

University of Louisville

ThinkIR: The University of Louisville's Institutional Repository


Electronic Theses and Dissertations

5-2019

Receptive fields optimization in deep learning for enhanced interpretability, diversity, and resource efficiency.

Babajide Odunitan Ayinde
University of Louisville

Follow this and additional works at: <https://ir.library.louisville.edu/etd>

 Part of the [Computational Engineering Commons](#), [Other Electrical and Computer Engineering Commons](#), and the [Signal Processing Commons](#)

Recommended Citation

Ayinde, Babajide Odunitan, "Receptive fields optimization in deep learning for enhanced interpretability, diversity, and resource efficiency." (2019). *Electronic Theses and Dissertations*. Paper 3243.
<https://doi.org/10.18297/etd/3243>

This Doctoral Dissertation is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

RECEPTIVE FIELDS OPTIMIZATION IN DEEP LEARNING FOR ENHANCED
INTERPRETABILITY, DIVERSITY, AND RESOURCE EFFICIENCY

By

Babajide Odunitan Ayinde
B.S., Obafemi Awolowo University, 2011
M.S., King Fahd University of Petroleum and Minerals, 2015

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy in Electrical Engineering

Electrical and Computer Engineering
University of Louisville
Louisville, Kentucky

May 2019

RECEPTIVE FIELDS OPTIMIZATION IN DEEP LEARNING FOR ENHANCED
INTERPRETABILITY, DIVERSITY, AND RESOURCE EFFICIENCY

Submitted by

Babajide Odunitan Ayinde

A Dissertation Approved on

January 23, 2019

by the Following Dissertation Committee:

Dr Jacek Zurada, Dissertation Director

Dr Tamer Inanc

Dr Eric Rouchka

Dr Huacheng Zeng

DEDICATION

This is dedicated to God Almighty for availing the strength, inspiration, knowledge, and understanding required to successfully complete this dissertation. I also dedicate this work to my father Dr Folorunsho Ayinde and my late mother Felicia Mojisola Ayinde. To my beloved wife Ajoke Ayinde and son David Ayinde who have been affected in every way possible by this rigorous quest.

ACKNOWLEDGMENTS

I would like to express my profound gratitude to my advisor, Dr. Jacek Zurada, Director of the Computational Intelligence Laboratory at the University of Louisville, for his valuable contributions, guidance, and support towards the completion of this work. I would also like to appreciate the School of Interdisciplinary and Graduate Studies, University of Louisville for awarding me the Dissertation Completion Fellowship to enable me put this work together. My appreciation also goes to Dr Tamer Inanc for the knowledge impacted through the Digital Signal Processing class and also for being one of my dissertation committee members. I would also like to thank Dr. Eric Rouchka and Dr. Huacheng Zeng for agreeing to serve on the dissertation committee. I am grateful for their suggestions, encouragements, and advice.

There are a number of people without whom this dissertation might not have been written, and to whom I am greatly indebted. I thank my wife and son, Ajoke and David, for their patience, sacrifice, and constant support while this dissertation slowly assumes its current form. I am indebted to my dad Dr. Folorunsho Ayinde, my parents-in-law Mr and Mrs Chiazor Ndidi, and Mr and Mrs Bamidele Oresegun for their love, support, and words of encouragement. Thanks to all my siblings, Dr Olusola, Engr. Olugbenga, Mrs Olukemi Oyem, Mrs Omobola Favour, Engr. Ayokunle, Mrs Omolola Akinrinde, Engr. Tolulope for all their support and prayers. I am grateful to all my friends and colleagues at University of Louisville for the wonderful times we shared. A big thank you to everybody who contributed in one way or the other to the success of this dissertation. Ultimately, I thank Almighty God for every success in this process.

ABSTRACT

RECEPTIVE FIELDS OPTIMIZATION IN DEEP LEARNING FOR ENHANCED INTERPRETABILITY, DIVERSITY, AND RESOURCE EFFICIENCY

Babajide Odunitan Ayinde

January 23, 2019

In both supervised and unsupervised learning settings, deep neural networks (DNNs) are known to perform hierarchical and discriminative representation of data. They are capable of automatically extracting excellent hierarchy of features from raw data without the need for manual feature engineering. Over the past few years, the general trend has been that DNNs have grown deeper and larger, amounting to huge number of final parameters and highly nonlinear cascade of features, thus improving the flexibility and accuracy of resulting models. In order to account for the scale, diversity and the difficulty of data DNNs learn from, the architectural complexity and the excessive number of weights are often deliberately built in into their design. This flexibility and performance usually come with high computational and memory demands both during training and inference. In addition, insight into the mappings DNN models perform and human ability to understand them still remain very limited. This dissertation addresses some of these limitations by balancing three conflicting objectives: computational/memory demands, interpretability, and accuracy.

This dissertation first introduces some unsupervised feature learning methods in a broader context of dictionary learning. It also sets the tone for deep

autoencoder learning and constraints for data representations in light of removing some of the aforementioned bottlenecks such as the feature interpretability of deep learning models with nonnegativity constraints on receptive fields. In addition, the two main classes of solution to the drawbacks associated with over-parameterization/over-complete representation in deep learning models are also presented. Subsequently, two novel methods, one for each solution class, are presented to address the problems resulting from over-complete representation exhibited by most deep learning models. The first method is developed to achieve inference-cost-efficient models via elimination of redundant features with negligible deterioration of prediction accuracy. This is important especially for deploying deep learning models into resource-limited portable devices. The second method aims at diversifying the features of DNNs in the learning phase to improve their performance without undermining their size and capacity. Lastly, feature diversification is considered to stabilize adversarial learning and extensive experimental outcomes show that these methods have the potential of advancing the current state-of-the-art on different learning tasks and benchmark datasets.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGMENTS	iv
ABSTRACT	v
List of Tables	x
List of Figures	xii
CHAPTER	
INTRODUCTION	1
I. CONSTRAINED FEATURE LEARNING	4
A. Dictionary Learning via Sparse Coding	4
B. Dictionary Learning via Nonnegative Matrix Factorization (NMF)	6
C. Dictionary Learning via Constrained Autoencoders	7
1. Constrained Autoencoders for Sparse Representation	9
2. Constrained Autoencoders for Part-based Data Representation	11
II. CONSTRAINED AUTOENCODERS FOR ENHANCED DATA UNDERSTANDING	16
A. L_1/L_2 -Nonnegativity Constrained Sparse Autoencoder (L_1/L_2 -NCSAE)	17
1. Implication of imposing nonnegative parameters with composite decay function	19
2. Experimental Results	20
a. Unsupervised Feature Learning of Image Data	20

b.	Unsupervised Semantic Feature Learning from Textual Data	29
c.	Illustration of Understandable Feature Extraction by L_1/L_2 -NCSAE	30
B.	Deep Learning of Understandable Features using Cascaded L_1/L_2 -NCSAE	33
1.	Image Classification with Enhanced Interpretability	35
2.	Document Categorization with Enhanced Interpretability	39
3.	Performance Evaluation on Supervised Learning	43
C.	Conclusion	45
III.	UNSUPERVISED NONREDUNDANT FEATURE EXTRACTION	46
A.	Filtering Redundancy Elimination in Autoencoder-based Deep Networks	47
1.	Filter Clustering and Reduction	48
a.	Static Reduction of the Number of Redundant Filters	48
b.	Dynamic Reduction and Reconciliation of Filters	50
B.	Experimental Setup	52
a.	Unsupervised Feature Reduction via Filter Pruning	61
b.	Effect of Redundant Feature Pruning on Supervised Learning	64
C.	Conclusion	72
IV.	REDUNDANCY-BASED FILTER PRUNING IN DEEP CONVOLUTIONAL NEURAL NETWORKS	74
A.	Convolutional Feature Clustering and Pruning	78
1.	Method A: Pruning of Redundant Filters	80
2.	Method B: Pruning of Random n_f Filters	82
B.	Experiments	83

1. VGG-16 on CIFAR-10	87
2. RESNET-56/110 on CIFAR-10	91
3. Prune and Train from Scratch	101
C. Conclusion	102
V. FEATURE DIVERSIFICATION IN DEEP NEURAL NETWORKS	103
A. Enhancing Feature Diversity by enforcing Dissimilar Feature Ex- traction	106
1. Diversity Regularization	108
2. Implications of imposing feature diversity	110
B. Online Redundant Filter Detection and Dropout	111
1. Online Filtering Redundancy Dropout	112
2. Online Redundancy-based Dropout	113
C. Experiments	115
1. Feature Evolution during Training	117
2. Diversity Regularized Image Classification	122
3. Diversity Regularized Natural Language Inference	126
D. Diversity Regularized Adversarial Learning (DiReAL)	131
E. Conclusion	133
VI. CONCLUSIONS	136
REFERENCES	139
CURRICULUM VITAE	159

LIST OF TABLES

1	Classification accuracy on MNIST and NORB dataset [1]	44
2	Parameter settings [2]	54
3	Classification performance on MNIST dataset using initial network configuration 784-200-20-10 [2].	65
4	Classification performance on MNIST dataset using initial network configuration 784-1000-20-10 [2].	67
5	Classification performance on MNIST dataset using initial network configuration 784-1000-200-10 [2].	69
6	Classification performance on MNIST dataset using initial network configuration 784-500-500-10 [2].	70
7	Classification performance on MNIST dataset using initial network configuration 784-1000-1000-10 [2].	71
8	Classification performance on NORB dataset [2].	72
9	Pruning performance on CIFAR dataset using VGG-16 model at $\tau = 0.54$ [3].	92
10	Performance evaluation for three pruning techniques on CIFAR-10 dataset. Performance with the lowest test error is reported [3].	93
11	Performance evaluation of three pruning techniques for ResNet 56/110 trained on CIFAR-10 dataset. Performance with the lowest test error is reported [3].	95
12	FLOP and CPU time reduction for inference. Operations in convolutional and fully connected layer are considered for computing FLOP [3].	101

13	Performance on CIFAR dataset [3].	102
14	Test-train error gap on MNIST [4]	120
15	Test error(%) on MNIST. Source: [4]	122
16	Test error(%) on CIFAR-10 without data augmentation. Source: [4]	125
17	Validation error on ImageNet. Source: [4]	126
18	A select examples from SNLI dataset where E, C, and N repre- sent Entailment, Contradiction, and Neutral, respectively. Source: [4]	128
19	Test accuracy (%) on SNLI dataset. Source: [4]	130

LIST OF FIGURES

1	Illustration of Data Matrix Factorization ($\mathbf{X} \approx \Phi\mathbf{A}$). \mathbf{X} is the data matrix, columns of Φ are basis vectors, and columns of \mathbf{A} are the encodings of the samples [5].	5
2	Schematic diagram of a three-layer AE	8
3	RFs or weights of randomly selected 32 out of 196 ($n' = 196$) hidden neurons of (a) NNSAE (b) NCAE trained using MNIST dataset. Black pixels indicate negative, grey pixels indicate zero-valued weights and white pixels indicate positive weights. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color [5].	12
4	Representation of test image as a linear combination of 4 out of 196 constrained RFs and decoding filters learned from MNIST dataset using NCAE with linear output activation function. Input consist of 784 values corresponding to a 28×28 pixel image. Only 70 RFs with largest activations to test image "6" and their corresponding decoding filters are shown. The RFs and the decoding filters are rescaled and portrayed as images on the right hand side. Black pixels indicate negative, and white pixels indicate positive weights. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. The biases are not shown [5].	15

5	Absolute function approximation using quadratic smoothing functions with $\kappa = 0.4, 0.7, 1.0$ and 1.5	20
6	(a) Symmetric (G_3) and skewed (G_1 and G_2) weight distributions. Decay function with three values of α_1 and α_2 for weight distribution (b) G_3 (c) G_1 and (d) G_2 . [1]	21
7	196 receptive fields ($\mathbf{W}^{(1)}$) learned from MNIST digit data set using (a) SAE, (b) DpAE (c) NCAE, and (d) L_1/L_2 -NCSAE. Black pixels indicate negative, and white pixels indicate positive weights. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color [1].	23
8	Encoding weights ($\mathbf{W}^{(1)}$) histograms learned from MNIST digit data set using (a) SAE, (b) DpAE (c) NCAE, and (d) L_1/L_2 -NCSAE [1].	24
9	(a) Reconstruction error and (b) Sparsity of hidden units measured by KL-divergence using MNIST train dataset with $p = 0.05$ [1].	24
10	t-SNE projection [6] of 196D representations of MNIST handwritten digits using (a) SAE (b) DpAE (c) NCAE (d) L_1/L_2 -NCSAE [1].	25
11	Weights of randomly selected 90 out of 200 receptive filters of (a) SAE (b) DpAE (c) NCAE, and (d) L_1/L_2 -NCSAE using NORB dataset. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w \leq -1$ is assigned to black, $w = 0$ to grey, and $w \geq 1$ is assigned to white color [1].	26

