysis (PCA) refers to a statistical technique deployed mostly for the purpose of reducing data dimension, compressing data, extracting features, and visualizing data.

It is a process of projecting the data in D dimensional feature space to k dimensional feature space, $k < D$, such that projected data has maximum variance.

Principal axes refer to the axes that best describe the data when projected onto them. For example, in case of individual points in a 2-dimensional cartesian coordinate system, x-axis and y-axis are the principal axes of the default 2-D system. In the 2-D coordinate system, each individual point cannot be described in any better way: every point when projected on these axes have distinct value and can be uniquely identified.

PCA was invented in 1901 by Karl Pearson [45], as an analog of the principal axis theorem in mechanics; it was later independently developed and named by Harold Hotelling in the 1930s [46]-[47]. The principal axis theorem states that the principal axes are perpendicular, and gives a constructive procedure for finding them. The operation of PCA reveals the internal structure of the data in a way that best explains the variance in the data. PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset.

3.3.2 Statement

Let, $\mathbf{\Sigma}$ be the covariance matrix.
λ_1 be the largest eigenvalue of $\mathbf{\Sigma}$.
$\boldsymbol{u_1}$ be the eigenvector corresponding to λ_1.
- also called the first principal component.

For $M < D$ dimensions:
- $\boldsymbol{u_1 u_2 ... u_M}$ are the eigenvectors corresponding to the largest eigenvalues λ_1 λ_2 ... λ_M of $\mathbf{\Sigma}$.

3.3.3 Proof by Induction

Base Step (k=1)

We want variance of projected $\mathbf{X}$ to be maximized. So, the optimization problem is to minimize -$\boldsymbol{u_1^T}\mathbf{\Sigma}\boldsymbol{u_1}$ subject to $\boldsymbol{u_1^T u_1} = 1$.

Using the Lagrangian function,

$$L_p(\boldsymbol{u_1}, \lambda_1) = -\boldsymbol{u_1^T}\mathbf{\Sigma}\boldsymbol{u_1} + \lambda_1(\boldsymbol{u_1^T u_1} - 1)$$

where λ_1 is the Lagrangian multiplier for constraint $\boldsymbol{u_1^T u_1} = 1$.

Solving

$$\frac{\delta L_p}{\delta \boldsymbol{u_1}} = 0$$

$$\boldsymbol{\Sigma u_1} = \lambda_1 \boldsymbol{u_1}$$

$\boldsymbol{u_1}$ is an eigenvector of $\boldsymbol{\Sigma}$, λ_1 is an eigenvalue of $\boldsymbol{\Sigma}$

$$-\boldsymbol{u_1^T \Sigma u_1} = -\lambda_1 \boldsymbol{u_1^T u_1} = -\lambda_1$$

We are minimizing -$\boldsymbol{u_1^T \Sigma u_1}$, and the result just above shows that this is equivalent to maximizing λ_1: λ_1 must be the largest possible. So, λ_1 is the largest eigenvalue of $\boldsymbol{\Sigma}$, and $\boldsymbol{u_1}$ is the eigenvector corresponding to λ_1. The first principal component of $\boldsymbol{x}$ is $\boldsymbol{u_1^T x}$.

Inductive Step

Additional principal components can be defined incrementally by choosing each new projection direction as the one with maximum projected variance among all directions orthogonal to those already considered.

Following the inductive hypothesis (for k=K), $\boldsymbol{u_K}$ is the eigenvector corresponding to the K^{th} largest eigenvalue λ_K of $\boldsymbol{\Sigma}$.

Then for k=K+1,

The $(K+1)^{th}$ principal component $\boldsymbol{u_{K+1}^T x}$ must minimize -$\boldsymbol{u_{K+1}^T \Sigma u_{K+1}}$ such that $\boldsymbol{u_{K+1}^T x}$ is uncorrelated with $\boldsymbol{u_K^T x}$, $\boldsymbol{u_{K-1}^T x}$, $\boldsymbol{u_{K-2}^T x}$, ..., $\boldsymbol{u_2^T x}$, $\boldsymbol{u_1^T x}$. The covariance between $\boldsymbol{u_{K+1}^T x}$ and $\boldsymbol{u_K^T x}$ is 0, $\boldsymbol{u_{K+1}^T x}$ and $\boldsymbol{u_{K-1}^T x}$ is 0, $\boldsymbol{u_{K+1}^T x}$ and $\boldsymbol{u_{K-2}^T x}$ is 0, ..., $\boldsymbol{u_{K+1}^T x}$ and $\boldsymbol{u_2^T x}$ is 0, $\boldsymbol{u_{K+1}^T x}$ and $\boldsymbol{u_1^T x}$ is 0.

$$cov(\boldsymbol{u_1^T x}, \boldsymbol{u_{K+1}^T x}) = E[(\boldsymbol{u_1^T x})(\boldsymbol{u_{K+1}^T x})^T] = \boldsymbol{u_1^T} E((xx^T))\boldsymbol{u_{K+1}} = \boldsymbol{u_1^T \Sigma u_{K+1}} = \lambda_1 \boldsymbol{u_{K+1}^T u_1}$$

Similarly,

$$cov(\boldsymbol{u_2^T x}, \boldsymbol{u_{K+1}^T x}) = \lambda_2 \boldsymbol{u_{K+1}^T u_2}$$

.

.

.

$$cov(\boldsymbol{u_{K-1}^T x}, \boldsymbol{u_{K+1}^T x}) = \lambda_{K-1} \boldsymbol{u_{K+1}^T u_{K-1}}$$

$$cov(\boldsymbol{u_K^T x}, \boldsymbol{u_{K+1}^T x}) = \lambda_K \boldsymbol{u_{K+1}^T u_K}$$

$\boldsymbol{u_1, u_2, ..., u_{K-1}, u_K}$ and $\lambda_1, \lambda_2, ..., \lambda_{K-1}, \lambda_K$ are already known from base step and inductive hypothesis. So, -$\boldsymbol{u_{K+1}^T \Sigma u_{K+1}}$ must be minimized with following constraints:

$$\boldsymbol{u_{K+1}^T u_{K+1}} = 1$$

$$\boldsymbol{u_{K+1}^T u_K} + \boldsymbol{u_{K+1}^T u_{K-1}} + ... + \boldsymbol{u_{K+1}^T u_2} + \boldsymbol{u_{K+1}^T u_1} = 0$$

Using Lagrangian function,

$$L_p(\boldsymbol{u_{K+1}}, \lambda_{K+1}) =$$

$$-\boldsymbol{u_{K+1}^T \Sigma u_{K+1}} + \lambda_{K+1}(\boldsymbol{u_{K+1}^T u_{K+1}} - 1) + \phi(\boldsymbol{u_{K+1}^T u_K} + \boldsymbol{u_{K+1}^T u_{K-1}} + ... + \boldsymbol{u_{K+1}^T u_2} + \boldsymbol{u_{K+1}^T u_1})$$

where λ_{K+1} and ϕ are Lagrangian multipliers.

Solving

$$\frac{\delta L_p}{\delta \boldsymbol{u_{K+1}}} = 0$$

$$\boldsymbol{\Sigma u_{K+1}} = \lambda_{K+1}\boldsymbol{u_{K+1}} + \phi(\boldsymbol{u_K} + \boldsymbol{u_{K-1}} + ... + \boldsymbol{u_2} + \boldsymbol{u_1})$$

Multiplying both sides by $\boldsymbol{u_K^T}$,

$$\boldsymbol{u_K^T \Sigma u_{K+1}} = \lambda_{K+1}\boldsymbol{u_K^T u_{K+1}} + \phi(\boldsymbol{u_K^T u_K} + \boldsymbol{u_K^T u_{K-1}} + ... + \boldsymbol{u_K^T u_2} + \boldsymbol{u_K^T u_1})$$

$$0 = 0 + \phi(1 + 0 + ... + 0 + 0), thatis, \phi = 0.$$

Then, we get,

$$\boldsymbol{\Sigma u_{K+1}} = \lambda_{K+1}\boldsymbol{u_{K+1}}$$

$\boldsymbol{u_{K+1}}$ is an eigenvector of $\boldsymbol{\Sigma}$, λ_{K+1} is an eigenvalue of $\boldsymbol{\Sigma}$

$$-\boldsymbol{u_{K+1}^T \Sigma u_{K+1}} = -\lambda_{K+1}\boldsymbol{u_{K+1}^T u_{K+1}} = -\lambda_{K+1}$$

We are minimizing -$\boldsymbol{u_{K+1}^T \Sigma u_{K+1}}$, and the result just above shows that this is equivalent to maximizing λ_{K+1}: λ_{K+1} must be the largest possible. So, λ_{K+1} is the $(K+1)^{th}$ largest eigenvalue of $\boldsymbol{\Sigma}$, and $\boldsymbol{u_{K+1}}$ is the eigenvector corresponding to λ_{K+1}. The $(K+1)^{th}$ principal component of $\boldsymbol{x}$ is $\boldsymbol{u_{K+1}^T x}$.

3.4 Feature Maps Covariance Minimization

Figure 3.1 shows the working of the proposed method. For a convolutional layer, at each step of gradient descent, the covariance between the pairs of feature maps with replacement is added as supplement cost to the overall cost function. (m*m-m)/2 or mC2 (m is the number of feature maps in a convolutional layer) covariances have to be calculated; finding the covariance between the pairs of feature maps without replacement will reduce the complexity (m/2). This process is repeated for all the convolutional layers and the average of the covariances is added as supplement cost to the overall cost function.

Covariance describes how two variables are related. Variables are positively related if they move in the same direction, and are inversely related if they move in opposite directions. If the two

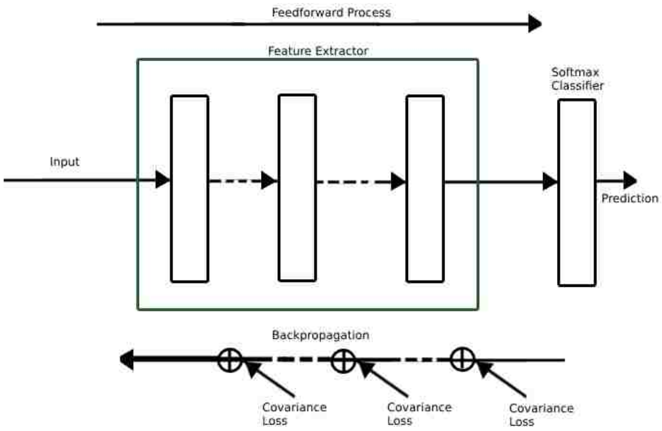

Figure 3.1: Backpropagation with feature maps covariance minimization.

random variables are independent, the covariance will be zero. But, in our case, in a convolutional layer, the feature maps represent different but similar features. For example, horizontal straight lines, vertical straight lines, slant lines, or curves. So, we propose not to nullify (make absolute value zero) the covariance, but to minimize the covariance.

Let us consider the case without replacement. m_k be the number of feature maps at k^{th} convolutional layer. Then, the number of pair-covariances to be calculated is $\frac{m_k}{2}$.

The covariance between a pair of feature maps x and y is given by

$$covariance(x, y) = \frac{\sum_{j=1}^{p} \frac{\sum_{i=1}^{n}(x_{p_i} - \overline{x}_p)(y_{p_i} - \overline{y}_p)}{n-1}}{p}$$

where, n is the batch size, and p is the number of pixels in each of the feature maps of the convolutional layer.

Then, the feature maps covariance loss for a convolutional layer is

$$covariance_k = \frac{\sum_{i=1}^{\frac{m_k}{2}} covariance(x_i, y_i)}{\frac{m_k}{2}}$$

Finally, assume there are t convolutional layers, then the total feature maps covariance loss is

$$loss_{feature-maps-covar} = \frac{\sum_{k=1}^{t} covariance_k}{t}$$

The new cost function is

$$loss_{total} = loss_{softmax} + \beta loss_{feature-maps-covar}$$

Hyperparameter β adjusts the tradeoff of the two terms.

3.5 Algorithm/Proposed Model

```
Input: N training samples {X, c}
Output: The trained model.
    Initialisation : $weights(Wg, W, Ws)$ ← normal, random or any other distributions
batches ← create_batches (X, c)
    LOOP Process
for $i = 1$ to $|epochs|$ do
    for $i = 1$ to $|batches|$ do
        feature ← feedforward_feature_extractor (batch, Wg)
        dense ← forward_propagation (W, feature)
        prediction ← softmax_classifier (Ws, dense)
        backpropagate_output_to_dense ()
        Here, the softmax loss only influences the gradient.
        backpropagate_feature_extractor ()
        Here, both the softmax loss term, and feature-maps-covariance loss influences the
          gradient.
    end for
end for
```

Algorithm 1: Proposed Algorithm

Algorithm 1 summarizes the proposed approach. The considered model architecture is composed of feature extractor, dropconnect/dropout (on top of Fully Connected ReLU Layer) layer, and softmax classifier.

For feature extractor, v = g(x; Wg) where v is the output features, x is input data to the overall model, and Wg is parameters for the feature extractor. We choose g() to be a multi-layered convolutional neural network (CNN), with Wg being the convolutional filters (and biases) of the CNN.

In case of DropConnect/Dropout layer; W is a fully connected weight matrix; outputs r as output.

Softmax classifier o = s(r; Ws) takes as input r and uses parameters Ws to map this to a k dimensional output (k being the number of classes).

The overall model f(x; θ, M) maps input data x to an output o through a sequence of operations given the parameters θ = {Wg , W, Ws }. It falls under the parametric approach to supervised learning. Model f is trained using gradient descent with backpropagation. Gradient descent is a method to find the optimal weights. The residual error is computed and in order to minimize the error, the sharp forward direction is taken in each step on the basis of the cost function. Backpropagation is a method to perform gradient descent on neural networks.

Chapter 4

Experimental Evaluation For Image Classification Task

4.1 Overall Settings

Batch size of 50, learning rate of 0.01, β (for Covariance) of $1e-3$, and exponential learning rate decay is used. For covariance-minimization, pairs are formed without replacement. Max-pool is used for the pooling.