12	The distribution of 200 encoding ($\mathbf{W}^{(1)}$) and decoding filters ($\mathbf{W}^{(2)}$) weights learned from NORB dataset using (a) DpAE (b) NCAE (c) L_1/L_2 -NCSAE [1].	27
13	Visualizing 20D representations of a subset of Reuters Documents data using (a) DpAE, (b) NCAE, and (c) L_1/L_2 -NCSAE [1].	32
14	Illustration of constrained (non-negative) RF feature extraction using a L_1/L_2 -NCSAE trained on synthetic data with 3 images (left). The RFs learned (right) are rescaled and portrayed as images. The range of weights are scaled to $[-1,1]$ and mapped to the grayscale map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. That is, black pixels indicate negative, and white pixels indicate positive weights. The dot product of each RF and input pattern shown as Activation Scores and the outputs of Softmax layer as Softmax scores. a,b, and c are the indices of input images; d,e, and f are the RFs indices. The biases are not shown [5].	34
15	Schematic diagram of a deep AE of $L + 1$ layers constructed using Stacked Sparse Autoencoder (SSAE) and Softmax Classifier (SMC).	35

16 Filtering the signal through the L_1/L_2 -NCSAE trained using the reduced MNIST data set with class labels 1, 2 and 6. The test image is a 28×28 pixels image unrolled into a vector of 784 values. Both the input test sample and the receptive fields of the first autoencoding layer are presented as images. The weights of the output layer are plotted as a diagram with one row for each output neuron and one column for every hidden neuron in $(L - 1)^{th}$ layer. The architecture is 784-10-10-3. The range of weights are scaled to $[-1,1]$ and mapped to the gray-color map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. That is, black pixels indicate negative, grey pixels indicate zero-valued weights and white pixels indicate positive weights [1]. 38

17 The weights were trained using two stacked L_1/L_2 -NCSAEs. RFs learned from the reduced NORB dataset are plotted as images at the bottom part of (a). The intensity of each pixel is proportional to the magnitude of the weight connected to that pixel in the input image with negative value indicating black, positive values white, and the value 0 corresponding to gray. The biases are not shown. The activations of first layer hidden units for the NORB objects presented in (b) are depicted on the bar chart on top of the RFs. The weights of the second layer AE are plotted as a diagram at the topmost part of (a). Each row of the plot corresponds to the weight of each hidden unit of second AE and each column for weight of every hidden unit of the first layer AE. The magnitude of the weight corresponds to the area of each square; white indicates positive, grey indicates zero, and black negative sign. The activations of second layer hidden units are shown as bar chart in the right-hand side of the second layer weight diagram. Each column shows the activations of each hidden unit for five color-coded examples of the same object. The outputs of Softmax layer for color-coded test objects with class labels (c) "fourlegged animals" tagged as class 1, (d) "human figures" as class 2, and (e) "airplanes" as class 3 [1]. 40

18 Deep network trained on Reuters-21578 data using (a) DpAE, (b) L_1/L_2 -NCSAE. The area of each square is proportional to the weight's magnitude. The range of weights are scaled to [-1,1] and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color [1]. . . 42

19	Filters learned from NORB data set using SAE with (a) 100 original filters (b) 14 examples of very similar filters with their corresponding indices at the bottom (c) 32 filters resulting from aggro-SAE with $\tau=16$, and (d) 32 original filters. Black pixels indicate negative, and white pixels indicate positive weights [2].	51
20	Performance of AE on the NORB dataset. (a) Reconstruction error vs. cluster similarity threshold. (b) Number of RFs vs. cluster similarity threshold for 196 initial filters [2].	55
21	Reconstruction error using SAE, NCAE, and L_1/L_2 -NCAE trained with the same number of hidden units from experiment with aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE on NORB data [2].	55
22	100 receptive fields learned from Yale Face Dataset using SAE with examples of two duplicative RFs [2].	56
23	100 receptive fields learned from Yale Face Dataset using NCAE with examples of duplicative RFs [2].	56
24	100 receptive fields learned using L_1/L_2 -NCAE from Yale Face Dataset, with examples of duplicative RFs [2].	57
25	Filters learned from MNIST data set using SAE (a) 200 RFs with examples of duplicative filters, (b) 125 RFs for aggro-SAE with $\tau=0.6$ [2]	57
26	Performance of AE on the MNIST dataset vs. cluster similarity threshold (a) Reconstruction error (b) RF size (n'_{new}) using 200 initial filters [2].	58

27	Reconstruction error using SAE, NCAE, and L_1/L_2 -NCAE trained with the same number of hidden units from experiment with aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE on MNIST data [2].	58
28	KL-Divergence sparsity measure with respect to a desired $p = 0.05$ using SAE, NCAE, and L_1/L_2 -NCAE trained with the same number of hidden units (n'_{new}) from experiment with aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE on MNIST data [2].	59
29	t-SNE projection [6] of 200D representations of MNIST handwritten digits using (a) SAE (b) NCAE (c) L_1/L_2 -NCAE, (d) 174D representations using aggro-SAE (e) 153D representations using aggro-NCAE, and (f) 172D representations using aggro- L_1/L_2 -NCAE [2].	60
30	Pruning schema of l^{th} layer. (a) Assume filters red, blue, and green in the first, third, and fifth columns of $\mathbf{W}^{(l)}$, respectively, are very similar and are located in the same cluster (b) If filter red is sampled as the representative of the cluster, filters blue and green are redundant and their corresponding feature maps in Z_{l+1} and related weights in the next layer (third and fifth rows of $\mathbf{W}^{(l+1)}$) are all pruned [3].	76
31	Average number of redundant features across all layers (\bar{n}_r) against threshold τ with (a) one (b) two (c) three, and (d) four hidden layers using MNIST dataset. Network width corresponds to the number of hidden units per layer and network depth corresponds to number of hidden layers. Networks with more than one hidden layer have equal number of hidden units in all layers.	85

32	t-SNE projection [7] of the activation of last layer of network with (a) one and (b) four hidden layers using 5000 MNIST handwritten digits test samples. All Networks have 1000 hidden units in all layers and all layers use Sigmoid activation function.	86
33	Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for VGG-16 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend [3].	88
34	Sensitivity to pruning (a) redundant filters (b) random $n' - n_f$ filters, and (c) redundant filters and retraining for 30 epochs for VGG-16 [3].	90
35	Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for ResNet-56 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend [3].	94
36	Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for ResNet-110 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend [3].	96
37	Sensitivity to pruning $n' - n_f$ redundant convolutional filters in ResNet-56 [3].	98
38	Sensitivity to pruning $n' - n_f$ redundant convolutional filters in ResNet-110 [3].	100
39	Illustration of effect of divReg with $\lambda = 10$ and $\tau = 0.1$ (a) on three toy filters in (b) iteration 1 (c) iteration 2 and (d) iteration 4 [4].	107
40	Effect of (a) diversity penalty factor λ and (b) thresholding parameter τ on diversity regularization cost J_D (Figure best viewed in color) [4]	108

41	The distribution of pairwise feature correlation ($\Omega(1)$) in first hidden layer at (a) epoch 2 (b) epoch 300 [4]	118
42	The distribution of pairwise feature correlation ($\Omega(2)$) in second hidden layer at (a) epoch 2 (b) epoch 300 [4]	119
43	150 out of 256 encoding features (left) learned from MNIST digit data set with autoencoders using (a) L_1 , (b) Dropout (c) orthoReg, and (d) divReg. The range of weights are scaled and mapped to the graycolor map (right) [4].	120
44	Performance of Multilayer Perceptron (with architecture 784-1024-1024-10) regularized using divReg-1 and trained on the MNIST dataset vs. threshold τ^* . (a) Number of nonredundant features for 1024 initial features. (b) percentage classification error [4]	121
45	Evolution of dropout fraction (α) with divReg-2 using the MNIST dataset for four different initializations of α in (a) layer 1 and (b) layer 2. Source: [4]	121
46	Performance evaluation using divReg-2 on MNIST dataset for four different initializations of α . Source: [4]	127
47	Learning rate (ζ) schedule for experiments on CIFAR-10 dataset. Source: [4]	127
48	Schema of Diversity Regularized Adversarial Learning (DiReAL)	132
49	Diversity loss of (a) generator with no regularization (b) generator with diReAL (c) discriminator with no regularization, and (d) discriminator with DiReAL trained on MNIST dataset.	133
50	Divergence, as measured by Wasserstein distance, between the discriminator output for real and synthesized samples	134

51	Synthesized hand-written digits with and without diversity Regularization.	134
----	--	-----

INTRODUCTION

Many real-world learning problems involve high-dimensional data and the curse of dimensionality is a fundamental issue. Analysis of data with high dimensions usually results in significant increase in computational time and space [8,9]. From practical standpoint, the importance of all features is not the same for a given discriminative task and a good number of features are highly correlated or even redundant. This redundancy, in general, would not only increase the computational complexity of the learning process, but also would hinder the interpretability and transparency of the resulting model. However, manual engineering and selection of the most important feature set in high dimensional data for the purpose of eliminating redundancy is extremely difficult and labor-intensive.

Noticeable research efforts have addressed this issue by designing preprocessing and feature extraction pipelines to condition original raw data into forms effectively usable by learning algorithms. However, the process is tedious and requires considerable efforts by experts. In fact, some of the best unsupervised feature extractors (Restricted Boltzmann machines (RBMs) [10], Deep Belief Networks (DBNs) [11], autoencoders (AE) [12], stacked AE [13], Sparse coding [14], Gaussian Mixture Models (GMMs) [15]) and supervised counterparts (Support Vector Machines (SVM) [16], Gradient Boosting machines (GBMs) [17], neural networks [18, 19], etc) produce outputs that are unintelligible and inherently hard to decipher their decision making processes. These issues are in fact more prominent when multilayer deep learning (DL) architectures are used. The notion of "deep" in DL does not refer to any kind of deeper understanding/knowledge, rather it refers to the learning of many layers or hierarchies of feature representations. Therefore,

most of the existing DL models can only be used as black-boxes despite their good performance because their knowledge is hidden and can hardly be used to explain their decision making process. Thus limiting their applicability in domains where both justifications of decisions and interpretable inference are required from machines as in medical applications and business intelligence [20].

Most of the problems associated with interpretability and computational efficiency especially in DL models have been attributed to huge number of parameters, high nonlinearity, and redundancy in input and/or weight spaces [21], [22] [23], [24]. Therefore, designing a fully trainable algorithms that have the capability to learn the appropriate interpretable features and simultaneously eliminate redundancy in both input and model is a step towards solving a long-standing open problem of obtaining an optimal architecture that balances accuracy, memory demand, and interpretability.

The following five chapters of this dissertation cover important methods in constrained feature learning and data representation, describe methods of training interpretable features, and discuss algorithms to improve post-training inference cost of DNN models. Chapter II starts by describing some of the theoretical foundations of representation learning via constrained data matrix decomposition. It then describes some of the main methods for unsupervised feature extraction in artificial neural networks and the concept of receptive fields (RFs). It also explains how interpretable models can result from the extraction of additive part-based features. It ends with presenting of two approaches for achieving part-based data decomposition with neural networks and highlights some of their tradeoffs in terms of part-based data decomposition and accuracy.

Chapters III focuses on improving the interpretability of autoencoder-based DNN model while preserving the output accuracy. A novel method for imposing non-negativity constraints on RFs is introduced to learn interpretable and dis-

criminative features. It focuses on methods that preserves the accuracy of models with interpretable features. Chapters IV and V describe methods that seek to eliminate redundancy in both supervised and unsupervised neural network via RF compression. Chapter IV presents two methods for unsupervised learning of non-redundant sparse RFs to improve both computation and accuracy in the supervised phase. Chapter V presents two methods for improving the post-training computational efficiency of supervised deep convolutional neural networks, also via elimination of redundant RFs. A novel method for imposing diversity among RFs during training is presented and discussed in chapter VI to prevent redundancy.

CHAPTER I

CONSTRAINED FEATURE LEARNING

Constrained feature learning (CFL) is an important concept in feature engineering for unearthing latent representations of data useful for such machine learning tasks as classification, regression, and compression. These representations could reveal what is important in data for a given discriminative task. CFL algorithms that enable feature extraction can generate latent codes for test set during inference [25,26]. CFL algorithms have become important tools in paradigm of representation learning. These algorithms range from sparse coding concept originally introduced in [14] to Nonnegative Matrix Factorization (NMF) that enforces nonnegativity of both basis vectors and the features to neural networks that implement learning with a variety of constraints. They are able to learn constrained representation usually by learning some dictionaries that represents the data. The term dictionary is often used in the context of semantic analysis such as document categorization. When dealing with other tasks and data, these dictionaries are called receptive fields, filters, basis vectors or latent factors.

A. Dictionary Learning via Sparse Coding

Dictionary learning is best illustrated through sparse coding or data matrix factorization. Assume data matrix \mathbf{X} contains m data vectors \mathbf{x}_j as columns, each with n elements as shown in Fig. 1. Sparse coding aims to find a set of k basis vectors (columns ϕ_i of matrix $\Phi \in \mathbb{R}^{n \times k}$) and encodings (columns \mathbf{a}_j of matrix

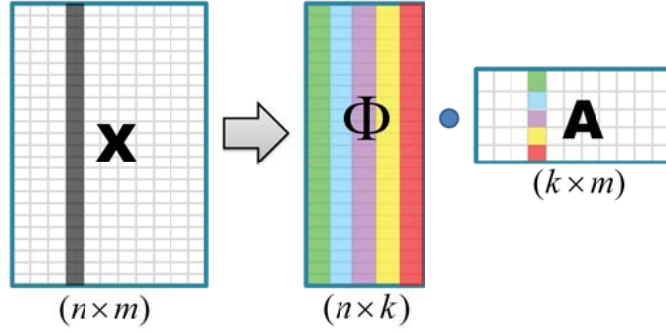


FIGURE 1 – Illustration of Data Matrix Factorization ($\mathbf{X} \approx \Phi\mathbf{A}$). \mathbf{X} is the data matrix, columns of Φ are basis vectors, and columns of \mathbf{A} are the encodings of the samples [5].

$\mathbf{A} \in \mathbb{R}^{k \times m}$) such that $\mathbf{X} \approx \Phi\mathbf{A}$ for $\mathbf{X} \in \mathbb{R}^{n \times m}$, and \mathbf{a}_j is a sparse vector for every j . When no limitation is imposed on k , it is possible to find via sparse coding an over-complete representation of data in which the number of basis vectors k is greater than the original data dimensionality n . That is, if $k > n$ the linear system of equations is under-determined and sparsity enforcement is needed to avoid obtaining a trivial solution [26].

In order to coerce \mathbf{a}_j to be sparse for every j , a sparsity term is introduced in the objective function. Sparse combination of basis from an over-complete dictionary to represent data has been suggested as the mechanism with which mammal primary visual cortex (V1) work [14,27–30]. The data matrix decomposition is usually formulated as an optimization problem solvable by balancing out the error of approximation of \mathbf{X} by $\Phi\mathbf{A}$ and the sparsity of \mathbf{A} . During the optimization process, a trivial solution may result in which entries of \mathbf{A} are small due to sparsity enforcement but are compensated by allowing entries of Φ to assume large values [27,31,32]. To alleviate this problem, magnitude constraints are usually placed on the basis vectors ϕ_i through a process known as regularization by adding decay term to the objective function. This magnitude constraint is sometimes referred to as a weight decay penalty. Most sparse coding methods [14, 27, 33] require solving iterative optimization problem in order to compute feature descriptor which is

usually computational expensive [26]. The complete optimization objective is thus formulated as in (1).

$$\min_{\mathbf{A}, \Phi} \sum_{j=1}^m \left[\|\Phi \mathbf{a}_j - \mathbf{x}_j\|_2^2 + \gamma_1 \text{Sparsity}(\mathbf{a}_j) \right] + \gamma_2 \sum_{i=1}^k \|\phi_i\|_2^2 \quad (1)$$

where γ_1 and γ_2 are positive constants that adjust the relative importance of sparsity and magnitude (or regularization) constraints, respectively. Formula (1) minimizes the distance between the data and its representation given the learned basis.

B. Dictionary Learning via Nonnegative Matrix Factorization (NMF)

Similar to sparse coding, NMF [34] belongs to a class of CFL paradigm that essential to data analysis such as compression, feature selection, visualization, just to mention a few [35]. NMF finds application in many diverse problem space such as computational biology [36–41], blind source separation [42], clustering [43, 44], community detection [45], collaborative filtering [46], just to mention a few. One of the motivation behind NMF is that the emergence of part-based representation in human cognition can be conceptually tied to the nonnegativity constraints [34]. The objective of NMF techniques in general is to approximate data matrix X with nonnegative entries with low rank matrix \mathbf{WH} , that is, $\mathbf{X} \approx \mathbf{WH}$ or simply $\mathbf{X} = \mathbf{WH} + N$. One of the key choices in NMF is the quantification of quality of approximation, which generally depends on error N . The most commonly used measure is the Frobenius norm of N , which assumes the noise in the data is Gaussian. Another important consideration in CFL is the assumption on the structure of factors \mathbf{W} and \mathbf{H} . For instance, if columns of \mathbf{W} are independent, then the resulting heuristic become independent component analysis (ICA) [47].

The strict constraint on the structure of factors W and H in NMF is that it enforces the encoding of both the basis vectors and features to be nonnegative

thereby resulting in additive data representation. The hidden structure of data can be unfolded by learning features that have capabilities to extract the data parts. Similar to the data decomposition illustrated in Fig. 1, NMF decomposes data matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ with nonnegative real entries into product of two nonnegative matrices $\mathbf{W} \in \mathbb{R}^{n \times k}$ and $\mathbf{H} \in \mathbb{R}^{k \times m}$, that is, $\mathbf{X} \approx \mathbf{WH}$. The factorization is generally formulated as an optimization problem with loss function in (2)

$$\min_{\mathbf{W} \in \mathbb{R}^{n \times k}, \mathbf{H} \in \mathbb{R}^{k \times m}} \|\mathbf{X} - \mathbf{WH}\|^2 \text{ such that } \mathbf{W} \geq 0 \text{ and } \mathbf{H} \geq 0 \quad (2)$$

C. Dictionary Learning via Constrained Autoencoders

Dictionaries are also learnt via a specialized neural network architecture known as autoencoder. One of the popular approaches to CFL is to train autoencoder (AE) in ways that enforces some desired attributes. The motivation behind the autoencoding is to reconstruct the input from its encoded representation with features that represent the data [48, 49]. The reconstruction is usually achieved by additive linear (sometimes nonlinear) combination through decoding filters. After training, generating latent encodings for test samples is extremely fast, requiring a simple matrix-vector multiplication.

The model of the neural network AE shown in Fig. 2 aims to reconstruct its input vector using unsupervised learning is given in (3).

$$\hat{\mathbf{x}} = f_{\mathbf{W}, \mathbf{b}}(\mathbf{x}) \approx \mathbf{x} \quad (3)$$

where x is a normalized input vector, $\mathbf{W} = \{\mathbf{W}_1, \mathbf{W}_2\}$, and $\mathbf{b} = \{\mathbf{b}_1, \mathbf{b}_2\}$ respectively represent the weight and biases of the network. It is worth mentioning that the weight matrix \mathbf{W}_2 may optionally be constrained by $\mathbf{W}_2 = \mathbf{W}_1^T$, in which case the autoencoder is said to have tied weights. The concept of tied weights is mainly used to reduce the effective number of parameters. Input data \mathbf{X} is first encoded

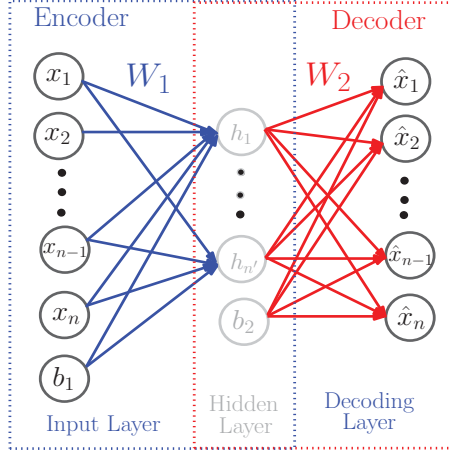


FIGURE 2 – Schematic diagram of a three-layer AE

through \mathbf{W}_1 into features \mathbf{h} . In turn, features \mathbf{h} are mapped back to the data $\hat{\mathbf{X}}$ through \mathbf{W}_2 in accordance with $\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1)$ where $\sigma(\cdot)$ is the activation function. One of the commonly used activation functions is the logistic sigmoid given as $\sigma(\mathbf{x}) = 1/(1 + \exp(-\mathbf{x}))$. In order to solve for parameters \mathbf{W} and \mathbf{b} in (3), the average reconstruction error in (4) serves as the optimization objective.

$$\mathcal{J}_{AE}(\mathbf{W}, \mathbf{b}) = \frac{1}{m} \sum_{i=1}^m \|\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2) - \mathbf{x}_i\|_2^2 \quad (4)$$

We should note that the dictionary learning in sparse coding (1) and AE (4) differ by two aspects. Firstly, the reconstruction error (4) involves mapping of data into itself by two matrices \mathbf{W}_1 and \mathbf{W}_2 , while the same error being the first term of (1) involves one matrix Φ . Secondly, (1) is solved by optimization, while (4) is based on unsupervised learning of \mathbf{h} .

Imposing meaningful limitations on network parameters generally forces AE network to learn representations that attempts to unearth the underlying structure in data. One of such limitations could be limiting the hidden layer size for compressed representation of the input. In this context, constrained AE implies that some constraints such as sparsity, nonnegativity, weight-decay regulariza-

tion, and/or other constraint types are imposed on the learned features. Examples of such constraints are sparsity as in the Sparse Autoencoder (SAE) [50], or nonnegativity and sparsity as in Nonnegativity-Constrained Autoencoder (NCAE) [1,51,52].

Sparsification of features that represent data is increasingly important in learning, especially from big data. This is because sparsity can facilitate efficient and automatic feature selection. In addition, regularization can shrink the magnitude of AE weights and improve the generalization. Therefore, constrained AEs are not only used for feature dimensionality reduction, but also for extracting sparse, part-based features, and for enhancing data understanding.

1. Constrained Autoencoders for Sparse Representation

In AE settings, a network is considered over-sized if the size of the hidden layer is the same or larger than the input vector size n . In this scenario, AE can be forced to learn useful representation if additional constraints are added. These constraints can come in form of regularization to ensure sparsity of the hidden-layer representation or addition of noise in the hidden layer. Sparse representation can provide a interpretation of the input data in terms of a reduced number of parts thereby revealing its hidden structure.

In order to force AE to learn sparse representation, \mathbf{h} is bounded using the Kullback-Leibler (KL) divergence function [53–56]. If $h_j(\mathbf{x}_i)$ denotes the activation (or output) of hidden neuron j due to the input \mathbf{x}_i , the average activation of this particular neuron is given as:

$$\hat{\mathbf{p}}_j = \frac{1}{m} \sum_{i=1}^m h_j(\mathbf{x}_i) \quad (5)$$

If a sparse AE with target activation p is considered, one common method for imposing sparsity is to limit the activation of hidden units using the KL function

[50] as in (6)

$$\text{Sparsity}(p||\hat{\mathbf{p}}) = \sum_{j=1}^{n'} p \log \frac{p}{\hat{p}_j} + (1-p) \log \frac{1-p}{1-\hat{p}_j} \quad (6)$$

One of many functions a regularizer provides is enforcing certain properties on the weights. Note that weight decay term is also added to the cost function of AE as to prevent overfitting [57]. For a conventional sparse autoencoder (SAE) the decay term is given as in (7).

$$\text{Decay}(w) = \frac{\alpha}{2} \sum_{l=1}^2 \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \|w_{ij}^{(l)}\|_2^2 \quad (7)$$

where α is the weight penalty factor, and $w_{ij}^{(l)}$ represents the connection between i th neuron in layer $l-1$ and j th neuron in layer l . The overall cost function based on (1) for SAE using penalization then becomes [50]:

$$J_{SAE}(\mathbf{W}, \mathbf{b}) = J_{AE}(\mathbf{W}, \mathbf{b}) + \beta \text{Sparsity}(p||\hat{\mathbf{p}}) + \text{Decay}(w) \quad (8)$$

where β controls the sparsity penalty term.

The gradient of (8) is computed in (11) for the purpose of updating the network parameters using the backpropagation algorithm [18].

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \xi \frac{\partial}{\partial w_{ij}^{(l)}} J_{SAE}(\mathbf{W}, \mathbf{b}) \quad (9)$$

$$b_i^{(l)} = b_i^{(l)} - \xi \frac{\partial}{\partial b_i^{(l)}} J_{SAE}(\mathbf{W}, \mathbf{b}) \quad (10)$$

where

$$\frac{\partial}{\partial w_{ij}^{(l)}} J_{SAE}(\mathbf{W}, \mathbf{b}) = \frac{\partial}{\partial w_{ij}^{(l)}} J_E(\mathbf{W}, \mathbf{b}) + \beta \frac{\partial}{\partial w_{ij}^{(l)}} \text{Sparsity}(p || \hat{\mathbf{p}}) + g(w_{ij}^{(l)}) \quad (11)$$

$\xi > 0$ is the learning rate and $g(w_{ij}^{(l)})$ is called the decay function and it is given as in (12)

$$g(w_{ij}^{(l)}) = \lambda w_{ij}^{(l)} \quad (12)$$

Other popular methods for sparsifying AE features while preventing overfitting are the dropout technique [58] and family of k -sparse AEs [59,60]. In dropout technique, units and their connections are randomly dropped from the network during training. In effect, dropout tends to prevent neurons from co-adapting thereby leading to good generalization. The concept of k -sparse AE relies on identifying the k neurons with largest activations and setting the rest to zero to prevent overfitting. The k -sparse AE has been found suitable for many dataset because the value k can be tuned to obtain desirable sparsity level in conformity with each dataset.