4.2 Data Description

The model performance is evaluated on the four standard datasets: MNIST, Fashion MNIST, CIFAR-10, and CIFAR-100.

The MNIST database of handwritten digits comprises of 60,000 training examples and 10,000 test examples. The features represent row-wise organized 28X28 pixel values of grayscale images of handwritten digits 0-9. Thus, the labels values are 0 to 9. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black). Zero padding of two is added rectangularly all around the image to meet the input dimension requirement of the networks used.

Fashion-MNIST is a dataset of article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 (zero padded to make 32X32) grayscale image, associated with a label from 10 classes (types of clothes).

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The CIFAR-100 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).

Training set refers to the dataset which is actually used to train the model. The test set is the dataset that is used to verify the performance of the model. The model during training time only sees the training data.

4.3 Experiments

4.3.1 MNIST

Table 4.1: The top-1 error rate comparison of different models on the MNIST dataset for first 35 epochs.

Model	Error Rate (within 35 epochs)
Baseline	0.72 (17 epochs)
Covar	0.79 (31)
Dropout	0.60 (24)
Dropconnect	0.56 (19)
Dropout+Covar	0.52 (32)
Dropconnect+Covar	0.53 (30)

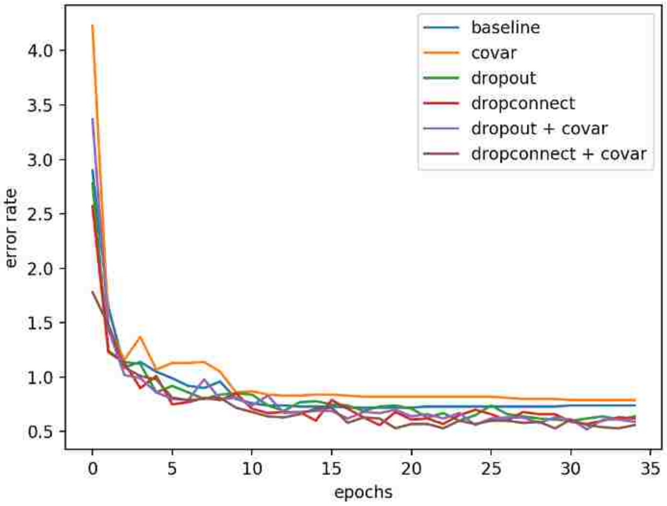

Figure 4.1: Performance comparison of different models on the MNIST test dataset.

Table 4.1 and Figure 4.1 show the performance comparison of different models on the MNIST test dataset. The experiment shows that the performance of the network goes better with dropout, dropconnect, covariance minimization(with dropconnect), and covariance minimization(with dropout). In this case, covariance minimization gives a better result only with dropout and dropconnect; the reason for it is because, without some sort of regularization, the model overfitted faster with covariance minimization. Overfitting refers to the situation when the model performs well on training data but not on test data; the model overfitted on training data and hence failed to generalize novel instances. Though the purpose of the model would be to perform well on test data, the model will always be trying to represent a function outputting the training data and always will have tendency to overfit without regularization unless we have infinite data; infinite in the sense all the possible test instances fall under training data, which is almost impossible in the practical sense.

3 layered CNN (32-64-128 feature maps) derived from the official tensorflow github example on MNIST is selected as the feature extractor.

Table 4.2: Confusion matrix of baseline

Pred\Truth	0	1	2	3	4	5	6	7	8	9	total
0	977	0	0	0	0	1	4	0	2	0	984
1	0	1134	0	0	0	1	2	2	0	0	1139
2	0	1	1027	1	1	0	0	1	1	0	1032
3	0	0	0	1001	0	3	0	0	2	0	1006
4	0	0	1	0	976	0	1	0	0	8	986
5	0	0	0	4	0	886	4	0	0	1	895
6	2	0	0	0	1	1	944	0	0	1	949
7	1	0	2	2	0	0	0	1022	1	2	1030
8	0	0	2	2	0	0	3	1	966	2	976
9	0	0	0	0	4	0	0	2	2	995	1003
total	980	1135	1032	1010	982	892	958	1028	974	1009	10000

Table 4.3: Confusion matrix of best model (dropout+covar)

Pred\Truth	0	1	2	3	4	5	6	7	8	9	total
0	979	0	1	0	0	1	2	0	1	0	984
1	0	1134	0	0	0	0	3	2	0	0	1139
2	0	0	1027	2	0	0	0	0	1	0	1030
3	0	0	0	1003	0	4	0	0	0	0	1007
4	0	0	0	0	978	0	2	0	0	5	985
5	0	0	0	4	0	886	1	0	0	3	894
6	0	0	0	0	1	1	950	0	0	0	952
7	0	1	4	0	0	0	0	1024	1	2	1032
8	1	0	0	1	1	0	0	1	969	1	974
9	0	0	0	0	2	0	0	1	2	998	1003
total	980	1135	1032	1010	982	892	958	1028	974	1009	10000

Table 4.2 and 4.3 lists the confusion matrices of baseline, and the best model (dropout+covar). The correct prediction of class 0 increased from 977 to 979, class 3 increased from 1001 to 1003, class 4 increased from 976 to 978, class 6 increased from 944 to 950, class 7 increased from 1022 to 1024, class 8 increased from 966 to 969, and class 9 increased from 995 to 998. The predictions of class 1, 2, and 5 remain unchanged.

4.3.2 Fashion MNIST

Table 4.4: The top-1 error rate comparison of different models on the fashion MNIST dataset for first 35 epochs.

Model	Error Rate (within 35 epochs)
Baseline	11.43 (33 epochs)
Covar	10.84 (34)
Dropout	9.16 (35)
Dropconnect	9.17 (33)
Dropout+Covar	9.88 (34)
Dropconnect+Covar	9.15 (34)

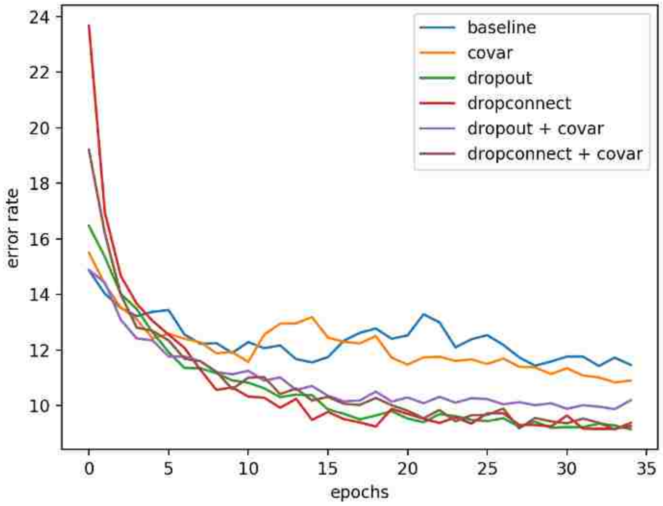

Figure 4.2: Performance comparison of different models on the fashion MNIST test dataset.

Table 4.4 and Figure 4.2 show the performance comparison of different models on the fashion MNIST dataset. The experiment shows that the performance of the network goes better with covar, dropconnect, dropout, and covariance minimization(with dropconnect). In this case, covariance minimization (with dropout) does not perform better than dropout.

3 layered CNN (32-64-128 feature maps) derived from the official tensorflow github example on MNIST is selected as the feature extractor.

Table 4.5 and 4.6 lists the confusion matrices of baseline, and the best model (dropconnect+covar). The correct prediction of class 0 increased from 815 to 903, class 1 increased from 975 to 994, class 2 increased from 806 to 845, class 3 increased from 890 to 931, class 4 increased from 809 to 884, class 5 increased from 968 t0 983, class 7 increased from 948 to 973, class 8 increased from 977 to 984, and class 9 increased from 970 to 971. But, the predictions of class 6 reduced from 699 to 617.

Table 4.5: Confusion matrix of baseline

Pred\Truth	0	1	2	3	4	5	6	7	8	9	total
0	815	1	15	19	3	0	106	0	3	0	962
1	5	975	3	8	0	0	3	0	1	0	995
2	13	0	806	17	87	0	88	0	1	0	1012
3	24	17	7	890	32	2	23	0	2	0	997
4	6	4	92	34	809	0	75	0	4	0	1024
5	0	0	0	0	0	968	0	10	0	7	985
6	129	1	75	29	67	0	699	0	7	0	1007
7	0	0	0	0	0	22	0	948	3	23	996
8	7	2	2	3	2	0	6	1	977	0	1000
9	1	0	0	0	0	8	0	41	2	970	1022
total	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	10000

Table 4.6: Confusion matrix of best model (dropconnect+covar)

Pred\Truth	0	1	2	3	4	5	6	7	8	9	total
0	903	0	18	11	1	0	183	0	0	1	1117
1	4	994	4	12	2	0	6	0	2	0	1024
2	13	0	845	8	58	0	67	0	1	0	992
3	23	5	9	931	23	0	34	0	4	0	1029
4	3	0	89	29	884	0	87	0	2	0	1094
5	1	0	0	0	0	983	0	6	3	4	997
6	44	0	35	7	31	0	617	0	1	0	735
7	0	0	0	0	0	8	0	973	3	24	1008
8	8	1	0	2	1	0	6	0	984	0	1002
9	1	0	0	0	0	9	0	21	0	971	1002
total	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	10000

4.3.3 CIFAR-10

Table 4.7 and Figure 4.3 show the performance comparison of different models on the CIFAR-10 dataset. The experiment shows that the performance of the network goes better with dropout, covariance minimization, dropconnect, and covariance minimization(with dropconnect). In this

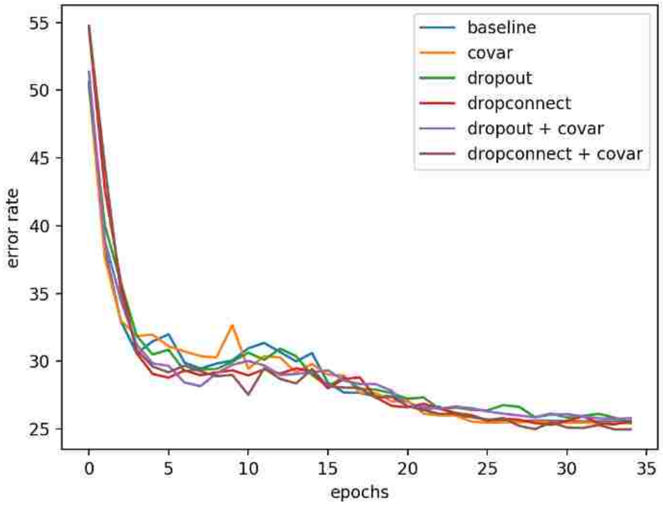

Figure 4.3: Performance comparison of different models on the CIFAR-10 test dataset.

Table 4.7: The top-1 error rate comparison of different models on the CIFAR-10 dataset for the first 35 epochs.

Model	Error Rate (within 35 epochs)
Baseline	25.56 (32 epochs)
Covar	25.42 (35)
Dropout	25.46 (35)
Dropconnect	25.30 (30)
Dropout+Covar	25.86 (29)
Dropconnect+Covar	24.97 (34)

case, up to 35 epochs, covariance minimization(with dropout) performs worse than the baseline model.

2 layered CNN (64-64 feature maps) derived from the official tensorflow github example on CIFAR-10 is selected as the feature extractor.

4.3.4 CIFAR-100

Table 4.10 and Figure 4.4 show the performance comparison of different models on the CIFAR-100 dataset. The experiment shows that the performance of the network goes better with droupout, covariance minimization(with dropout), covariance minimization, dropconnect, and covariance minimization(with dropconnect).

2 layered CNN (64-64 feature maps) derived from the official tensorflow github example on CIFAR-10 is selected as the feature extractor.

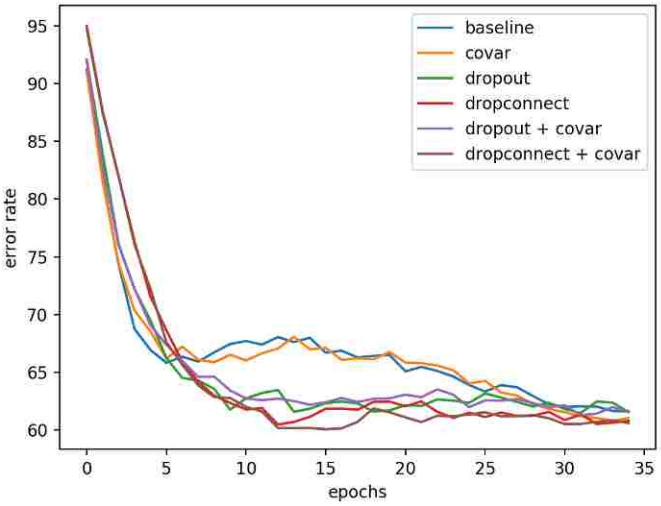

Figure 4.4: Performance comparison of different models on the CIFAR-100 test dataset.

Table 4.8: The top-1 error rate comparison of different models on the CIFAR-100 dataset for the first 35 epochs.

Model	Error Rate (within 35 epochs)
Baseline	61.62 (35 epochs)
Covar	60.79 (34)
Dropout	61.47 (32)
Dropconnect	60.46 (13)
Dropout+Covar	61.27 (32)
Dropconnect+Covar	60.06 (16)

4.4 Discussion

Experimentally, it is observed that in general case, covariance minimization with dropconnect gives the best results. However, in some cases, covariance minimization with dropout works better.

For covariance-minimization, if pairs are formed with replacement computational cost is too high, however, the performance of the model might be better. Max-pool works in practice as it accelerates the on-going decrement of the computational cost of deep convolutional neural networks, while, transmitting the maximum information from the max-pool candidates. In the convolutional network, shared weights explicitly imply less computation. Also, translation invariance behavior leads to faster convergence.