2. Constrained Autoencoders for Part-based Data Representation

Part-based representation is a way of decomposing data into parts, which when additively combined regenerate the data [34]. As shown in [34], one way of representing data is by shattering it into various distinct pieces in a manner that additive merging of these pieces can reconstruct the original data. Mapping this intuition to AEs, the idea is to sparsely disintegrate data into parts in the encoding layer and additively process the parts to recombine the original data in the decoding layer.

One way to achieve data decomposition with AEs is by using asymmetric piecewise linear weight decay function to constrain network parameters to be nonnegative and the resulting network is called Nonnegative Sparse Autoencoder (NNSAE) [61]. Unlike SAE, NNSAE is trained with an online algorithm and tied weights and linear output activation function. It is capable of extracting nonnegative features for part-based representation of data. The main difference between

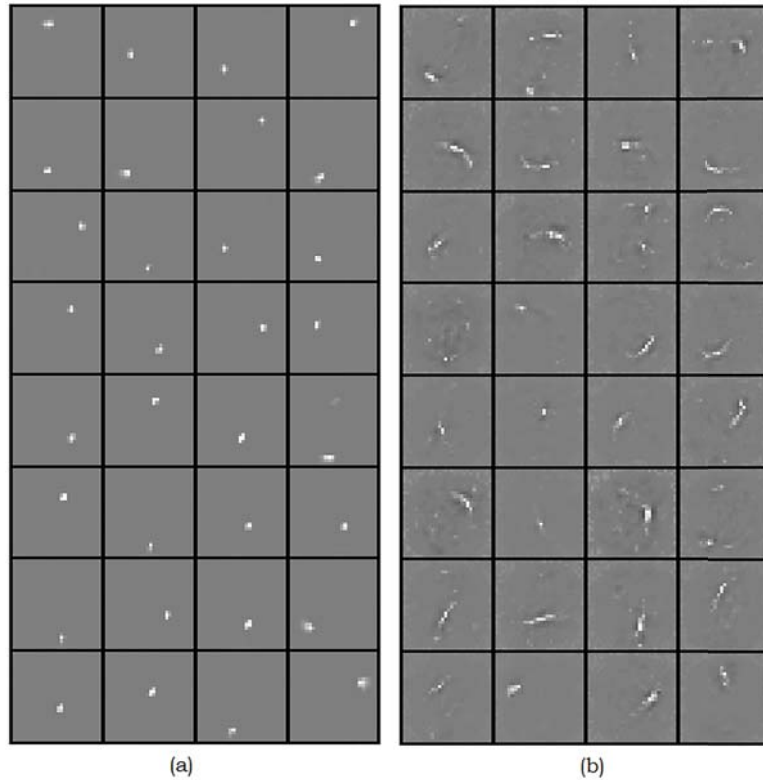


FIGURE 3—RFs or weights of randomly selected 32 out of 196 ($n' = 196$) hidden neurons of (a) NNSAE (b) NCAE trained using MNIST dataset. Black pixels indicate negative, grey pixels indicate zero-valued weights and white pixels indicate positive weights. The range of weights are scaled to $[-1,1]$ and mapped to the gray-color map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color [5].

conventional SAE and NNSAE is in the decay function given in (13)

$$g(w_{ij}^{(l)}) = \begin{cases} -\alpha w_{ij}^{(l)} & w_{ij} < 0 \\ -\beta w_{ij}^{(l)} & w_{ij} \geq 0 \end{cases} \quad (13)$$

where α and β are hyperparameters and $0 \leq \alpha \ll 1$. If $\alpha = 1$, the decay function in (13) ensures a complete prohibition of negative weights. The weight decay function in (12) for SAE can also be viewed as imposing Gaussian prior distribution on network weights while NNSAE uses a weight decay mechanism that assumes a virtually deformed Gaussian prior that is skewed with respect to the sign of the weight. It must be noted that $\alpha = \beta$, (13) is equivalent to (12).

Another variant of NNSAE is the Nonnegativity-Constrained AE (NCAE) [51], which also aim at eliminating negative weight through regularization. This is achieved by imposing nonnegativity constraint in form of a penalty term by replacing the decay term in (8) with (14)

$$\text{Decay}(w) = \begin{cases} \frac{\alpha}{2} \sum_{l=1}^2 \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (w_{ij}^{(l)})^2 & w_{ij} < 0 \\ 0 & w_{ij} \geq 0 \end{cases} \quad (14)$$

where $\alpha > 0$ is a nonnegativity-constraint weight penalty factor and decay function is given as

$$g(w_{ij}^{(l)}) = \begin{cases} -\alpha w_{ij}^{(l)} & w_{ij} < 0 \\ 0 & w_{ij} \geq 0 \end{cases} \quad (15)$$

It is worthy to note that the decay function of NNSAE in (13) is a generalization of both SAE when $\beta = \alpha$ as in (12) and NCAE when $\beta = 0$ as in (15).

Part-based data decomposition is illustrated using NCAE and NNSAE trained on MNIST digit and both AEs have 196 hidden neurons. Weights of trained networks are portrayed as images of receptive fields (RFs). Figures 3a and b show the RFs learned by NNSAE and NCAE, respectively. It can be observed that the RFs learned are select parts of handwritten digits such as strokes and dots. The learned featured are localized and tend to look like parts of digits. Part-based

representation is also illustrated in Figure 4 using NCAE trained on MNIST handwritten characters. NCAE with linear decoder architecture (that is, the activation function $\sigma(\cdot)$ for decoding layer is identity function) was trained in such a manner that the column of \mathbf{W}_1 are coerced to be sparse. RFs and the decoding filters are displayed on the right hand side. A test image of digit 6 shown is filtered through the network and activations \mathbf{h} are listed. The vector of activations is very sparse since it only stimulates 4 out of 196 RFs. The test sample can be reconstructed by additively combining four outputs of decoding filters scaled with magnitudes of select \mathbf{h} values.

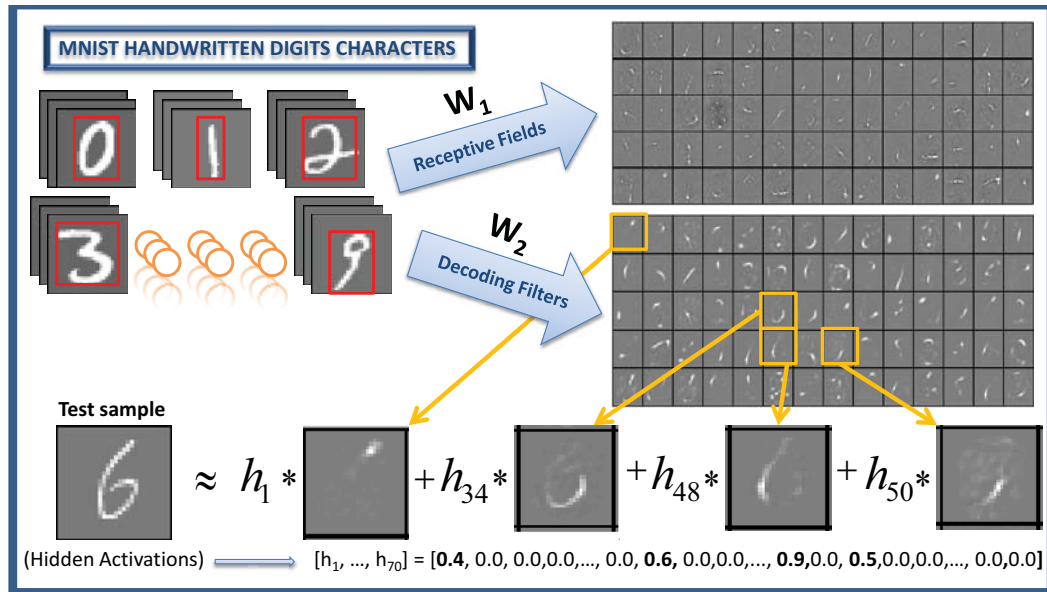


FIGURE 4: Representation of test image as a linear combination of 4 out of 196 constrained RFs and decoding filters learned from MNIST dataset using NCAE with linear output activation function. Input consist of 784 values corresponding to a 28×28 pixel image. Only 70 RFs with largest activations to test image "6" and their corresponding decoding filters are shown. The RFs and the decoding filters are rescaled and portrayed as images on the right hand side. Black pixels indicate negative, and white pixels indicate positive weights. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. The biases are not shown [5].

CHAPTER II

CONSTRAINED AUTOENCODERS FOR ENHANCED DATA UNDERSTANDING

It is a general belief that humans analyze complex interactions by breaking them into isolated and understandable hierarchical concepts. Methods for learning understandable models, such as decision tree [62] extract only flat data descriptions that lack hierarchical concepts. On the other hand, methods that build model with hierarchical structure usually extract features that are difficult to understand and/or interpret [63, 64]. As shown in [22], one way to reconcile the requirements of hierarchical organization and easier human understandability of concepts in neural networks is by imposing nonnegative-only weights. Moreover, the emergence of part-based representation in human cognition can be conceptually tied to the nonnegativity constraints [34]. Although deep feedforward neural networks have the capability to model multi-level abstraction of data, they are difficult to train and the understandability of features they learn is very limited [65].

Owing to unsupervised pretraining using AE or Restricted Boltzmann machine [66] and better initialization heuristics [67–70], the training difficulties have been alleviated. Some heuristics have also attempted to address the problem of understandability by extracting rules from neural network for individual neurons. These heuristics, however, differentiate rules from individual neuron and their states using special symbols, which in turn increases the opaqueness of the extracted rules. In addition, these symbolic rules become more complicated for deep neural networks with no meaningful interpretations. Also, the rule extraction process has been shown to be computationally expensive [21].

The issue of understandability is addressed by drawing inspiration from the idea of NMF [34] and sparse coding [71] and appropriately enforcing space non-negative features. As highlighted in [22], understandability of features learned by neural network models can be fostered by enforcing weights in the network to be nonnegative. This would allow easier inspection and interpretation by eliminating cancelations of incoming neuron signals. In addition, there are neural activities for a subset of hidden units that are strongly correlated with the input and threshold of this correlation is controlled by the bias term. The main shortcoming in [22] is that sparse nonnegative features are obtained by directly mapping negative weights to zero, in effect, significantly deteriorates the performance of the entire network. That is, a portion of the performance is traded with extraction of understandable features. In addition, the heuristic was developed and customized for shallow neural networks. Of special interest to the work in this chapter is the extraction of understandable deep neural network features that preserves the overall network performance.

A. L_1/L_2 -Nonnegativity Constrained Sparse Autoencoder (L_1/L_2 -NCSAE)

As earlier shown using NNSAE [61] and NCAE [51], negative weight can be eliminated from neural network models in an online fashion through regularization. This is achieved by regularizing the learning cost function with appropriate penalty term. However, a close scrutiny of the weight distribution of both the encoding and decoding layer enforced by the penalty function of NCAE in (14) reveals that many weights are still negative despite imposing nonnegativity constraints. The reason for this is that the original L_2 norm used in NCAE penalizes the negative weights with big magnitudes stronger than those with smaller magnitudes. This forces a good number of the weights to take on small negative values. From experiments carried out using NCAE, these negative weights are essential

for achieve good performance in terms of both reconstruction and classification. It will be shown that additional L_1 term can be used to even out this occurrence, that is, the additional L_1 penalty forces most of the small-valued negative weights to become zero. The resulting architecture extracts features that are more sparse with improved reconstruction error and is renamed as L_1/L_2 -Nonnegativity Constrained Sparse Autoencoder (L_1/L_2 -NCSAE). The penalty term of NNSAE [61] in (13) on the other hand also extracts strictly nonnegative feature, however, it does so in a way that deteriorates the classification accuracy when used to pretrain a deep network due of its relatively high reconstruction error.