Increasing the number of feature maps, depth of the network, and the number of epochs might increase the accuracy.

4.5 Further Improvements

The methods to further improve the performance of the proposed approach are listed below.

4.5.1 ZCA Whitening/Sphering

ZCA whitening is a rough model of how the biological eye (the retina) processes images (through retinal neurons). ZCA whitening also tries to add noise robustness to the data.

4.5.2 Pair Formation (for Covar Minimization) with Replacement

While performing covariance minimization, during the backpropagation process, if the pairs are formed with replacement, the performance might get better.

4.5.3 Using Additional Pair-wise Networks

In case of fewer classes(N), two passes: one through 'N class network', and then through 'pair-wise network' (first two with higher class values). Additional NC2 networks have to be trained, however, each network can be trained independently. When N goes larger, the number of pair-wise networks to be trained is very huge and not feasible.

4.5.4 Voting Among the Networks

One way to vote can be to train N ($N > 1$) networks, each network exactly the same in structure, but trained with different hyperparameters. Another way to vote might be to call the listed 6 models for a vote, and the decision is made on the basis of that. These processes might increase accuracy.

Chapter 5

Experimental Evaluation For Enhancing Image Resolution

5.1 Overall Settings

Batch size of 50, the learning rate of 0.01, and exponential learning rate decay is used. Maxpool is used for the pooling. Mean Squared Error is used as the loss. Feature maps covariance minimization was tried here as well, and the mean squared error improved, but at the sixth/seventh decimal places only, and hence it is not included here.

5.2 Data Description

The model performance is evaluated on the four standard datasets: MNIST, Fashion MNIST, CIFAR-10, and CIFAR-100.

The images are reduced to 16x16 to make the input training features while the original images (32x32; in case of mnist and fashion mnist zero padded to make 32x32) are used as the output of the network.

The thus diminished 16x16 image is zero padded pixel by pixel to make 32x32 (requirement of the networks used). Then it is fed to the CNN as input while the original 32x32 image is fed as output. As the second method, the original 32x32 image is split into 4 equal parts and then 4 networks (exactly same configuration wise) are independently trained while feeding one of the 4 parts to each network as output.

5.3 Experiments

5.3.1 MNIST

Table 5.1 and Figure 5.1 show the performance comparison of different models on the MNIST test dataset. The experiment shows that the performance gets better when the problem is decentralized and more networks are used to solve the problem.

3 layered CNN (32-64-128 feature maps) derived from the official tensorflow github example on MNIST is selected as the feature extractor.

Table 5.1: Mean Squared Error (MSE) comparison of different models on the MNIST dataset for first 35 epochs.

Model	MSE (within 35 epochs)
Baseline (1 network)	0.0540
4 networks	0.0394

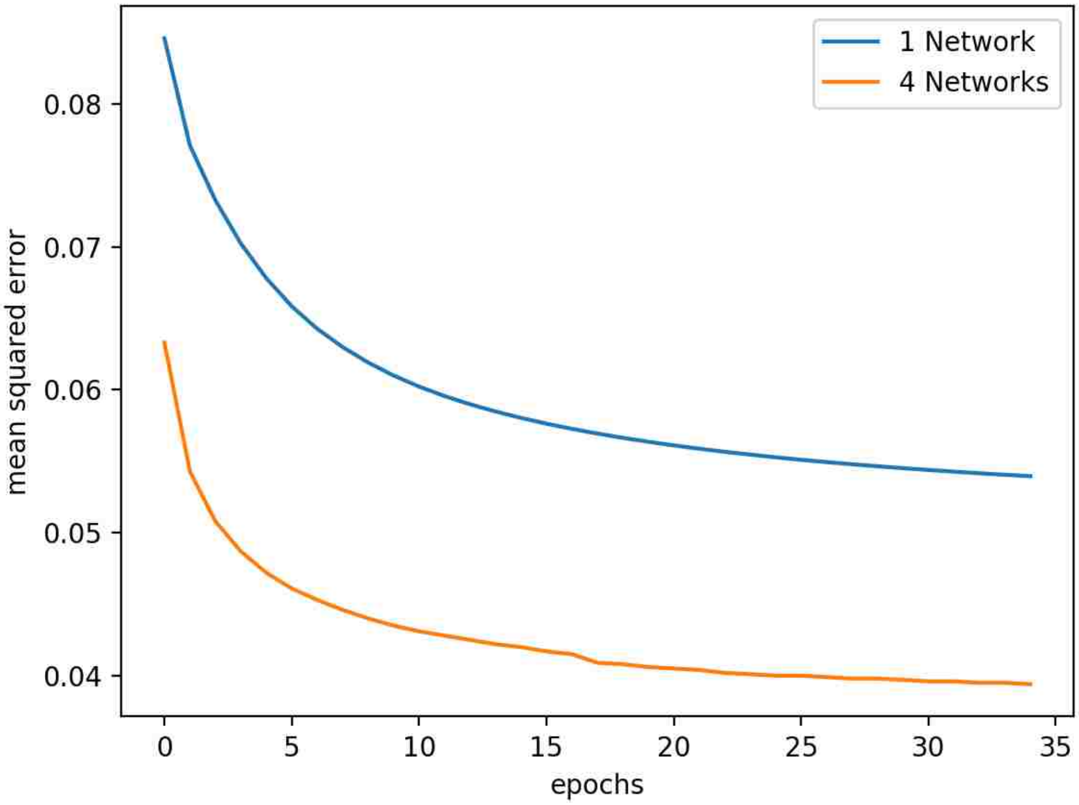

Figure 5.1: Performance comparison of different models on the MNIST test dataset.

5.3.2 Fashion MNIST

Table 5.2 and Figure 5.2 show the performance comparison of different models on the fashion MNIST test dataset. The experiment shows that the performance gets better when the problem is decentralized and more networks are used to solve the problem.

3 layered CNN (32-64-128 feature maps) derived from the official tensorflow github example on MNIST is selected as the feature extractor.

Table 5.2: Mean Squared Error (MSE) comparison of different models on the fashion MNIST dataset for first 35 epochs.

Model	MSE (within 35 epochs)
Baseline (1 network)	0.0361
4 networks	0.0253

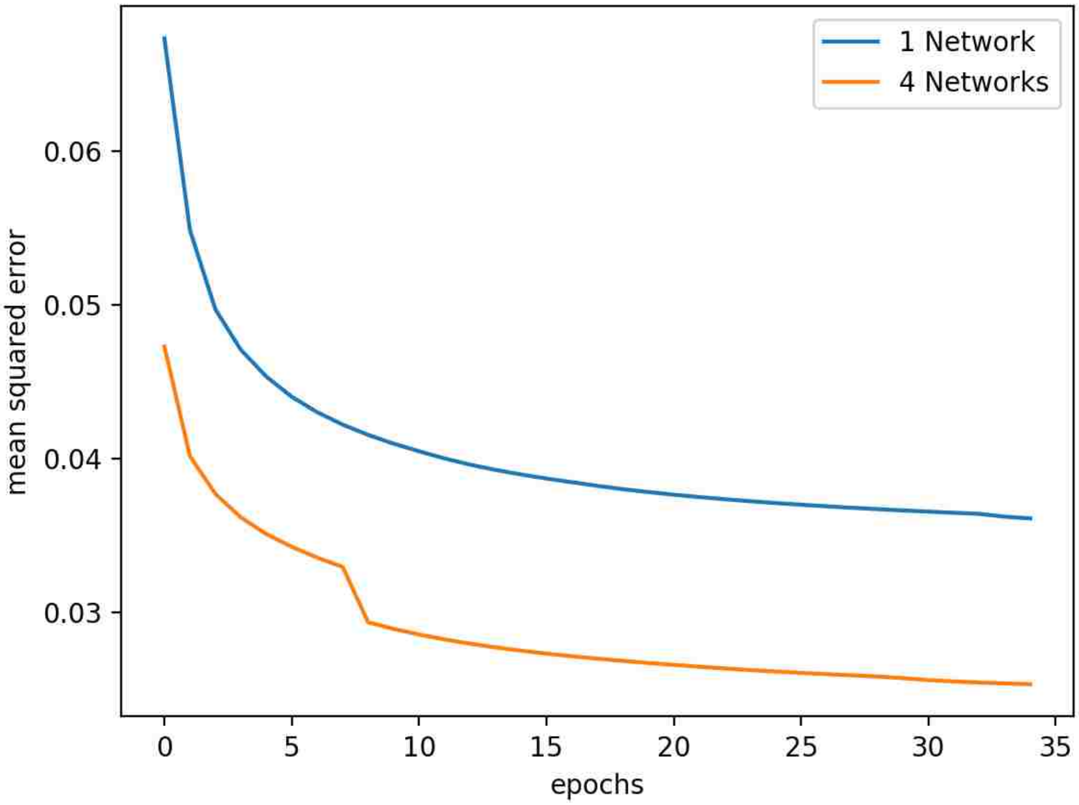

Figure 5.2: Performance comparison of different models on the fashion MNIST test dataset.

5.3.3 CIFAR-10

Table 5.3 and Figure 5.3 show the performance comparison of different models on the fashion CIFAR-10 test dataset. The experiment shows that the performance gets better when the problem is decentralized and more networks are used to solve the problem.

2 layered CNN (64-64 feature maps) derived from the official tensorflow github example on CIFAR-10 is selected as the feature extractor.

Table 5.3: Mean Squared Error (MSE) comparison of different models on the CIFAR-10 dataset for first 35 epochs.

Model	MSE (within 35 epochs)
Baseline (1 network)	0.0289
4 networks	0.0139

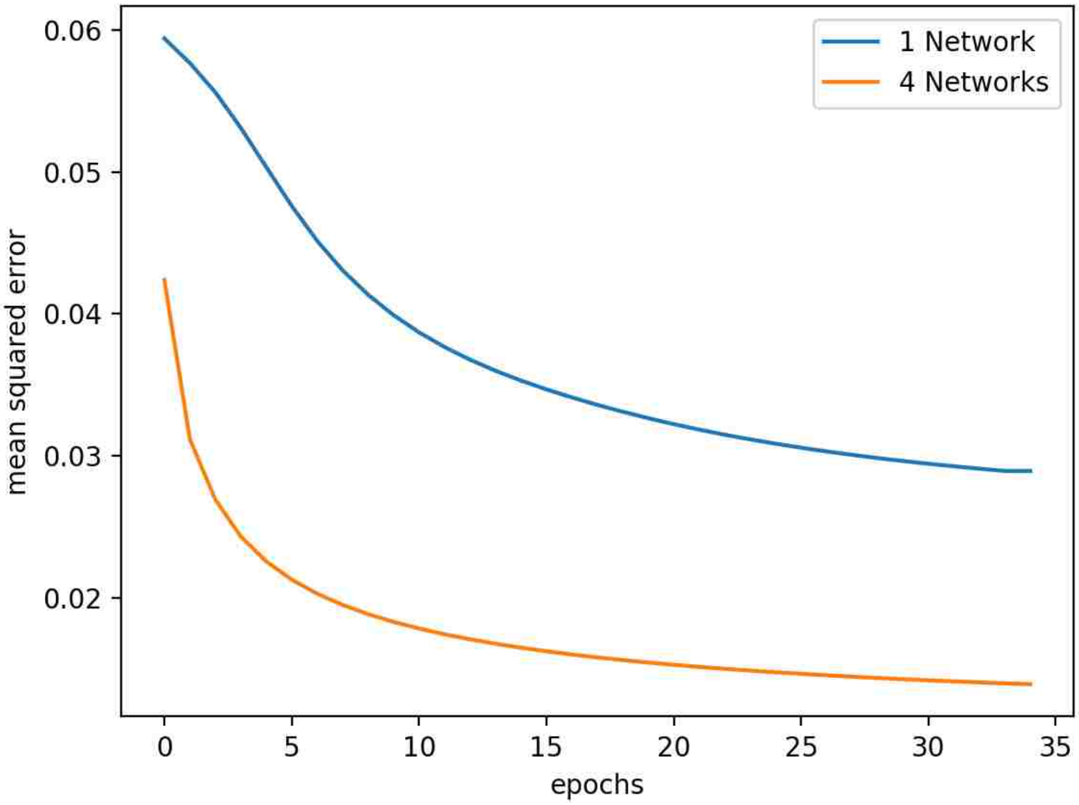

Figure 5.3: Performance comparison of different models on the fashion CIFAR-10 test dataset.

5.3.4 CIFAR-100

Table 5.4 and Figure 5.4 show the performance comparison of different models on the fashion CIFAR-100 test dataset. The experiment shows that the performance gets better when the problem is decentralized and more networks are used to solve the problem.

2 layered CNN (64-64 feature maps) derived from the official tensorflow github example on CIFAR-10 is selected as the feature extractor.

Table 5.4: Mean Squared Error (MSE) comparison of different models on the CIFAR-100 dataset for first 35 epochs.

Model	MSE (within 35 epochs)
Baseline (1 network)	0.0382
4 networks	0.0150

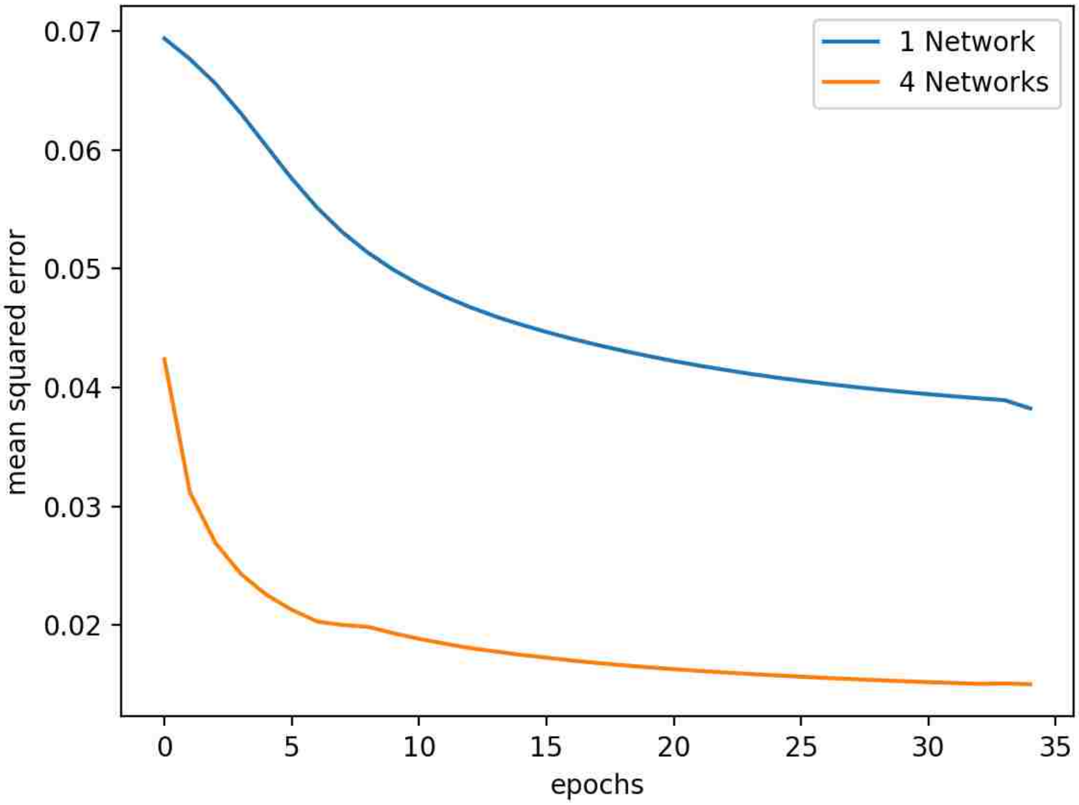

Figure 5.4: Performance comparison of different models on the fashion CIFAR-100 test dataset.

5.4 Further Improvements

The methods to further improve the performance of the proposed approach are listed below.

5.4.1 ZCA Whitening/Sphering

ZCA whitening is a rough model of how the biological eye (the retina) processes images (through retinal neurons). ZCA whitening also tries to add noise robustness to the data.

5.4.2 Increasing the Number of Networks

Dividing the output to more number of networks might give better results, as the output size of the network will be reduced.

Chapter 6

Conclusion

We propose a novel framework for boosting CNN classification performance via minimizing the covariance between the feature maps while training the model. This explicitly makes the learned features to be less redundant.

It is shown that decentralization of the output image to more networks; proper distribution of workload can increase the network performance while the network is used to solve the pointwise regression problem, or, image generation.

The functions involved in the generation of the data are based on physical functions and these follow the properties like compositionality, locality, symmetry, log probability. This very fact makes computations cheap. Though a huge number of possible instances/outcomes, the number of favorable instances/outcomes is surprisingly very less. For example, if we consider a 1000x1000 pixel image, each pixel can take a value from the pool of 256 (0-255) possible values. So, $255^{1000000}$ possible instances are there, which is way larger than the number of atoms in our known, observable universe (10^{78} to 10^{82}). The images we deal with barely make up to that number. It is because of the laws of nature having the property to be describable using physical equations. Physical properties governing the laws of nature limit the number of favorable instances. We have a low order of polynomial in nature because the properties of nature can be represented by a Hamiltonian polynomial of low order d. This extremely awe-inspiring and magnificent behavior of nature is the foremost factor that makes the machine learning algorithms work, given good enough data and required computing power.

Deep architecture has more representation power than a shallow one. In case of deep learning, neural network with 1 hidden layer is considered as the shallow network, while neural network with more than 1 hidden layers is considered as the deep network. Corresponding to the fractal nature of the universe (infinite levels of pattern representations as we go towards either the microlevels or the macrolevels), the physical processes generating the data comprises of steps of internal processes to get the output. Deep architecture provides such an opportunity.

Future works for image classification task include performing ZCA whitening/sphering during the data pre-processing step, pair formation for covariance minimization with replacement, and voting among the networks. Similarly, future works for enhancing image resolution includes ZCA whitening/sphering, and increasing the number of networks.

Appendix A

Artificial Neural Network

Consider $\boldsymbol{X} = \{\boldsymbol{x_j}\}, j = 1, 2, 3, ..., N$.

where, N is the number of training data. $\boldsymbol{x_j}$ is a D dimensional input feature vector. $\boldsymbol{X}$ consists of N 'D dimensional' feature vectors.

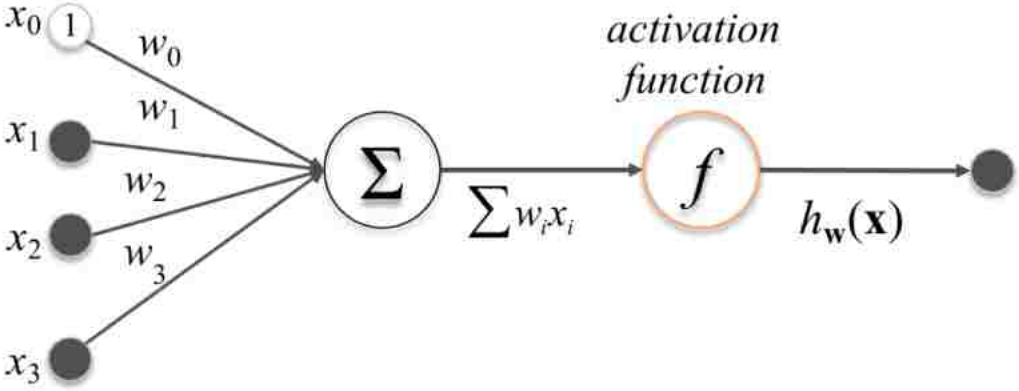

Figure A.1: Working of single neuron/perceptron, or a logit unit with D=3

Let, $\boldsymbol{x} = \boldsymbol{x_j} = \{x_1, x_2, ..., x_D\}$, and $\boldsymbol{w} = \{w_1, w_2, ..., w_D\}$ be the weight vector that projects the feature vector to produce a new scalar value, represented by Σ or simply S. Adding bias to the sum will not enforce the function that the neuron is going to learn to always pass through the origin in hyperplane; for example, considering a simple function of line in slope-intercept form, $y = f(x) = slope * x + y\_intercept * 1$, intercept acts as the bias. If the intercept is zero, the line always have to pass through the origin, but this is not a necessary condition for a line, as there are infinitely as many lines that do not pass through the origin. So, the $\boldsymbol{x} = \{x_0, x_1, x_2, ..., x_D\}$, and $\boldsymbol{w} = \{w_0, w_1, w_2, ..., w_D\}, x_0 = 1, w_0$ also represented as b is the bias.

$S = \sum_{i=0}^{D} w_i x_i$

First of all, let us deal with the case without activation. S is the predicted result. Let T be the truth. Then the residual error is $S - T$. The squared error is $(S - T)^2$. We take the squared error because it is a derivable convex function (a requirement for gradient descent).

Let, the cost/loss/error function be

$$E = \frac{1}{2}(S - T)^2$$

The reason for taking $\frac{1}{2}$ will be shown later.

We want to minimize the error, in order to improve the performance of the model. The error is because of the weights being incorrectly set. The task is to find the weights that try to minimize E to zero; the model should learn the weights in such a way that the minima of the function E is reached. The derivative of any function gives minima/maxima of that function; in our case the function is convex, so minima. Or, in other words, derivative gives the slope at a certain point of the function/equation. With the help of gradient descent, following the opposite path of derivative, the model reaches slowly and taking sharp steps to the minimum value of the cost function. For the case of the convex function, if the minimum point is crossed, the slope is positive, so taking reverse direction will make a backward turn. While, if we have not yet reached the minimal point, the slope is negative; following the reverse direction of the derivative will actually, in this case, make a forward move.

Now, at each step of gradient descent, we calculate the gradients of the weights and update the weights.

$$w_{i_{new}} = w_{i_{old}} - learning\_rate * \nabla w_i$$

Initially, the weights are set randomly between some range or using normal distribution, or any other sort of distributions.

The calculation of gradients is done using partial differentiation.

$$\nabla w_0 = \frac{\partial E}{\partial w_0} = \frac{\partial E}{\partial S}\frac{\partial S}{\partial w_0} = \frac{1}{2} * (2 * (S - T)) * (1) = (S - T) * 1$$

Similarly,

$$\nabla w_1 = \frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial S}\frac{\partial S}{\partial w_1} = \frac{1}{2} * (2 * (S - T)) * (x_1) = (S - T) * x_1$$

$$\nabla w_2 = \frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial S}\frac{\partial S}{\partial w_2} = \frac{1}{2} * (2 * (S - T)) * (x_2) = (S - T) * x_2$$

$$\nabla w_3 = \frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial S}\frac{\partial S}{\partial w_3} = \frac{1}{2} * (2 * (S - T)) * (x_3) = (S - T) * x_3$$

.

.

.

.

$$\nabla w_D = \frac{\partial E}{\partial w_D} = \frac{\partial E}{\partial S}\frac{\partial S}{\partial w_D} = \frac{1}{2} * (2 * (S - T)) * (x_D) = (S - T) * x_D$$

This shows that the second-last part of the gradient is the residual error at that node/neuron, and the last part is the feature which the weight projects.

This is actually the linear regression. Figure A.2 below shows the use of linear regression with the Fisher Iris Setosa dataset, wherby it tries to predict the sepal length given sepal width; sepal length being the truth(T)/y, and sepal width (x_1) being feature. D=1, N=50, $\boldsymbol{x} = \{x_0 x_1\}$ $x_0 = 1$ is for the bias. . In this particular example, $learning\_rate = 0.1$. Initially all the weights are set to zero. At the end of the learning process, the weights learned $w_0 = 2.3614$, and $w_1 = 0.7706$. With the help of learned weights, given a sepal width (x), corresponding sepal length (y) is given by the function $y = S = f(x) = w_0 * 1 + w_1 * x = 2.3614 + 0.7706 * x$. That is, the dot product of weight vector with feature vector (with bias added) gives the prediction. The line in the diagram is the equation represented by S/y of the trained model.

First 10 out of the 50 data is shown in Table A.1.

Now, let us take the case with activation. $Z = h_w(\boldsymbol{S})$ is the predicted result. S acts as the new input $\boldsymbol{x}$, in this case, x. Now, the residual error is $Z - T$. The squared error is $(Z - T)^2$. As we work with the squared error, the cost function is $E = \frac{1}{2}(Z - T)^2$. We take half so that the 2 we will get from the derivative of $(Z - T)^2$ can cancel each other. We are merely avoiding the division operation.

The calculation of the gradients is done as shown.

$$\nabla w_0 = \frac{\partial E}{\partial w_0} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial S}\frac{\partial S}{\partial w_0} = \frac{1}{2} * (2 * (Z - T)) * f'(S) * (1) = (Z - T) * f'(S) * 1$$

Similarly,

Table A.1: Fisher Iris Setosa Dataset.

N	Sepal Length (y or truth)	Sepal Width (x)
1	5.1	3.5
2	4.9	3
3	4.7	3.2
4	4.6	3.1
5	5	3.6
6	5.4	3.9
7	4.6	3.4
8	5	3.4
9	4.4	2.9
10	4.9	3.1

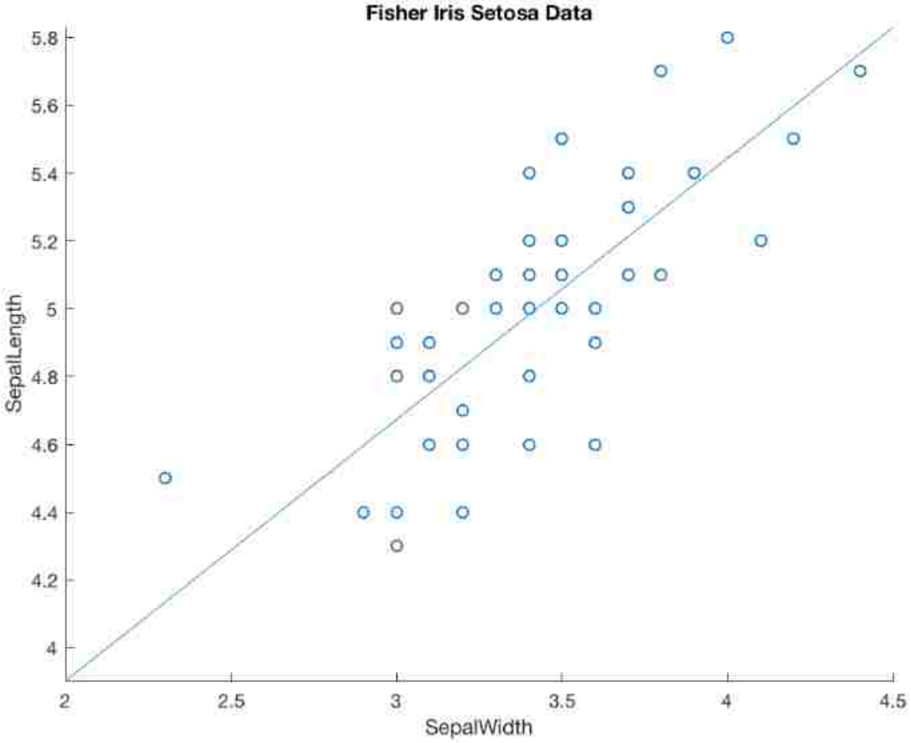

Figure A.2: Linear regression with fisher iris setosa dataset.

$$\nabla w_1 = \frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial S}\frac{\partial S}{\partial w_1} = \frac{1}{2} * (2 * (Z - T)) * f'(S) * (x_1) = (Z - T) * f'(S) * x_1$$

$$\nabla w_2 = \frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial S}\frac{\partial S}{\partial w_2} = \frac{1}{2} * (2 * (Z - T)) * f'(S) * (x_2) = (Z - T) * f'(S) * x_2$$

$$\nabla w_3 = \frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial S}\frac{\partial S}{\partial w_3} = \frac{1}{2} * (2 * (Z - T)) * f'(S) * (x_3) = (Z - T) * f'(S) * x_3$$

$$.$$

$$.$$

$$.$$

$$.$$

$$\nabla w_D = \frac{\partial E}{\partial w_D} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial S}\frac{\partial S}{\partial w_D} = \frac{1}{2} * (2 * (Z - T)) * f'(S) * (x_D) = (Z - T) * f'(S) * x_D$$

This shows that with activation, the third-last part of the gradient is the residual error at that node/neuron, the second-last part is the derivative of post-synaptic value with respect to pre-synaptic value, and the last part is the feature which the weight projects.