In order to encourage higher degree of nonnegativity in network's weights, a penalty term in (16) is added to the objective function resulting in the cost function expression for L_1/L_2 -NCSAE [1]. The negative weights are regularized by minimizing their absolute values (L_1 norm) and their squares (L_2 norm). The combined action of the L1 and L2 penalties is that they both select only the important negative weights and limits their magnitude. This thus employ a penalty-based negative weight pruning mechanism.

$$Decay(w_{ij}) = \begin{cases} \alpha_1 \Gamma(w_{ij}, \kappa) + \frac{\alpha_2}{2} \|w_{ij}\|^2 & w_{ij} < 0 \\ 0 & w_{ij} \geq 0 \end{cases} \quad (16)$$

where α_1 and α_2 are L_1 and L_2 nonnegativity-constraint weight penalty factors, respectively. The decay function $d(w_{ij})$ is a composite function denoting the derivative of $Decay(w_{ij})$ (16) with respect to w_{ij} as in (17).

$$g(w_{ij}) = \begin{cases} \alpha_1 \nabla_{\mathbf{w}} \|w_{ij}\| + \alpha_2 w_{ij} & w_{ij} < 0 \\ 0 & w_{ij} \geq 0 \end{cases} \quad (17)$$

1. Implication of imposing nonnegative parameters with composite decay function

The graphical illustration of the relation between the weight distribution and the composite decay function is shown in Fig. 6. Ideally, addition of Frobenius norm of the weight matrix ($\alpha\|\mathbf{W}\|_F^2$) in (12) to the reconstruction error imposes a Gaussian prior on the weight distribution as shown in curve G_3 in Figure 6a. However, using the composite function in (17) results in imposition of positively-skewed deformed Gaussian distribution as in curves G_1 and G_2 . The degree of nonnegativity can be adjusted using parameters α_1 and α_2 . Both parameters have to be carefully chosen to enforce nonnegativity while simultaneously ensuring good supervised learning outcomes. The effect of L_1 ($\alpha_2 = 0$), L_2 ($\alpha_1 = 0$) and L_1/L_2 ($\alpha_1 \neq 0$ and $\alpha_2 \neq 0$) nonnegativity penalty terms on weight updates for weight distributions G_1 , G_2 and G_3 are respectively shown in Figure 6c,d, and b. It can be observed for all the three distributions that L_1/L_2 regularization enforces stronger weight decay than individual L_1 and L_2 regularization. Other observation from Figure 6 is that the more positively-skewed the weight distribution becomes, the lesser the weight decay function.

The consequences of minimizing the reconstruction under the regularization in (16) are that: (i) the average reconstruction error is reduced (ii) the sparsity of the hidden layer activations is increased because more negative weights are forced to zero thereby leading to sparsity enhancement, and (iii) the number of nonnegative weights is also increased. As earlier mentioned, the resultant effect of penalizing the weights simultaneously with L_1 and L_2 norm is that they both select only the important negative weights and limits their magnitude. However, the L_1 norm in (16) and (17) is non-differentiable at the origin, and this can lead to numerical instability during simulations. To circumvent this drawback, one of the well known smoothing function that approximates L_1 norm is utilized. The

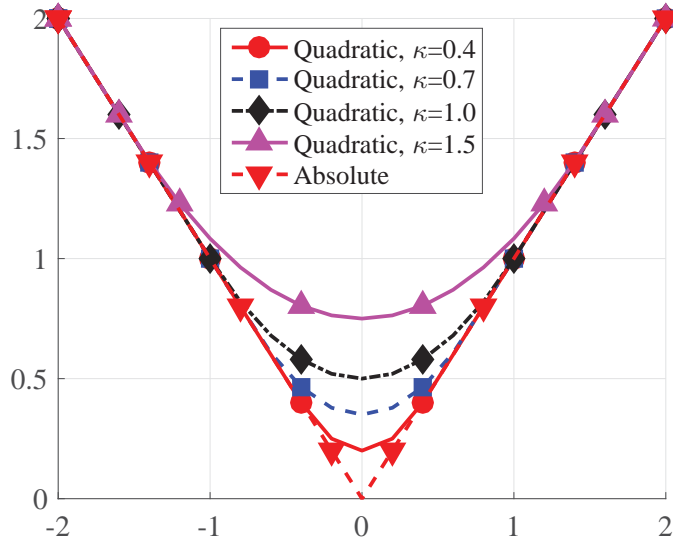


FIGURE 5–Absolute function approximation using quadratic smoothing functions with $\kappa = 0.4, 0.7, 1.0$ and 1.5

approximation is defined as follows: Given any finite dimensional vector \mathbf{z} and positive constant κ , the following smoothing function approximates L_1 norm:

$$\Gamma(\mathbf{z}, \kappa) = \begin{cases} \|\mathbf{z}\| & \|\mathbf{z}\| > \kappa \\ \frac{\|\mathbf{z}\|^2}{2\kappa} + \frac{\kappa}{2} & \|\mathbf{z}\| \leq \kappa \end{cases} \quad (18)$$

with gradient

$$\nabla_{\mathbf{z}}\Gamma(\mathbf{z}, \kappa) = \begin{cases} \frac{\mathbf{z}}{\|\mathbf{z}\|} & \|\mathbf{z}\| > \kappa \\ \frac{\mathbf{z}}{\kappa} & \|\mathbf{z}\| \leq \kappa \end{cases} \quad (19)$$

2. Experimental Results

a. Unsupervised Feature Learning of Image Data In the first set of experiments, three-layer L_1/L_2 -NCSAE, NCAE [51], DpAE [72], and conventional SAE network with 196 hidden neurons were trained using MNIST dataset of handwritten digits and their ability to discover patterns in high dimensional data are com-

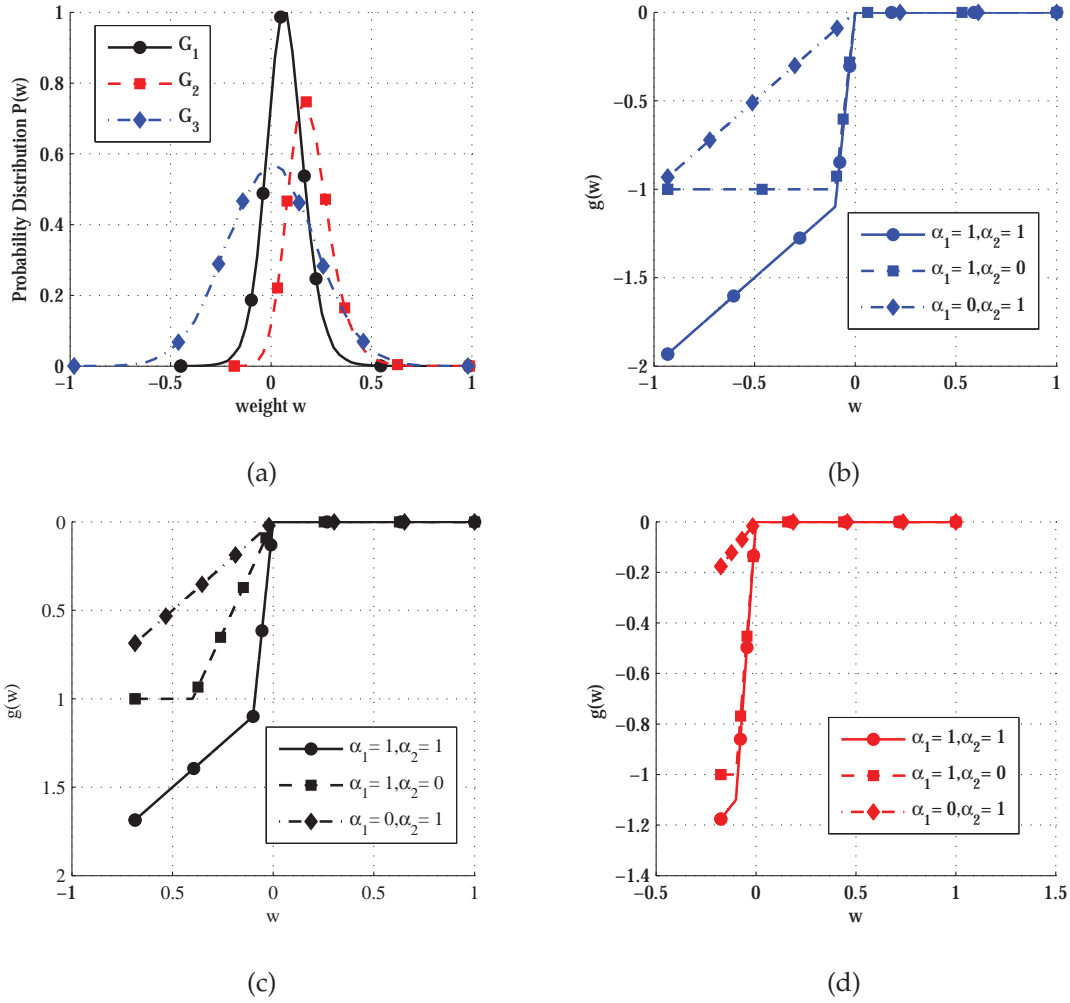


FIGURE 6: (a) Symmetric (G_3) and skewed (G_1 and G_2) weight distributions. Decay function with three values of α_1 and α_2 for weight distribution (b) G_3 (c) G_1 and (d) G_2 . [1]

pared. These experiments were run one time and recorded. The encoding weights $\mathbf{W}^{(1)}$, also known as receptive fields or filters as in the case of image data, are reshaped, scaled, centered in a 28×28 pixel box and visualized. The filters learned by L_1/L_2 -NCSAE are compared with that learned by its counterparts, NCAE and SAE. It can be easily observed from the results in Figure 25 that L_1/L_2 -NCSAE learned receptive fields that are more sparse and localized than those of SAE, DpAE, and NCAE. It is remarked that the black pixels in both SAE and DpAE features are results of the negative weights whose values and numbers are reduced in NCAE with nonnegativity constraints, which are further reduced by imposing an additional L_1 penalty term in L_1/L_2 -NCSAE as shown in the histograms located on the right side of the figure. Although the penalty function in NCAE is a special case of that in L_1/L_2 -NCSAE (obtained by setting α_1 to zero), a close scrutiny of the weight distribution of both the encoding and decoding layer in NCAE reveals that many weights are still negative despite imposing nonnegativity constraints. The reason for this is that the original L_2 norm used in NCAE penalizes the negative weights with big magnitudes stronger than those with smaller magnitudes. This forces a good number of the weights to take on small negative values. L_1/L_2 -NCSAE uses additional L_1 to even out this occurrence, that is, the L_1 penalty forces most of the negative weights to become nonnegative.

In the case of L_1/L_2 -NCSAE, tiny strokes and dots which constitute the basic part of handwritten digits, are unearthed compared to SAE, DpAE, and NCAE. Most of the features learned by SAE are major parts of the digits or the blurred version of the digits, which are obviously not as sparse as those learned by L_1/L_2 -NCSAE. Also, the features learned by DpAE are fuzzy compared to those of L_1/L_2 -NCSAE which are sparse and distinct. Therefore, the achieved sparsity in the encoding can be traced to the ability of L_1 and L_2 regularization in enforcing high degree of weights' nonnegativity in the network.

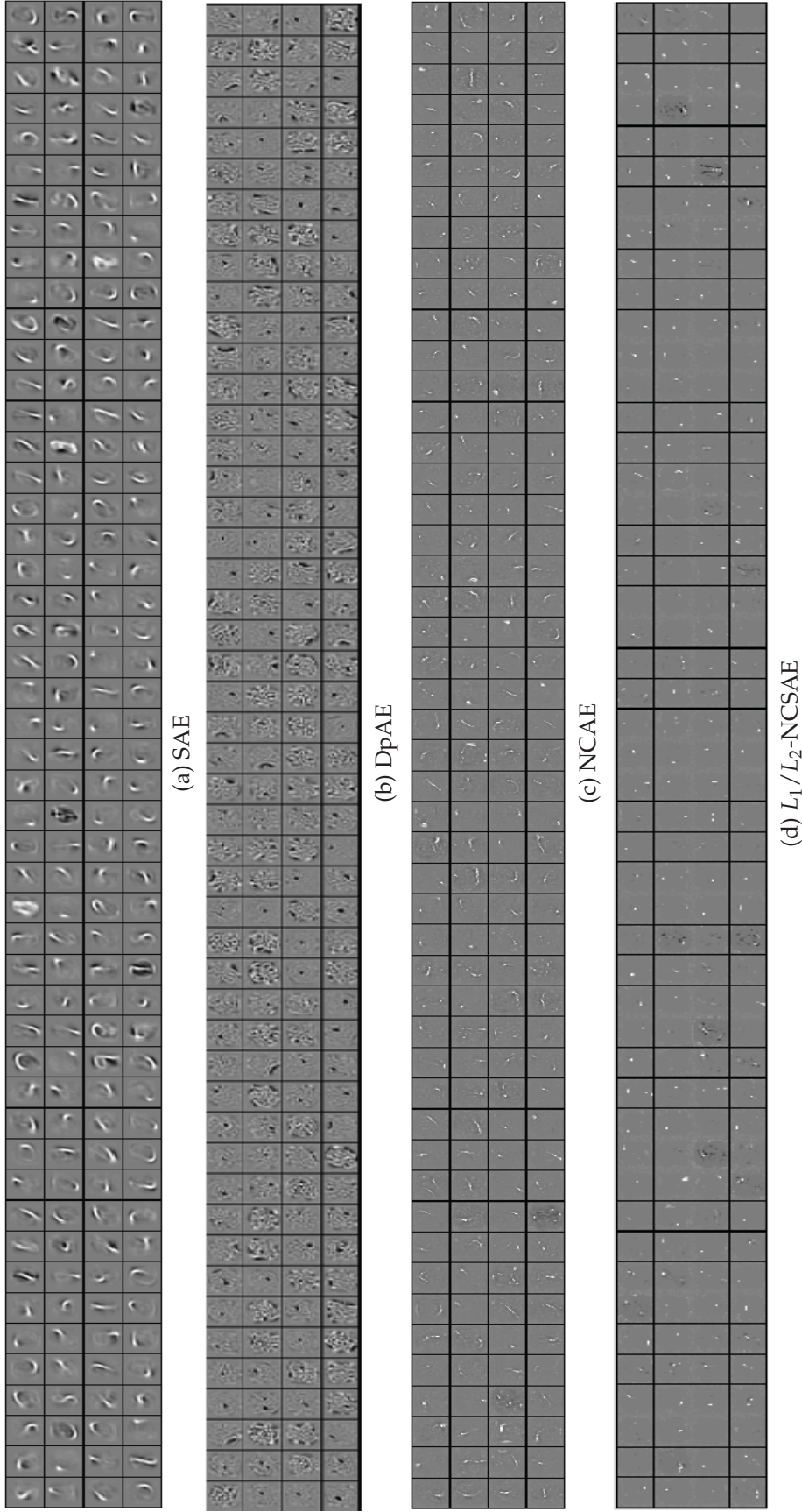


FIGURE 7: 196 receptive fields ($W^{(1)}$) learned from MNIST digit data set using (a) SAE, (b) DpAE (c) NCAE, and (d) L_1/L_2 -NCSAE. Black pixels indicate negative, and white pixels indicate positive weights. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color [1].