$f'(S)$ is the partial derivative of post-synaptic value with respect to pre-synaptic value. Partial derivative refers to the derivative taken with respect to one variable when multiple variables are involved in the system. All the individual values which do not contain the specific term with respect to which the derivative is being taken yield zero; their value does not have an impact and appear as constant to the system in which we are currently existing.

For example, in the case of ReLU (Rectified Linear Unit; rectifier is used to block the signal from specific bands, and ReLU is linear unit: if $x > 0, Z = ReLU(x) = x$ and $ReLU'(x) = 1$; else Z = ReLU(x) = relu constant * x and ReLU'(x) = relu constant; relu constant is some small value, $0 <= relu\ constant < 1$, it bears very close relation to the learning rate. In case of relu (not leaky relu), $relu\ constant = 0$. But this will not work in most of the conditions, as there is possibility of all the weights being negative, and if such situation arises, the network will neither forward any information, nor will correct itself. Linear activation, that is, $Z = Linear(x) = x$, is just a relu activation with $relu\ constant = 1$, that is, entire available/visible field is accepted by the rectifier.

In case of sigmoid activation function, Z = sigmoid(x) = $\frac{1}{1+e^{-x}}$ and sigmoid'(x) = Z(1-Z). For hyperbolic tangent (tanh), Z = $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ and tanh'(x) = $1 - Z^2$.

Taylor series expansion of f(x) about 0 is given by

$$f(x) = f(0) + \frac{x}{1!}f'(0) + \frac{x^2}{2!}f''(0) + ... + \frac{x^k}{k!}f^k(0) + ...$$

So,

$$e^x = e^0 + \frac{x}{1!}e^0 + \frac{x^2}{2!}e^0 + ...$$

$f'(e^{cx}) = ce^{cx}$, and $f'(e^{-cx}) = -ce^{-cx}$

Following the Taylor series expansion,

$$e^{\diamond} = 1 + \frac{\diamond}{1!} + \frac{\diamond^2}{2!} + \frac{\diamond^3}{3!} + ...$$

This equation gives the value of any power of e. Taking the sum up to a considerable point is the feasible solution.

It is obvious that in this case, with the activation applied to the weighted sum, the first part of the gradient is the residual error, the second part is the partial derivative of the post-activated/synaptic value with respect to pre-activated value, and the third part is the feature which the weight projects.

This is actually the logistic regression. Figure A.3 Fisher Iris Versicolor/Virginica dataset is shown; starting from this sort of representation, the model has to be able to learn some sort of representation that separates the two classes as farthest as possible.

Few 10 out of the 100 data is shown in Table A.2. With a learning rate of 0.1 and using sigmoid activation function, the accuracy rate of 95% can be easily reached. Starting from zero valued weights, the trained weights being $w_0 = -27.8236$, $w_1 = -4.1712$, $w_2 = 10.9724$. $x_0 = 1$. If the predicted value is greater than or equal to 0.5, it belongs to class 1, else it belongs to class 0. sigmoid activation function limits the value between 0 and 1. If input x tends to infinity, sigmoid returns 1, that is, y=1. If x tends to -infinity, y=0. At x=0, y=0.5. We use this feature of sigmoid function to decide the class to be 0 or 1. $y >= 0.5$ or $x >= 0$ means the instance belongs to the class 1, and $y < 0.5$ or $x < 0$ means the instance belongs to the class 0. In fact, the decision boundary here is linear, as the line y=0.5 (it is used in practice) or the line x=0 (x-axis) can be used as the decision boundary between the instances belonging to different categories.

The classes are changed to 0 and 1. It is a binary classification problem.

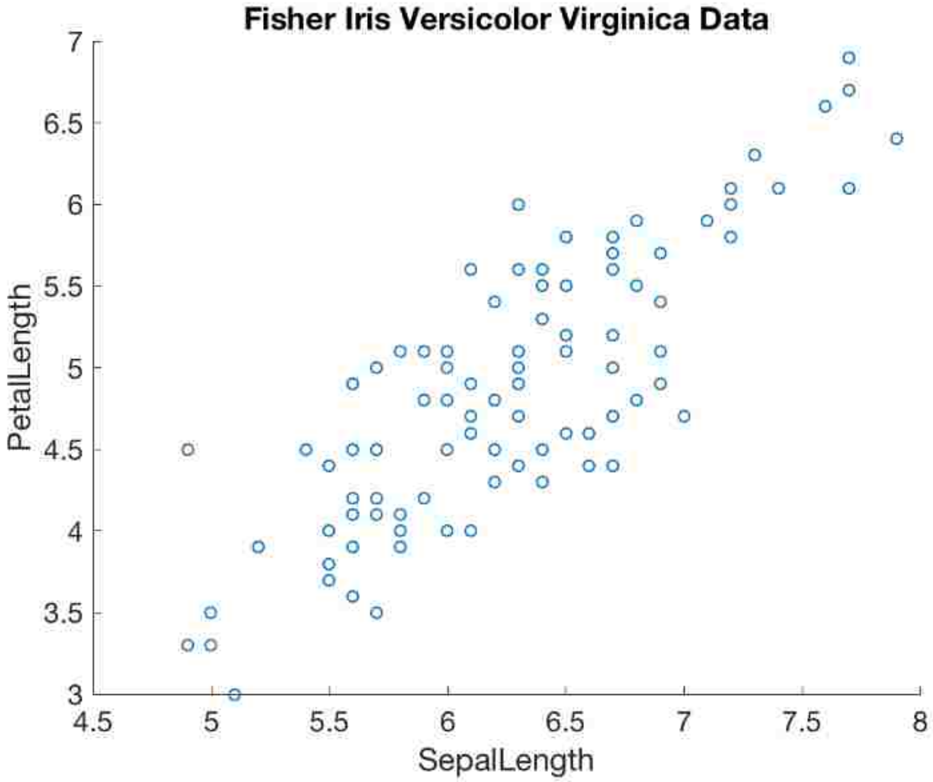

Figure A.3: Fisher Iris: Versicolor/Virginica dataset.

Table A.2: Fisher Iris: Versicolor/Virginica dataset

N	sepal length(x_1)	petal length(x_2)	class (1 for Iris Versicolor, 2 for Iris Virginica)
1	6.2	4.3	1
2	5.1	3	1
3	5.7	4.1	1
4	6.7	5.6	2
5	6.9	5.1	2
6	5.8	5.1	2
7	6.8	5.9	2
8	6.7	5.7	2
9	6.7	5.2	2
10	6.3	5	2

Logistic regression (LR) can also solve some N-classes problem. It achieves this by creating N different models; each model takes the instances of it as class 1 and the instances of all other classes as class 0. It is referred to as the 1-vs-all model.

Logistic regression can solve the case of boolean 'AND' and boolean 'OR' as when the four possible cases are plotted in the 2-D cartesian coordinate system, the instances belonging to class '1' and class '0' can be separated by using a straight line. As logistic regression is searching for a linear decision boundary, it is unable to solve the case of 'XOR' (see Fig. A.4).

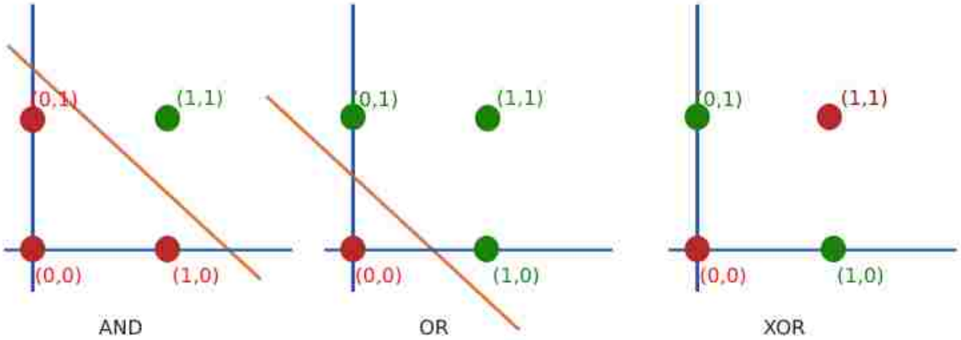

Figure A.4: Logistic Regression, and AND, OR, XOR boolean functions

Nevertheless, LR is capable of getting 100% accuracy on the XOR problem with a simple transformation to the feature space, by adding new feature x1*x2 to the existing features x1 and x2. The addition of the nonlinear feature assists linear regression in learning a decision boundary; the decision boundary is linear in regard to the features, but not necessarily in the original data space. Though the decision boundary is linear in nature, the features need not be linearly related to each other, and also each feature in individual dataspace need not be a linear function.

Another approach of solving xor without adding any new feature is to use a simple neural network; compose many logit unit or neuron, where each neuron is connected to some other neurons/inputs and passes its activated value to some other neurons/outputs (Fig. A.5). The superscripts represent the layer number, and the subscript denotes the neuron number.

Table A.3: Dataset for boolean 'XOR'

N	bias	x_1	x_2	Truth (T)
1	1	0	0	0
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

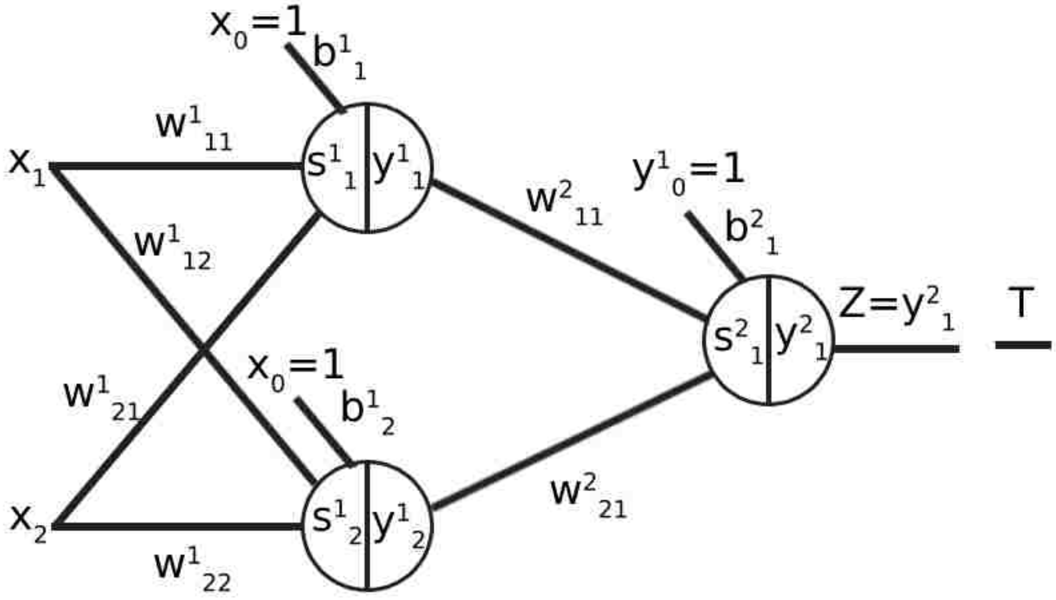

Figure A.5: Simple Neural Network for XOR

The residual error is $Z - T$, and the cost function is $E = \frac{1}{2}(Z - T)^2$. The weights be corrected at each iteration of gradient descent are w^1_{11}, w^1_{12}, w^1_{21}, w^1_{22}, b^1_1 or w^1_{01}, b^1_2 or w^1_{02}, w^2_{11}, w^2_{21}, and b^2_1 or w^2_{01}.

The calculation of the gradients is done as shown.

For the neurons (in this case, 1) of the last/output layer,

$$\nabla w^2_{01} = \nabla b^2_1 = \frac{\partial E}{\partial b^2_1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s^2_1}\frac{\partial s^2_1}{\partial b^2_1} = \frac{1}{2} * (2 * (Z - T)) * f'(s^2_1) * (1) = (Z - T) * f'(s^2_1) * 1 = \delta^2_1 * 1$$

where, $\delta^2_1 = (Z - T) * f'(s^2_1)$. So,

$$\nabla w^2_{11} = \frac{\partial E}{\partial w^2_{11}} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s^2_1}\frac{\partial s^2_1}{\partial w^2_{11}} = \frac{1}{2} * (2 * (Z - T)) * f'(s^2_1) * (1) = (Z - T) * f'(s^2_1) * y^1_1 = \delta^2_1 * y^1_1$$

$$\nabla w^2_{21} = \frac{\partial E}{\partial w^2_{21}} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s^2_1}\frac{\partial s^2_1}{\partial w^2_{21}} = \frac{1}{2} * (2 * (Z - T)) * f'(s^2_1) * (1) = (Z - T) * f'(s^2_1) * y^1_2 = \delta^2_1 * y^1_2$$

Now, for the neurons of the second last layer.