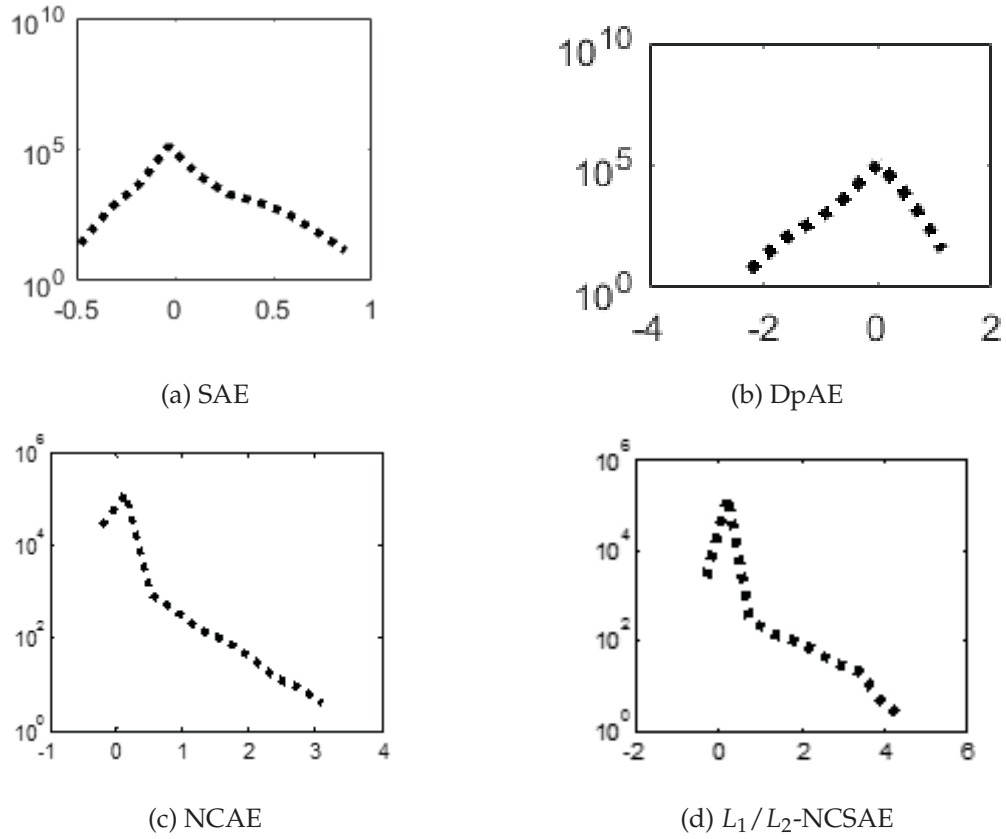


FIGURE 8: Encoding weights ($W^{(1)}$) histograms learned from MNIST digit data set using (a) SAE, (b) DpAE (c) NCAE, and (d) L_1/L_2 -NCSAE [1].

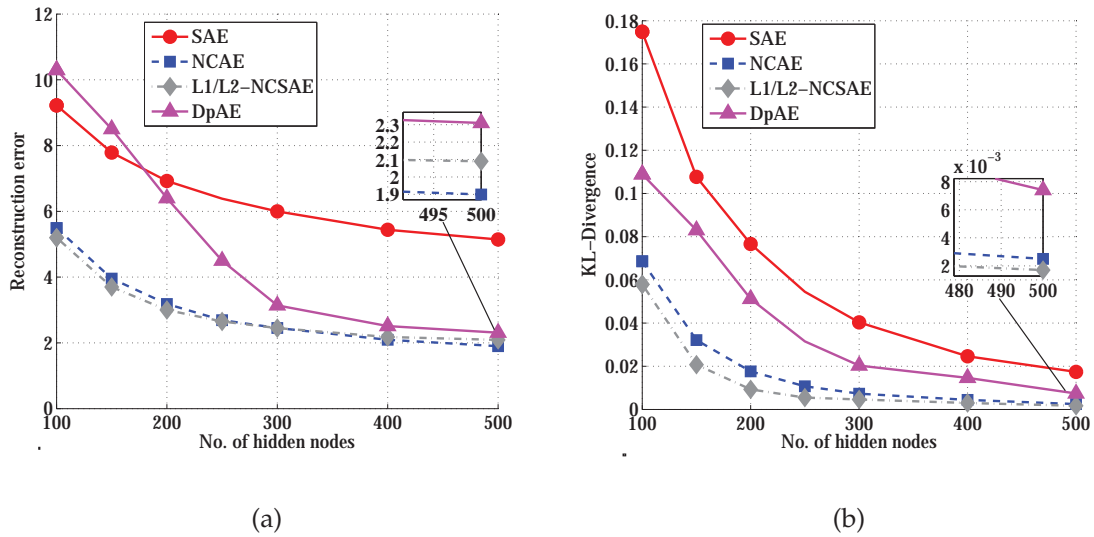


FIGURE 9: (a) Reconstruction error and (b) Sparsity of hidden units measured by KL-divergence using MNIST train dataset with $p = 0.05$ [1].

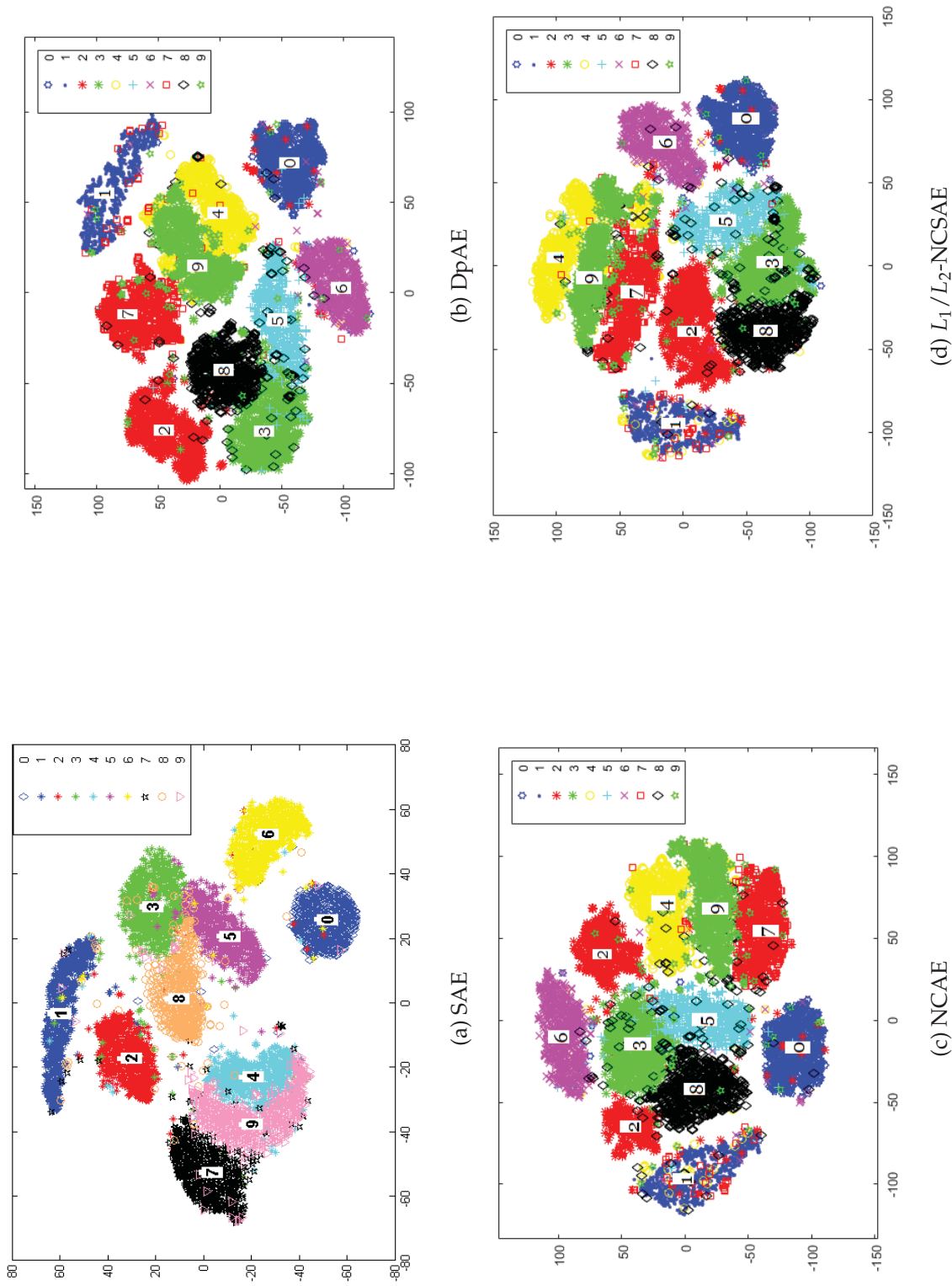
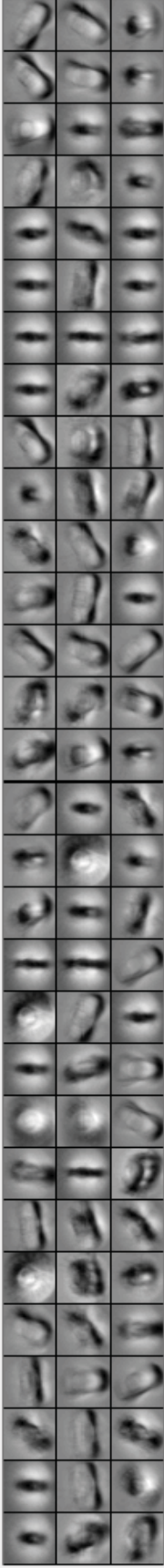
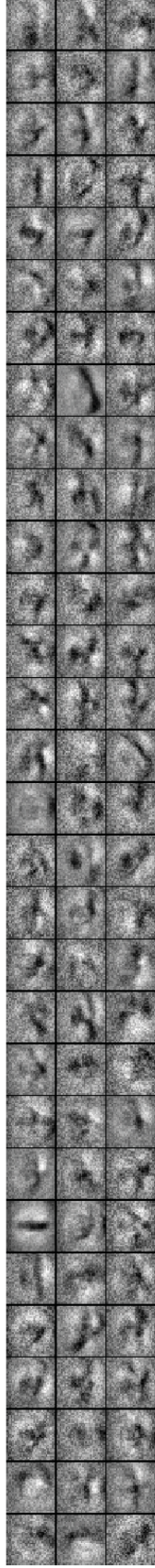


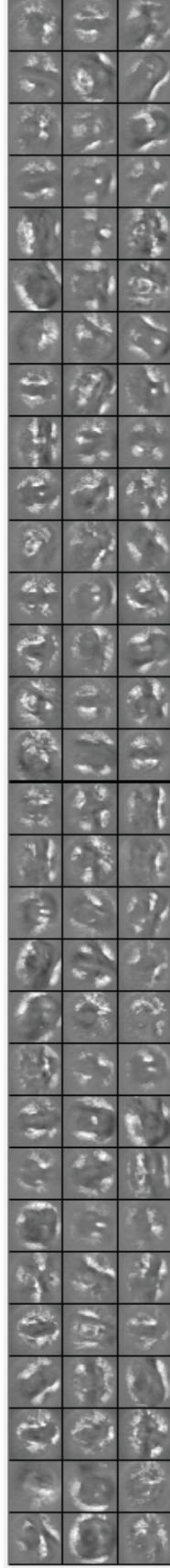
FIGURE 10: t-SNE projection [6] of 196D representations of MNIST handwritten digits using (a) SAE (b) DpAE (c) NCAE (d) L_1/L_2 -NCSAE [1].