For the top neuron.

$$\nabla w_{01}^1 = \nabla b_1^1 = \frac{\partial E}{\partial b_1^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial b_1^1} = (Z - T) * f'(s_1^2) * w_{11}^2 * f'(s_1^1) * 1 = \delta_1^2 * w_{11}^2 * f'(s_1^1) * 1 = \delta_1^1 * 1$$

where, $\delta_1^1 = \delta_1^2 * w_{11}^2 * f'(s_1^1) = (Z - T) * f'(s_1^2) * w_{11}^2 * f'(s_1^1)$. So,

$$\nabla w_{11}^1 = \frac{\partial E}{\partial w_{11}^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{11}^1} = (Z - T) * f'(s_1^2) * w_{11}^2 * f'(s_1^1) * x_1 = \delta_1^1 * x_1$$

$$\nabla w_{21}^1 = \frac{\partial E}{\partial w_{21}^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{21}^1} = (Z - T) * f'(s_1^2) * w_{11}^2 * f'(s_1^1) * x_2 = \delta_1^1 * x_2$$

For the bottom neuron,

$$\nabla w_{02}^1 = \nabla b_2^1 = \frac{\partial E}{\partial b_2^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_2^1}{\partial s_2^1}\frac{\partial s_2^1}{\partial b_2^1} = (Z - T) * f'(s_1^2) * w_{21}^2 * f'(s_2^1) * 1 = \delta_1^2 * w_{21}^2 * f'(s_2^1) * 1 = \delta_2^1 * 1$$

where, $\delta_2^1 = \delta_1^2 * w_{21}^2 * f'(s_2^1) = (Z - T) * f'(s_1^2) * w_{21}^2 * f'(s_2^1)$. So,

$$\nabla w_{12}^1 = \frac{\partial E}{\partial w_{12}^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_2^1}{\partial s_2^1}\frac{\partial s_2^1}{\partial w_{12}^1} = (Z - T) * f'(s_1^2) * w_{21}^2 * f'(s_2^1) * x_1 = \delta_2^1 * x_1$$

$$\nabla w_{22}^1 = \frac{\partial E}{\partial w_{22}^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_2^1}{\partial s_2^1}\frac{\partial s_2^1}{\partial w_{22}^1} = (Z - T) * f'(s_1^2) * w_{21}^2 * f'(s_2^1) * x_2 = \delta_2^1 * x_2$$

It is clear from these equations that the δs of a layer is dependent upon the delta of the previous layer. Each neuron along with output has its δ. Doing so, we can reuse the value of δ within a layer, as well as for the calculation of δs of the previous layer. It uses the concept of dynamic programming (memory vs speed). The other benefit it provides is the generalization of the working of the neurons. Wherever the neuron is present in the network, the weights associated with it, at each iteration of gradient descent, can be corrected with the help of δ of that neuron (fig. A.6). The error of the network is because of the error of individual neurons.

From above equations, we have,

$$\nabla w_{01}^1 = \nabla b_1^1 = \frac{\partial E}{\partial b_1^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial b_1^1} = (Z - T) * f'(s_1^2) * w_{11}^2 * f'(s_1^1) * 1 = \delta_1^2 * w_{11}^2 * f'(s_1^1) * 1 = \delta_1^1 * 1$$

$$\nabla w_{01}^1 = \nabla b_1^1 = \frac{\partial E}{\partial b_1^1} = \frac{\partial E}{\partial Z}\frac{\partial Z}{\partial s_1^2}\frac{\partial s_1^2}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial b_1^1} = \frac{\partial E}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial b_1^1} = \frac{\partial E}{\partial y_1^1} * f'(s_1^1) * 1 = \delta_1^2 * w_{11}^2 * f'(s_1^1) * 1$$

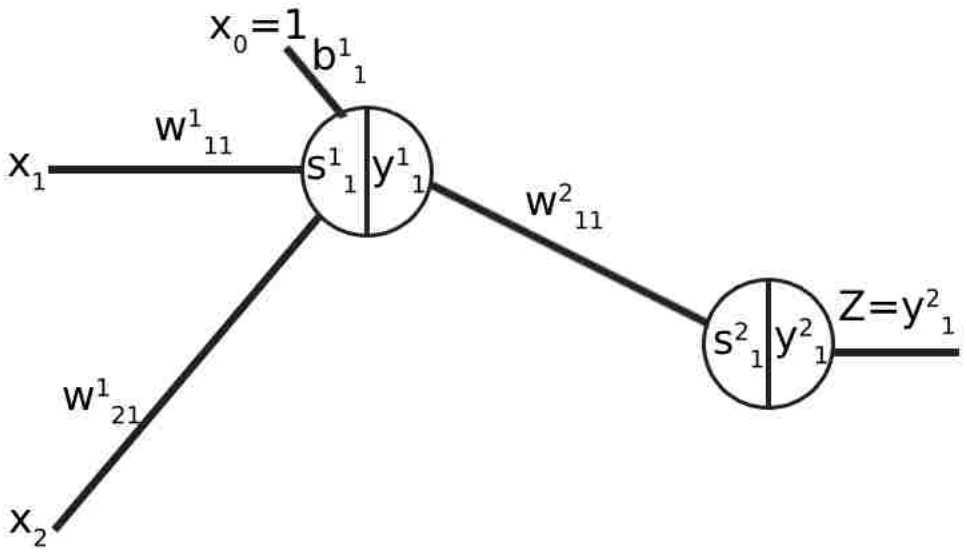

Figure A.6: Simple Neural Network for XOR: showing the first neuron of first layer

$\frac{\partial y_1^1}{\partial s_1^1}$ is the partial derivative of post-synaptic value with respect to pre-synaptic value, and it can be calculated. Also, $\frac{\partial s_1^1}{\partial b_1^1}$ is just the feature which the current weight (in case of bias, 1) is projecting.

$\frac{\partial E}{\partial y_1^1} = \sum_{i=1}^{n_o} w_{1i}^2 * \delta_1^2$, n_o is the number of neurons the current neuron passes its information to during the time of forward propagation. that is, the weighted sum of δs; each δ is weighted by the weight of the link that had impact on that specific output which finally effected the specific δ.

Now, let us take the case of a neural network with two outputs: one for AND, and another for XOR (fig. A.7). The data for the model is shown in table A.4.

Table A.4: Dataset for boolean 'XOR' and 'AND'

N	bias	x_1	x_2	Truth for AND (T_1)	Truth for XOR (T_2)
1	1	0	0	0	0
2	1	0	1	0	1
3	1	1	0	0	1
4	1	1	1	1	0

Forward propagation is done first.

$$s_1^1 = x_0 * w_{01}^1 + x_1 * w_{11}^1 + x_2 * w_{21}^1$$

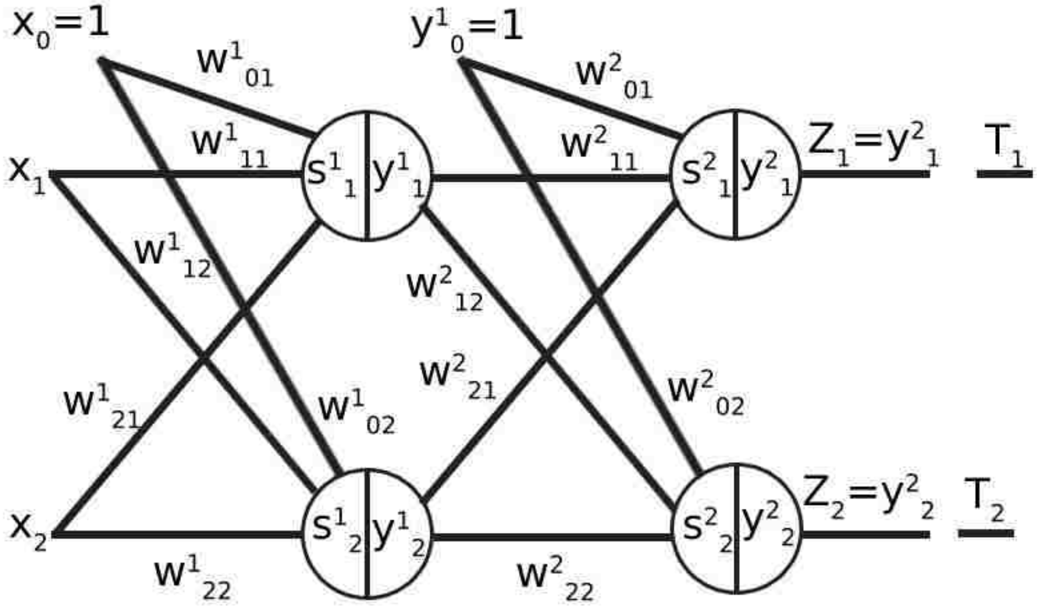

Figure A.7: Simple Neural Network for XOR and AND

$$y_1^1 = activation(s_1^1)$$

$$s_2^1 = x_0 * w_{02}^1 + x_1 * w_{12}^1 + x_2 * w_{22}^1$$

$$y_2^1 = activation(s_2^1)$$

Now, y_1^1, and y_2^1 with $y_0^1 = 1$ for bias added, act as the inputs to the output layer.

$$s_1^2 = y_0^1 * w_{01}^2 + y_1^1 * w_{11}^2 + y_2^1 * w_{21}^2$$

$$Z_1 = y_1^2 = activation(s_1^2)$$

$$s_2^2 = y_0^1 * w_{02}^2 + y_1^1 * w_{12}^2 + y_2^1 * w_{22}^2$$

$$Z_2 = y_2^2 = activation(s_2^2)$$

The residual error for AND is $Z_1 - T_1$, and for XOR is $Z_2 - T_2$. The cost function $E_1 = \frac{1}{2}(Z_1 - T_1)^2$, and $E_2 = \frac{1}{2}(Z_2 - T_2)^2$, whereby the total error $E = E_1 + E_2$. The weights be to corrected at each iteration of gradient descent are w_{01}^1, w_{02}^1, w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1, w_{01}^2, w_{02}^2, w_{11}^2, w_{12}^2, w_{21}^2, w_{22}^2.

The calculation of the gradients is done as shown.

$$\nabla w_{01}^2 = \frac{\partial E}{\partial w_{01}^2} = \frac{\partial E_1}{\partial w_{01}^2} + \frac{\partial E_2}{\partial w_{01}^2}$$

$$\frac{\partial E_1}{\partial w_{01}^2} = \frac{\partial E_1}{\partial Z_1}\frac{\partial Z_1}{\partial s_1^2}\frac{\partial s_1^2}{w_{01}^2} = (Z_1 - T_1) * f'(s_1^2) * y_0^1$$

If the activation function used was tanh, $f'(s_1^2) = 1 - (Z_1)^2$.

As E_2 is not affected by w_{01}^2, that is, E_2 will not contain any term involving w_{01}^2,

$$\frac{\partial E_2}{\partial w_{01}^2} = 0$$

$$\nabla w_{01}^2 = (Z_1 - T_1) * f'(s_1^2) * y_0^1 = \delta_1^2 * y_0^1$$

$$\nabla w_{11}^2 = (Z_1 - T_1) * f'(s_1^2) * y_1^1 = \delta_1^2 * y_1^1$$

$$\nabla w_{21}^2 = (Z_1 - T_1) * f'(s_1^2) * y_2^1 = \delta_1^2 * y_2^1$$

$$\nabla w_{02}^2 = (Z_2 - T_2) * f'(s_2^2) * y_0^1 = \delta_2^2 * y_0^1$$

$$\nabla w_{12}^2 = (Z_2 - T_2) * f'(s_2^2) * y_1^1 = \delta_2^2 * y_1^1$$

$$\nabla w_{22}^2 = (Z_2 - T_2) * f'(s_2^2) * y_2^1 = \delta_2^2 * y_2^1$$

where, $\delta_1^2 = (Z_1 - T_1) * f'(s_1^2)$, and $\delta_2^2 = (Z_2 - T_2) * f'(s_2^2)$

Now, for the first hidden layer,

$$\nabla w_{01}^1 = \frac{\partial E}{\partial w_{01}^1} = \frac{\partial E_1}{\partial w_{01}^1} + \frac{\partial E_2}{\partial w_{01}^1}$$

$$\frac{\partial E_1}{\partial w_{01}^1} = \frac{\partial E_1}{\partial Z_1}\frac{\partial Z_1}{\partial s_1^2}\frac{\partial s_1^2}{y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{01}^1} = (Z_1 - T_1) * f'(s_1^2) * w_{11}^2 * f'(s_1^1) * x_0$$

$$\frac{\partial E_2}{\partial w_{01}^1} = \frac{\partial E_2}{\partial Z_2}\frac{\partial Z_2}{\partial s_2^2}\frac{\partial s_2^2}{y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{01}^1} = (Z_2 - T_2) * f'(s_2^2) * w_{12}^2 * f'(s_1^1) * x_0$$

$$\frac{\partial E}{\partial w_{01}^1} = \frac{\partial E_1}{\partial Z_1}\frac{\partial Z_1}{\partial s_1^2}\frac{\partial s_1^2}{y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{01}^1} + \frac{\partial E_2}{\partial Z_2}\frac{\partial Z_2}{\partial s_2^2}\frac{\partial s_2^2}{y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{01}^1}$$

$$\frac{\partial E}{\partial w_{01}^1} = \left(\frac{\partial E_1}{\partial Z_1}\frac{\partial Z_1}{\partial s_1^2}\frac{\partial s_1^2}{y_1^1} + \frac{\partial E_2}{\partial Z_2}\frac{\partial Z_2}{\partial s_2^2}\frac{\partial s_2^2}{y_1^1}\right)\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{01}^1} = \frac{\partial E}{\partial y_1^1}\frac{\partial y_1^1}{\partial s_1^1}\frac{\partial s_1^1}{\partial w_{01}^1}$$

$$\nabla w_{01}^1 = \big((Z_1 - T_1) * f'(s_1^2) * w_{11}^2 + (Z_2 - T_2) * f'(s_2^2) * w_{12}^2\big) * f'(s_1^1) * x_0 = \nabla y_1^1 * f'(s_1^1) * x_0$$

$$\nabla w_{01}^1 = \big(\delta_2^1 * w_{11}^2 + \delta_2^2 * w_{12}^2\big) * f'(s_1^1) * x_0 = \nabla y_1^1 * f'(s_1^1) * x_0 = \delta_1^1 * x_0$$

$$\nabla w_{11}^1 = \big(\delta_2^1 * w_{11}^2 + \delta_2^2 * w_{12}^2\big) * f'(s_1^1) * x_1 = \nabla y_1^1 * f'(s_1^1) * x_1 = \delta_1^1 * x_1$$

$$\nabla w_{21}^1 = \big(\delta_2^1 * w_{11}^2 + \delta_2^2 * w_{12}^2\big) * f'(s_1^1) * x_2 = \nabla y_1^1 * f'(s_1^1) * x_2 = \delta_1^1 * x_2$$

$$\nabla w_{02}^1 = \big(\delta_2^1 * w_{21}^2 + \delta_2^2 * w_{22}^2\big) * f'(s_2^1) * x_0 = \nabla y_2^1 * f'(s_2^1) * x_0 = \delta_2^1 * x_0$$

$$\nabla w_{12}^1 = \big(\delta_2^1 * w_{21}^2 + \delta_2^2 * w_{22}^2\big) * f'(s_2^1) * x_1 = \nabla y_2^1 * f'(s_2^1) * x_1 = \delta_2^1 * x_1$$

$$\nabla w_{22}^1 = \big(\delta_2^1 * w_{21}^2 + \delta_2^2 * w_{22}^2\big) * f'(s_2^1) * x_2 = \nabla y_2^1 * f'(s_2^1) * x_2 = \delta_2^1 * x_2$$

The computation of ∇y_j^is is similar to the computation of pre-synaptic values (s_j^is) during the forward propagation, but in a backward manner and δs take the role of input.