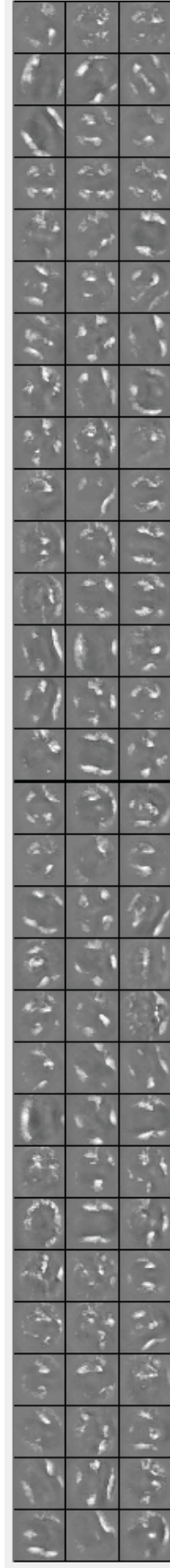
(a) SAE



(b) DpAE



(c) NCAE



(d) L_1/L_2 -NCSAE

FIGURE 11: Weights of randomly selected 90 out of 200 receptive filters of (a) SAE (b) DpAE (c) NCAE, and (d) L_1/L_2 -NCSAE using NORB dataset. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w <= -1$ is assigned to black, $w = 0$ to grey, and $w >= 1$ is assigned to white color [1].

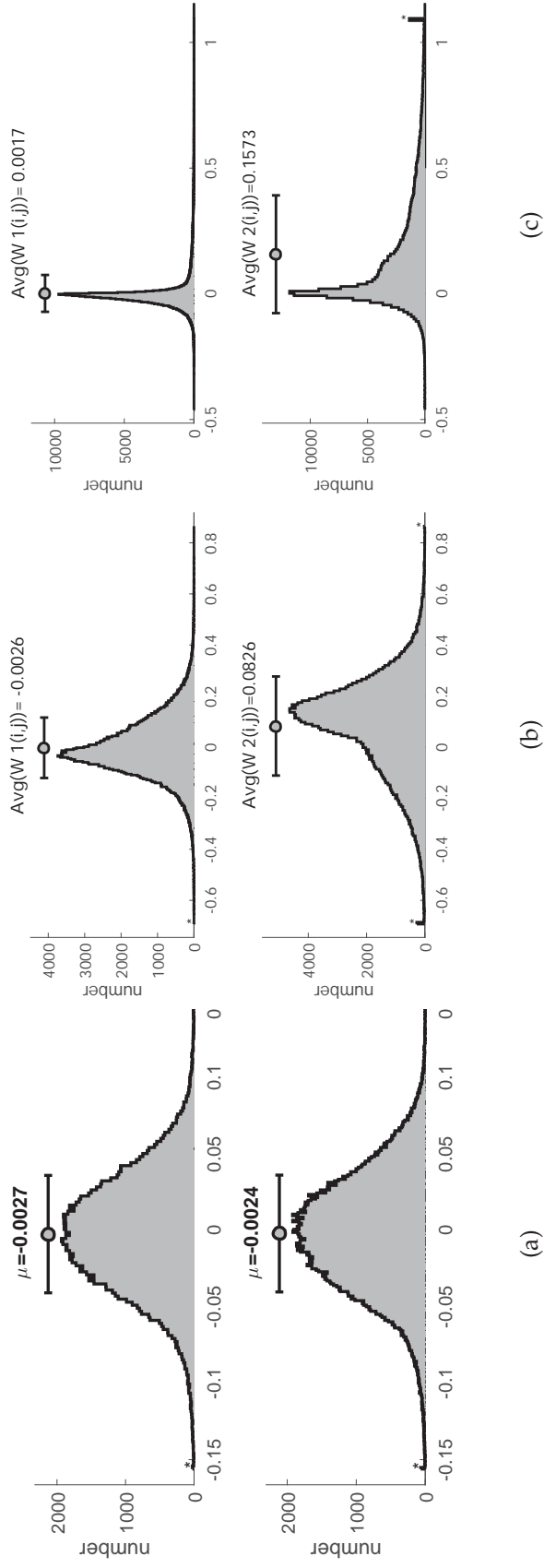


FIGURE 12: The distribution of 200 encoding ($\mathbf{W}^{(1)}$) and decoding filters ($\mathbf{W}^{(2)}$) weights learned from NORB dataset using (a) DpAE (b) NCAE (c) L_1/L_2 -NCSAE [1].

Likewise in Figure 9a, L_1/L_2 -NCSAE with other AEs are compared in terms of reconstruction error, while varying the number of hidden nodes. As expected, it can be observed that L_1/L_2 -NCSAE yields a reasonably lower reconstruction error on the MNIST training set compared to SAE, DpAE, and NCAE. Although, a close scrutiny of the result also reveals that the reconstruction error of L_1/L_2 -NCSAE deteriorates compared to NCAE when the hidden size grows beyond 400. However on the average, L_1/L_2 -NCSAE reconstructs better than other AEs considered. It can also be observed that DpAE with 50% dropout has high reconstruction error when the hidden layer size is relatively small (100 or less). This is because the few neurons left are unable to capture the dynamics in the data, which subsequently results in underfitting the data. However, the reconstruction error improves as the hidden layer size is increased. Lower reconstruction error in the case of L_1/L_2 -NCSAE and NCAE is an indication that nonnegativity constraint facilitates the learning of parts of digits that are essential for reconstructing the digits. In addition, the KL-divergence sparsity measure reveals that L_1/L_2 -NCSAE has more sparse hidden activations than SAE, DpAE and NCAE for different hidden layer size as shown in Figure 9b. Again, averaging over all the training examples, L_1/L_2 -NCSAE yields less activated hidden neurons compared to its counterparts.

Also, using t-distributed stochastic neighbor embedding (t-SNE) to project the 196-D representation of MNIST handwritten digits to 2D space, the distribution of features encoded by 196 encoding filters of SAE, DpAE, NCAE, and L_1/L_2 -NCSAE are respectively visualized in Figures 10a, b, c, and d. A careful look at Figure 10b reveals that digits "4" and "9" are overlapping in DpAE, and this will inevitably increase the chance of misclassifying these two digits. It can also be observed in Figure 10c corresponding to NCAE that digit "2" is projected with two different landmarks. In sum, the manifolds of digits with L_1/L_2 -NCSAE are more separable than its counterpart as shown in Figure 10d, aiding the classifier to map

out the separating boundaries among the digits more easily.

In the second experiment, SAE, NCAE, L_1/L_2 -NCSAE, and DpAE with 200 hidden nodes were trained using the NORB normalized-uniform dataset. The NORB normalized-uniform dataset, which is the second dataset, contains 24,300 training images and 24,300 test images of 50 toys from 5 generic categories: four-legged animals, human figures, airplanes, trucks, and cars. The training and testing sets consist of 5 instances of each category. Each image consists of two channels, each of size 96×96 pixels. The inner 64×64 pixels of one of the channels cropped out and resized using bicubic interpolation to 32×32 pixels that form a vector with 1024 entries as the input. Randomly selected weights of 90 out of 200 neurons are plotted in Figure 19. It can be seen that L_1/L_2 -NCSAE learned more sparse features compared to features learned by all the other AEs considered. The receptive fields learned by L_1/L_2 -NCSAE captured the real actual edges of the toys while the edges captured by NCAE are fuzzy, and those learned by DpAE and SAE are holistic. As shown in the weight distribution depicted in Figure 12, L_1/L_2 -NCSAE has both its encoding and decoding weights centered around zero with most of its weights positive when compared with those of DpAE and NCAE that have weights distributed almost even on both sides of the origin.

b. Unsupervised Semantic Feature Learning from Textual Data In this experiment DpAE, NCAE, and L_1/L_2 -NCSAE are evaluated and compared based on their ability to extract semantic features from text data, and how they are able to discover the underlined structure in text data. For this purpose, the Reuters-21578 text categorization dataset with 200 features is utilized to train all the three types of AEs with 20 hidden nodes. A subset of 500 examples belonging to categories "grain", "crude", and "money-fx" was extracted from the test set. The experiments were run three times, averaged and recorded. In Figure 13, the 20-dimensional representations of the Reuters data subset using DpAE, NCAE, and L_1/L_2 -NCSAE are

visualized. It can be observed that L_1/L_2 -NCSAE is able to disentangle the documents into three distinct categories with more linear manifolds than NCAE. In addition, L_1/L_2 -NCSAE is able to group documents that are closer in the semantic space into the same categories than DpAE that finds it difficult to group the documents into any distinct categories with less overlap.

c. Illustration of Understandable Feature Extraction by L_1/L_2 -NCSAE In vision-related task, basis vectors sensitive to a region in an image and to specific stimuli are called RFs. Figure 14 illustrates the idea of constrained RF using L_1/L_2 Nonnegativity Constrained Sparse Autoencoder (L_1/L_2 -NCSAE) [1, 5] trained on synthetic data that comprises of three images as depicted. L_1/L_2 -NCSAE is a specialized AE architecture capable of extracting nonnegative features (and nonnegative RFs) as shown. (L_1/L_2 -NCSAE is explained in more detail in Section II). Input $\mathbf{X} \in \mathbb{R}^{25 \times 3}$ consists of three 5×5 images. The three RFs are rows of weight matrix $\mathbf{W}_1 \in \mathbb{R}^{3 \times 25}$. For visualization they are resized to match the square input image (both the inputs and the 25 weights of hidden neurons are presented as images). Neurons' outputs are the Activation Scores computed as the dot product of each RF and the input pattern.

It can be observed from Figure 14 that first RF (1st row of Activation Scores table) is most sensitive to first T-shaped image and captures features that strongly react to T-shaped image. Similarly, second RF reacts mostly to the second input pattern. Likewise the third input pattern stimulates the third RF and maximally activates it with largest magnitude. It is remarked that using appropriate bias and softmax layer, first RF helps in classifying first image, second RF for classifying second one, and lastly, the third RF for third image. This observation is consistent with what is observed at the output of the softmax neurons given as Softmax scores in Figure 14. The Softmax layer is also known as the classification or output layer [22, 73, 74]. Softmax scores are computed as $\text{Softmax}(\mathbf{W}_C \cdot \mathbf{h} + \mathbf{b}_C)$, where

Softmax(\mathbf{v}) maps a vector \mathbf{v} into a vector of values according to $\text{Softmax}(\mathbf{v})_i = \exp(\mathbf{v}_i) / \sum_{j=1}^c \exp(\mathbf{v}_j)$, \mathbf{W}_C and \mathbf{b}_C are respectively the matrix of weights and vector of bias values for the classification layer, and c is the size of the output vector equal to the number of classes.

The concept of RFs is not restricted only to visual information but also to many pattern recognition tasks such as those involving audio and semantic data.

B. Deep Learning of Understandable Features using Cascaded L_1/L_2 -NCSAE

Deep networks (DNs) based on AEs are created by stacking pretrained AEs layer by layer, followed by a supervised fine-tuning. They are able to extract salient features from input data through greedy, unsupervised, layerwise training algorithm. In deep autoencoding, cascade of AEs is trained to detect feature hierarchies from training samples to generate latent encodings. Each additional layer of AE adds an additional abstract representation of the input. Deep AE architectures invariably result in lower layerwise reconstruction error and a better representation of the input [75]. One of the key factors that contributes to high performance of deep network is the appropriate initialization achieved by pretraining each layer. In deep feature learning, AEs are stacked over one another with the output of each layer feeding the input of the successive layer. A greedy layer-wise training approach is adopted to train each successive layer [55]. The activations of the last AE are then used as the input to the Softmax layer (SMC), a supervised classifier as shown in Figure 15. The parameters obtained after the training yield the transformation $f: \mathcal{R}^{d_x} \rightarrow \mathcal{R}^{d_{h^{(L)}}}$ which maps input to new high level feature representation $h^{(L)}$. Since the activation of the last AE is the input to the Softmax layer, the training input of the supervised learning (classification) is given as $\{h^{(L)(k)}, y^{(k)}\}_{k=1}^m$ which is the pair of high level feature representation and its corresponding label. In the case of nonnegativity-constrained AEs, it must be noted that weights in the softmax layer are also nonnegativity constrained [1].