A.1 Backpropagation Algorithm

A single pass of the backpropagation algorithm is summarized as below.

A.1.1 Forward Propagation

Feedforward passes on input $\boldsymbol{x}$ to compute all the activations y_j^L of the final layers, j = 1, 2, ..., m_L. m_L is the number of neurons in the output layer.

A.1.2 Backward Propagation

For each output unit j, compute:

$$\delta_j^L = (y_j^L - T_j) * f'(s_j^L)$$

Then, for l = $L - 1$, $L - 2$, ..., 1 compute:

$$\delta_i^l = (\sum_{j=1}^{m_{l+1}} w_{ij}^{l+1} \delta_j^{l+1}) * f'(s_i^l)$$

where, i = 1, 2, ..., m_l. m_l is the number of neurons in the current layer, and m_{l+1} is the number of neurons in the following or next layer.

L is the total number of layers including L-1 hidden layers and an output layer. m_{l+1} is the number of neurons in the $(l+1)^{th}$ layer.

After that, compute the partial derivatives of the cost with respect to every weight (including bias).

$$\nabla w_{ij}^{l} = y_i^{l-1} \delta_j^l$$

$y_0^{l-1} = 1$ for bias, and $y_i^{l-1} = x_i$ for the first hidden layer.

Finally, update the weights with some learning rate.

A.2 Proof by Induction

Fig. A.9 and A.8 show the ANN with L layers (L-1 hidden layers, and an output layer). The number of neurons in output layer is m_L, and the number of features/input to the ANN is D. All the hidden layers can have different number of neurons (m_1, m_2, ..., m_{L-1}); generally between D and m_L, and in the decreasing order from D to m_L. Only a few weights, pre-activated values, and post-activated values are shown in the diagram as it will look cluttered and unclear. Remaining weights and activation values can be shown in a similar fashion. Superscript is the layer number, the subscript is the neuron number in a layer. In the case of weights w_{ij} i is the neuron number of from-neuron and j is the neuron number of to-neuron.

A.2.1 Output/Final layer

The residual error for neuron 1 of output/final layer is $y_1^L - T_1$. Similarly for neuron 2, 3, ..., m_L are $y_2^L - T_2$, $y_3^L - T_3$, ..., $y_{m_L}^L - T_{m_L}$. The cost functions are $E_1 = \frac{1}{2}(y_1^L - T_1)^2$, $E_2 = \frac{1}{2}(y_2^L - T_2)^2$, $E_3 = \frac{1}{2}(y_3^L - T_3)^2$, ..., $E_{m_L} = \frac{1}{2}(y_{m_L}^L - T_{m_L})^2$; whereby the total error $E = E_1 + E_2 + E_3 + ... + E_{m_L}$. The weights be to corrected at each iteration of gradient descent for this layer are w_{ij}^Ls where $0 < i < m_{L-1}$ (number of neurons in last hidden layer) and $1 < j < m_L$(number of neurons in last/output layer).

The calculation of the gradients is done as shown.

$$\nabla w_{01}^L = \frac{\partial E}{\partial w_{01}^L} = \frac{\partial E_1}{\partial w_{01}^L} + \frac{\partial E_2}{\partial w_{01}^L} + \frac{\partial E_3}{\partial w_{01}^L} + ... + \frac{\partial E_{m_L}}{\partial w_{01}^L}$$

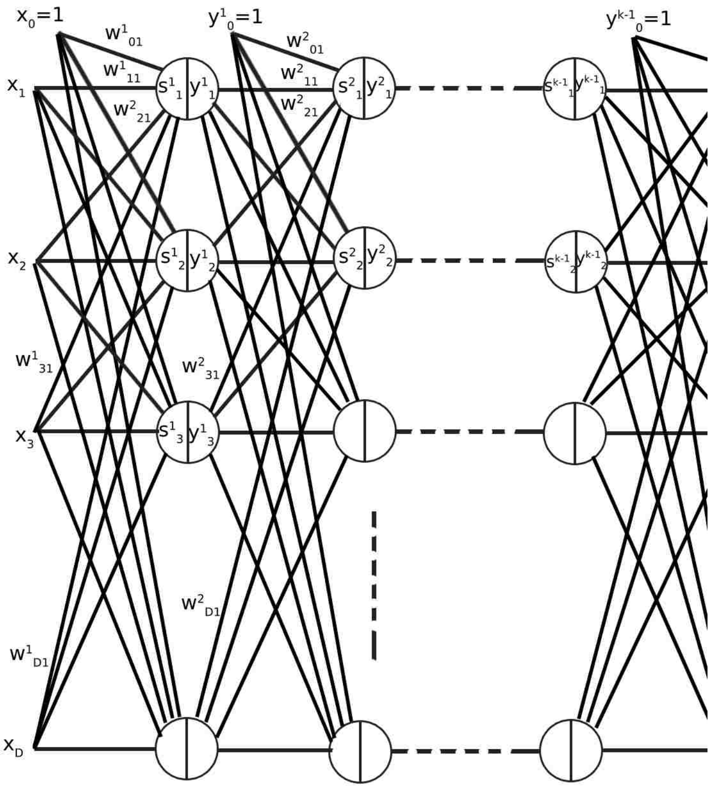

Figure A.8: Initial Layers of ANN containing L layers (L-1 hidden layers and 1 output layer)

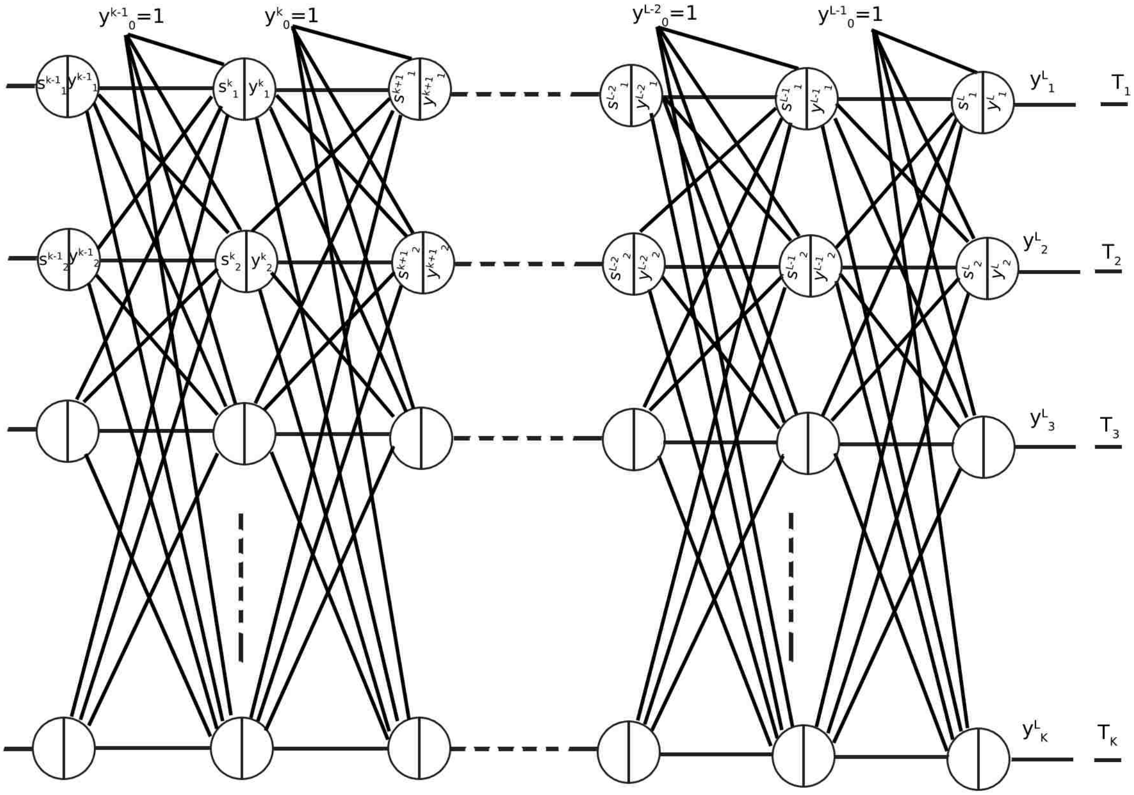

Figure A.9: Final Layers of ANN containing L layers (L-1 hidden layers and 1 output layer)

$$\frac{\partial E_1}{\partial w_{01}^L} = \frac{\partial E_1}{\partial y_1^L}\frac{\partial y_1^L}{\partial s_1^L}\frac{\partial s_1^L}{w_{01}^L} = (y_1^L - T_1) * f'(s_1^L) * y_0^{L-1}$$

As E_2, E_3, ..., E_{m_L} are not affected by w_{01}^L, that is, E_2, E_3, ..., E_{m_L} will not contain any term involving w_{01}^L,

$$\frac{\partial E_2}{\partial w_{01}^L} = 0$$

And, so is the case with the partial derivative of other errors.

$$\nabla w_{01}^L = (y_1^L - T_1) * f'(s_1^L) * y_0^{L-1} = \delta_1^L * y_0^{L-1}$$

$$\nabla w_{11}^L = (y_1^L - T_1) * f'(s_1^L) * y_1^{L-1} = \delta_1^L * y_1^{L-1}$$

$$\nabla w_{21}^L = (y_1^L - T_1) * f'(s_1^L) * y_2^{L-1} = \delta_1^L * y_2^{L-1}$$

All the other gradients of the weights of this layer can be calcluated in this way. Thus generalizing,

$\delta_j^L = (y_j^L - T_j) * f'(s_j^L)$, and $\nabla w_{ij}^L = y_i^{L-1} * \delta_j^L$, where $0 < i < m_{L-1}$ (number of neurons in last hidden layer) and $1 < j < m_L$(number of neurons in last/output layer).

A.2.2 Base Step (last hidden layer)

Now, for the last hidden layer,

$$\nabla w_{01}^{L-1} = \frac{\partial E}{\partial w_{01}^{L-1}} = \frac{\partial E_1}{\partial w_{01}^{L-1}} + \frac{\partial E_2}{\partial w_{01}^{L-1}} + \frac{\partial E_3}{\partial w_{01}^{L-1}} + ... + \frac{\partial E_{m_L}}{\partial w_{01}^{L-1}}$$

$$\frac{\partial E_1}{\partial w_{01}^{L-1}} = \frac{\partial E_1}{\partial y_1^L}\frac{\partial y_1^L}{\partial s_1^L}\frac{\partial s_1^L}{y_1^{L-1}}\frac{\partial y_1^{L-1}}{\partial s_1^{L-1}}\frac{\partial s_1^{L-1}}{\partial w_{01}^{L-1}} = (y_1^L - T_1) * f'(s_1^L) * w_{11}^L * f'(s_1^{L-1}) * y_0^{L-2}$$

$$\frac{\partial E_1}{\partial w_{01}^{L-1}} = (\delta_1^L * w_{11}^L) * f'(s_1^{L-1}) * y_0^{L-2}$$

$$\frac{\partial E_2}{\partial w_{02}^{L-1}} = (\delta_2^L * w_{12}^L) * f'(s_1^{L-1}) * y_0^{L-2}$$

.

.

.

$$\frac{\partial E_{m_L}}{\partial w_{0m_L}^{L-1}} = (\delta_{m_L}^L * w_{1m_L}^L) * f'(s_1^{L-1}) * y_0^{L-2}$$

$$\nabla w_{01}^{L-1} = (\sum_{j=1}^{m_L} w_{1j}^L \delta_j^L) * f'(s_1^{L-1}) * y_0^{L-2} = \delta_1^{L-1} * y_0^{L-2}$$

All the other gradients for this layer can be calculated similarly. Thus, generalizing with i=1,2,3,...,m_{L-1}, and j=1,2,3,...,m_L

$$\delta_i^{L-1} = (\sum_{j=1}^{m_L} w_{ij}^L \delta_j^L) * f'(s_i^{L-1})$$

And, with i=0,1,2,...,m_{L-2}, and j=1,2,3,...,m_{L-1}.

$$\nabla w_{ij}^{L-1} = y_i^{L-2} * \delta_j^{L-1}$$

This can also be proved in a similar manner for the second last hidden layer as well.

A.2.3 Inductive Hypothesis ($(k+1)^{th}$ layer)

Inductive hypothesis states that

$$\delta_i^{k+1} = \nabla y_i^{k+1} * f'(s_i^{k+1}) = \frac{\partial E}{\partial y_i^{k+1}} * f'(s_i^{k+1}) = (\sum_{j=1}^{m_{k+2}} w_{ij}^{k+2} \delta_j^{k+2}) * f'(s_i^{k+1})$$

$$\nabla w_{ij}^{k+1} = y_i^k * \delta_j^{k+1}; i = 0, 1, 2, ..., m_k; j = 1, 2, 3, ..., m_{k+1}$$

A.2.4 Inductive Step ($(k)^{th}$ layer)

Each neuron in a layer receives the back-propagated error from all the neurons of the next layer. The error propagated by the following layer already includes all the possible paths via which the error from the output layer could travel; each error signal is built out of the error signals from the consecutive layer. Or in other words, during backpropagation, the gradients of a path is the same. It does not matter the path is used by which neurons to receive the error from the output layer.

Mathematically speaking,

$$\nabla w_{01}^k = \frac{\partial E}{\partial w_{01}^k} = \frac{\partial E}{\partial y_1^{k+1}} \frac{\partial y_1^{k+1}}{\partial s_1^{k+1}} \frac{\partial s_1^{k+1}}{\partial y_1^k} \frac{\partial y_1^k}{\partial s_1^k} \frac{\partial s_1^k}{\partial w_{01}^k} + \frac{\partial E}{\partial y_2^{k+1}} \frac{\partial y_2^{k+1}}{\partial s_2^{k+1}} \frac{\partial s_2^{k+1}}{\partial y_1^k} \frac{\partial y_1^k}{\partial s_1^k} \frac{\partial s_1^k}{\partial w_{01}^k} + ...$$

$$+ \frac{\partial E}{\partial y_{m_{k+1}}^{k+1}} \frac{\partial y_{m_{k+1}}^{k+1}}{\partial s_{m_{k+1}}^{k+1}} \frac{\partial s_{m_{k+1}}^{k+1}}{\partial y_1^k} \frac{\partial y_1^k}{\partial s_1^k} \frac{\partial s_1^k}{\partial w_{01}^k}$$

$$\nabla w_{01}^{k} = (\delta_1^{k+1} * w_{11}^{k+1} + \delta_2^{k+1} * w_{12}^{k+1} + ... + \delta_{m_{k+1}}^{k+1} * w_{1m_{k+1}}^{k+1}) * f'(s_1^k) * y_0^{k-1}$$

$$\nabla w_{01}^{k} = \delta_1^k * y_0^{k-1}$$

This means with i=1,2,3,...,m_k, and j=1,2,3,...,m_{k+1}

$$\delta_i^k = (\sum_{j=1}^{m_{k+1}} w_{ij}^{k+1} \delta_j^{k+1}) * f'(s_i^k)$$

And, with i=0,1,2,...,m_{k-1}, and j=1,2,3,...,m_k.

$$\nabla w_{ij}^{k} = y_i^{k-1} * \delta_j^k$$

A.2.5 Summary

Following the base step, and the inductive hypothesis; the necessary and sufficient conditions for proof by induction is met. Backpropagation algorithm actually is inspired by retrograde analysis or backward induction. The error signals travel backward in exactly the same way as the pre-activated value travel forward during the time of the forward propagation. The faster algorithm of backpropagation that used the dynamic algorithm for speed, is a little bit unclear as regard to how actually the backpropagation algorithm works. But that is fine, as it itself is an example of representation learning: moreover, it is faster to implement by the computers.

Appendix B

Representation Learning

B.1 Neural Network as a 1-vs-all Model

In an ANN, at the moment the network is trained, the network could be split into two parts; the first part contains all the hidden layers, while the second part contains the output layer. The first part is the feature extractor. And, the second part is the classifier. Whenever the system gets a new instance to categorize, it pre-processes the image first using the first network and extracts the best features from the data that it could learn during the training time. Now, the output of the feature extractor network is used by the second network (classifier) to predict the class of the instance.

In the case of K-class classification, we will have K neurons in the output layer, that is, $m_L = K$. During the training time, the network thus requires 10 truths to correct the network when it sees each training sample. This is achieved by a process called one hot vectorization. Let us say the current data instance belongs to class K. Now, the truth vector is achieved simply by $[0, 0, 0, 0, ..., 0, 0, 0, ..., 0, 0, 0, 1]$, that is, the truth for the output layer belonging to that particular class is set 1, while the truth for all other classes is set to 0. This is called hard classification. Even though the neural network, by default, does soft classification, with the use of this technique we change the training labels into hard classification. Similarly, during the test period, after the training is completed, the output of the network might be $[r_1, r_2, r_3, ..., r_K]$. r_i is some real number value. Condition like this is called soft classification. One hot vectorization changes the values of r_i to either 0 or 1; if r_i is the maximum, that is, $max([r_1, r_2, r_3, ..., r_K])$, then it is set to 1. Else, to 0.

The K neurons of the output layer are independent nodes, that is, they do not depend upon each other. Once, we separate the network into two parts as mentioned earlier. The second part is just K independent neurons. The output from the first network (feature extractor) is passed as input to all these K neurons, but independently. These independent K neurons are thus K independent logit units, or, logistic regression units. If we use softmax function for the output layer during training, what softmax function does is changes the output of the network into probabilistic one, that is, $r_1 + r_2 + r_3 + ... + r_K = 1$. It is a soft classification as it gives the probability of the data instance belonging to all the classes, the total sum is 1. One hot vectorization during the time of the test, changes this into hard classification using the highest class probability by giving it as 1 and all the others zero value.

Now, again during the testing time, this does not change the idea that the second part of the two-part split network is K-bunch of independent units. Without applying the softmax function we can decide the class of the current test instance as the logit unit among K-independent units with a current highest class value. If we need the probability of this particular test instance belonging to each class, one way to find it is by using softmax. Hence, the second part of the network, or, the classifier network is just a 1-vs-all logistic regression classifiers. Logistic regression heavily depends on the representation of the data, but if provided with the one, is a very good classification model.

This helps in the consideration of neural networks, especially the deep ones, as great examples of representation learning.

B.2 Neural Network as Image Generator

ANNs, when used for image generation, is alike to ANNs used for classification. The difference lies in the point that in the case of image generation, the output values are pixel values lying between $0 - 255$. This is the case of pointwise regression. Splitting the network into two parts as before yields as the first network as the feature extractor again, while the second network as K-independent linear/logistic regression units; linear if no activation or linear activation is used on the final layer. Linear regression is merely logistic regression with linear activation used.

Hence, this strengthens the concept of deep neural networks being a representation learning models.

Bibliography

[1] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013.

[2] Zhang, Wei (1988). "Shift-invariant pattern recognition neural network and its optical architecture". Proceedings of annual conference of the Japan Society of Applied Physics.

[3] Zhang, Wei (1990). "Parallel distributed processing model with local space-invariant interconnections and its optical architecture". Applied Optics. 29 (32): 47907.

[4] Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network", 2003.

[5] T. Serre, A. Oliva, and T. Poggio, A feedforward architecture accounts for rapid categorization, Proc. Nat. Acad. Sci. USA, vol. 104, no. 15, pp. 64246429, 2007.

[6] N. Pinto, N. Majaj, Y. Barhomi, E. Solomon, D. Cox, and J. DiCarlo, Human versus machine: Comparing visual object recognition systems on a level playing field, in Proc. Comput. Syst. Neurosci., Mar. 2010, [Online]. Available: http:// www.frontiersin.org/10.3389/conf.fnins.2010.03.00283/event_abstract, doi: 10.3389/conf.fnins.2010.03.00283.

[7] J. J. DiCarlo, D. Zoccolan, and N. C. Rust, How does the brain solve visual object recognition? Neuron, vol. 73, no. 3, pp. 415434, 2012.

[8] Goodale MA, Milner AD. "Separate visual pathways for perception and action". Trends Neurosci, 1992.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 10971105.

[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio, Maxout networks, in Proc. 30th Int. Conf. Mach. Learn., 2013, pp. 13191327.

[11] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet clas- sification, in Proc. IEEE Int. Conf. Comput. Vis., Dec. 2015, pp. 10261034.

[12] M. Zeiler and R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, in Proc. ICLR, 2013. [Online]. Available: https://arxiv.org/abs/1301.3557

[13] M. Malinowski and M. Fritz. (2013). Learnable pooling regions for image classification. [Online]. Available: https://arxiv. org/abs/1301.3516

[14] K. He, X. Zhang, S. Ren, and J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 9, pp. 19041916, Sep. 2015.

[15] Y. Gong, L. Wang, R. Guo, and S. Lazebnik, Multi-scale orderless pooling of deep convolutional activation features, in Proc. Eur. Conf. Comput. Vis., 2014, pp. 392407.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res., vol. 15, no. 1, pp. 19291958, 2014.

[17] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, Regularization of neural networks using dropconnect, in Proc. 30th Int. Conf. Mach. Learn., 2013, pp. 10581066.

[18] A. Krogh and J. A. Hertz, A simple weight decay can improve gener- alization, in Proc. Adv. Neural Inf. Process. Syst., vol. 4. San Mateo, CA, USA, 1995, pp. 950957.

[19] G. E. Dahl, D. Yu, L. Deng, and A. Acero, Context-dependent pre- trained deep neural networks for large-vocabulary speech recognition, IEEE Trans. Audio, Speech, Lang. Process., vol. 20, no. 1, pp. 3042, Jan. 2012.

[20] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in Proc. Int. Conf. Mach. Learn., 2015, pp. 448456.

[21] K. Simonyan and A. Zisserman. (2014). Very deep convolutional networks for large-scale image recognition. [Online]. Available: https://arxiv.org/abs/1409.1556

[22] C. Szegedy et al., Going deeper with convolutions, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2015, pp. 19.

[23] K. He, X. Zhang, S. Ren, and J. Sun. (2015). Deep residual learning for image recognition. [Online]. Available: https://arxiv.org/abs/1512.03385

[24] M. Lin, Q. Chen, and S. Yan, Network in network, in Proc. ICLR, 2014. [Online]. Available: https://arxiv.org/abs/1312.4400

[25] Weiwei Shi, Yihong Gong, Xiaoyu Tao, Jinjun Wang, and Nanning Zheng, Fellow, "Improving CNN Performance Accuracies With MinMax Objective", IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 29, NO. 7, JULY 2018.

[26] Tsung-Yu Lin,Aruni RoyChowdhury, and Subhransu Maji, "Bilinear CNN Models for Fine-grained Visual Recognition"

[27] Ian Goodfellow and Yoshua Bengio and Aaron Courville, "Deep Learning", MIT Press, 2016, pp. 1-26

[28] Mor-Yosef S, Samueloff A, Modan B, Navot D, Schenker JG, "Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study", in Proc. National Center for Biotechnology Information, 1990 Jun

[29] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115133. doi:10.1007/BF02478259.

[30] Hebb, Donald (1949). The Organization of Behavior. New York: Wiley. ISBN 978-1-135-63190-1.

[31] Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". Psychological Review. 65 (6): 386408.

[32] Rumelhart, D.E; McClelland, James (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge: MIT Press. ISBN 978-0-262-63110-5.

[33] Hinton, G. E.; Osindero, S.; Teh, Y. (2006). "A fast learning algorithm for deep belief nets" (PDF). Neural Computation. 18 (7): 15271554. CiteSeerX 10.1.1.76.1541. doi:10.1162/neco.2006.18.7.1527. PMID 16764513.

[34] Larochelle, Hugo; Erhan, Dumitru; Courville, Aaron; Bergstra, James; Bengio, Yoshua (2007). An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. Proceedings of the 24th International Conference on Machine Learning. ICML '07. New York, NY, USA: ACM. pp. 473480. CiteSeerX 10.1.1.77.3242

[35] Ranzato, Marc Aurelio; Boureau, Y-Lan (2007). "Sparse Feature Learning for Deep Belief Networks" (PDF). Advances in Neural Information Processing Systems. 23: 18.

[36] Widrow, B., and Hoff, M. E. (1960). Adaptive switching circuits. In IRE WESCON Convention Record.

[37] Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". Biological Cybernetics. 36 (4): 93202. doi:10.1007/BF00344251. PMID 7370364.

[38] LeCun, Yann; Lon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 22782324. CiteSeerX 10.1.1.32.9552. doi:10.1109/5.726791. Retrieved October 7, 2016.

[39] Fukushima, K.: Cognitron: a self-organizing multilayered neural network. Biol. Cybernetics 20, 121-136 (1975)

[40] Fukushima, K. : Improvement in pattern-selectivity of a cognitron (in Japanese). Pap. Tech. Group MBE78-27, IECE Japan (1978)

[41] Vinod Nair, and Geoffrey E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines", 2010.

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, "Deep Sparse Rectifier Neural Networks", 2011.

[43] K Jarrett, K Kavukcuoglu, Y LeCun 2009 IEEE 12th International Conference on Computer Vision (ICCV), 2146-2153.

[44] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533536. Bibcode:1986Natur.323..533R. doi:10.1038/323533a0.

[45] Pearson, K. (1901). ”On Lines and Planes of Closest Fit to Systems of Points in Space”. Philosophical Magazine. 2 (11): 559572. doi:10.1080/14786440109462720.

[46] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology, 24, 417441, and 498520.

[47] Hotelling, H (1936). ”Relations between two sets of variates”. Biometrika. 28 (3/4): 321377. doi:10.2307/2333955. JSTOR 2333955.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Bikram Basnet
basnetbik@gmail.com

Degrees:

Bachelor in Computer Engineering (B.E.) 2014
Institute of Engineering Central Campus, Tribhuvan University, Nepal

Thesis Title: A Novel Feature Maps Covariance Minimization Approach for Advancing Convolutional Neural Network Performance

Thesis Examination Committee:

Chairperson, Dr. Justin Zhan, Ph.D.
Committee Member, Dr. Wolfgang Bein, Ph.D.
Committee Member, Dr. Laxmi Gewali, Ph.D.
Graduate Faculty Representative, Dr. Ge Lin Kan, Ph.D.