Deep autoencoding architectures offer a way to combine many simple transformations into a more complicated one, but they do not enhance understandability of data unless the base model, which is stacked, is an understandable one. It is shown in this section that deep understandable features can be learnt by cascading L_1/L_2 -NCSAEs followed by a classifier. Each layer of L_1/L_2 -NCSAE is constrained to extract additive part-based features with high degree of understand-

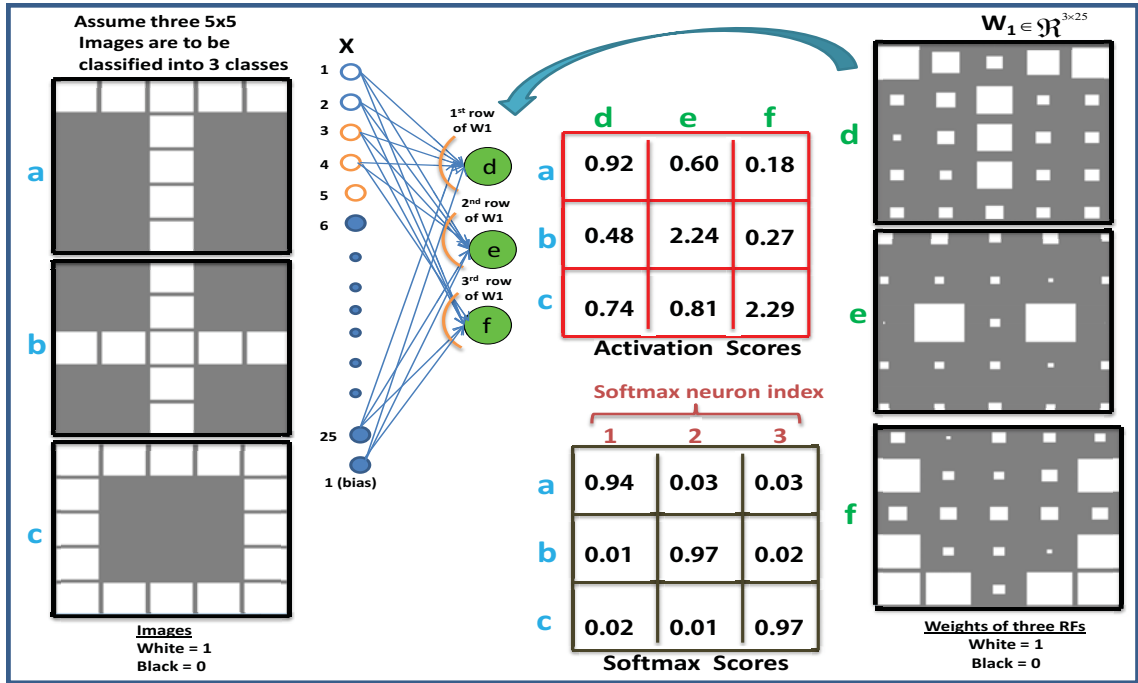


FIGURE 14: Illustration of constrained (non-negative) RF feature extraction using a L_1/L_2 -NCSAE trained on synthetic data with 3 images (left). The RFs learned (right) are rescaled and portrayed as images. The range of weights are scaled to $[-1,1]$ and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. That is, black pixels indicate negative, and white pixels indicate positive weights. The dot product of each RF and input pattern shown as Activation Scores and the outputs of Softmax layer as Softmax scores. a,b, and c are the indices of input images; d,e, and f are the RFs indices. The biases are not shown [5].

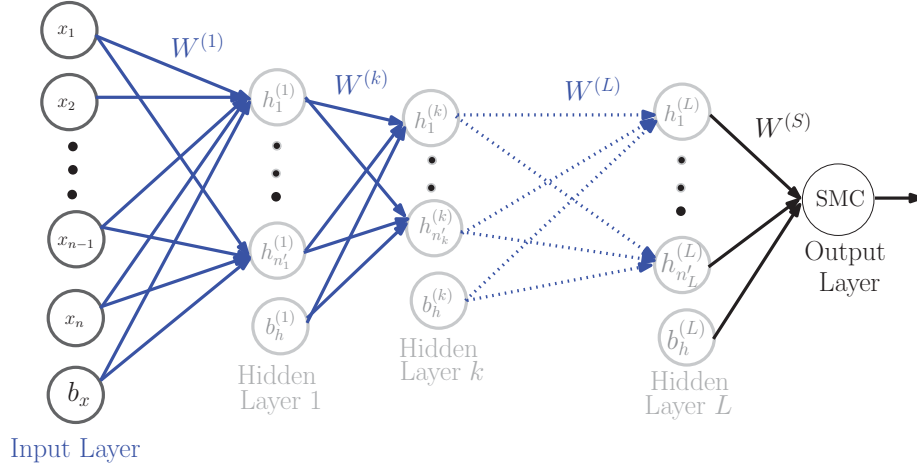


FIGURE 15–Schematic diagram of a deep AE of $L + 1$ layers constructed using Stacked Sparse Autoencoder (SSAE) and Softmax Classifier (SMC).

ability. The input to a particular L_1/L_2 -NCSAE is the encoding of the preceding L_1/L_2 -NCSAE and the classifier simply combines the encodings of the last L_1/L_2 -NCSAE additively and in direct proportion to their weights. It must be noted that the nonnegativity constraint is also imposed on the weights of the classification layer in order to generate a model that is understandable end-to-end. One of the most important advantages of L_1/L_2 -NCSAE-based deep network is its ability to be understandable and still show competitive performance on all the benchmark datasets considered.

1. Image Classification with Enhanced Interpretability

In this experiment, the subset 1, 2 and 6 from the MNIST handwritten digits as extracted for the purpose of understanding how the deep network constructed using L_1/L_2 -NCSAE processes and classifies its input. For easy interpretation, a small deep network was constructed and trained by stacking two AEs with 10 hidden neurons each and 3 softmax neurons. The number of hidden neurons was chosen to obtain reasonably good classification accuracy while keeping the network reasonably small. The network is intentionally kept small because the full

MNIST data would require larger hidden layer size and this may limit network interpretability. An image of digit 2 is then filtered through the network, and it can be observed in Figure 16 that sparsification of the weights in all the layers is one of the aftermath of nonnegativity constraints imposed on the network. Another observation is that most of the weights in the network have been confined to non-negative domain, which removes opaqueness of the deep learning process. It can be seen that the fourth and seventh RFs of the first AE layer have dominant activations (with activation values 0.12 and 0.13 respectively) and they capture most information about the test input. Also, they are able to filter distinct part of input digit. The outputs of the first layer sigmoid constitute higher level features extracted from test image with emphasis on the fourth and seventh features. Subsequently in second layer the second, sixth, eighth, and tenth neurons have dominant activations (with activation values 0.0914, 0.0691, 0.0607, and 0.0606 respectively) because they have stronger connections with the dominant neurons in first layer than the rest. Lastly in the softmax layer, the second neuron was 99.62% activated because it has strongest connections with the dominant neurons in second layer thereby classifying the test image as "2".

The fostering of interpretability is also demonstrated using a subset of NORB normalized-uniform dataset [76] with class labels "four-legged animals", "human figures", "airplanes". The 1024-10-5-3 network configuration was trained on the subset of the NORB data using two stacked L_1/L_2 -NCSAEs and a Softmax layer. Figure 17b shows the randomly sampled test patterns and the weights and activations of first and second AE layer are shown in Figure 17a. The bar charts indicate the activations of hidden units for the sample input patterns. The features learned by units in each layer are localized, sparse and allow easy interpretation of isolated data parts. The features mostly show nonnegative weights making it easier to visualize to what input object patterns they respond. It can be seen that units in the

network discriminate among objects in the images and react differently to input patterns. Third, sixth, eighth, and ninth hidden units of layer 1 capture features that are common to objects in class "2" and react mainly to them as shown in the first layer activations. Also, the features captured by the second layer activations reveal that second and fifth hidden units are mainly stimulated by objects in class "2".

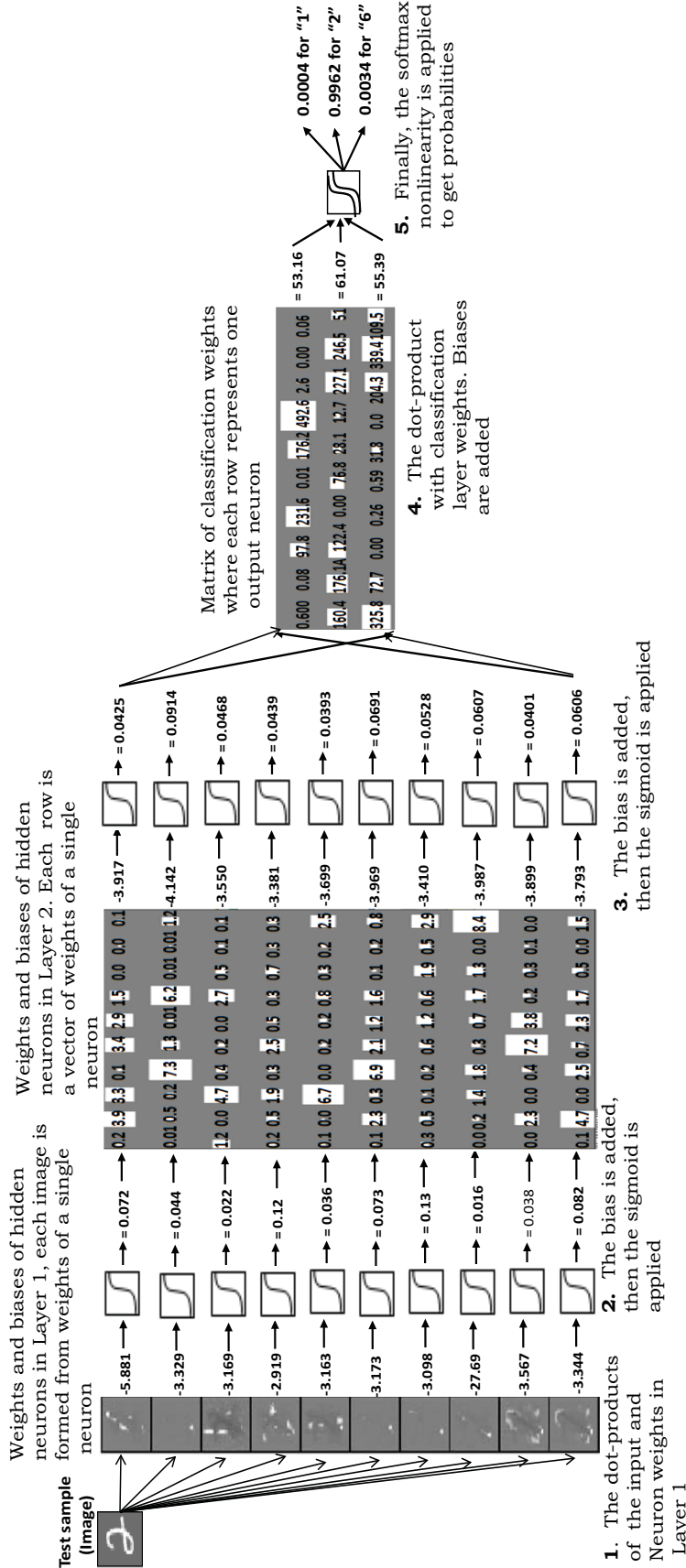


FIGURE 16: Filtering the signal through the L_1/L_2 -NCSAE trained using the reduced MNIST data set with class labels 1, 2 and 6. The test image is a 28×28 pixels image unrolled into a vector of 784 values. Both the input test sample and the receptive fields of the first autoencoding layer are presented as images. The weights of the output layer are plotted as a diagram with one row for each output neuron and one column for every hidden neuron in $(L - 1)^{th}$ layer. The architecture is 784-10-3. The range of weights are scaled to $[-1, 1]$ and mapped to the grayscale map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. That is, black pixels indicate negative, grey pixels indicate zero-valued weights and white pixels indicate positive weights [1].

The outputs of Softmax layer represent the *a posteriori* class probabilities for a given sample and are denoted as Softmax scores. An important observation from Figure 17a,b, and c is that hidden units in both layers did not capture significant representative features for class "1" white color-coded test sample. This is one of the reasons why it is misclassified into class "3" with probability of 0.57. The argument also goes for class "1" dark-grey color-coded test sample misclassified into class "3" with probability of 0.60. In contrast, hidden units in both layers capture significant representative features for class "2" test samples of all color codes. This is why all class "2" test samples are classified correctly with high probabilities as shown in Figure 17d. Lastly, the network contains a good number of representative features for class "3" test samples and was able to classify 4 out of 5 correctly as given in Figure 17e.

2. Document Categorization with Enhanced Interpretability

In light of constructing an interpretable deep network, an L_1/L_2 -NCSAE pre-trained deep network with 10 hidden neurons in the first AE layer, 5 hidden neurons in the second AE, and 10 output neurons (one for each category) in the softmax layer was constructed. It was trained on Reuters data, and compared with that pre-trained using DpAE. The interpretation of the encoding layer of the first AE is provided by listing words associated with 10 strongest weights, and the interpretation of the encoding layer of the second AE is portrayed as images characterized by both the magnitude and sign of the weights. Compared to the AE with weights of both signs shown in Figure 18a, Figure 18b allows for much better insight into the categorization of the topics.

Topic *earn* in the output weight matrix resonates with the 5th hidden neuron most, lesser with the 3rd, and somewhat with the 4th. This resonance can happen only when the 5th hidden neuron reacts to input by words of columns 1 and 4,

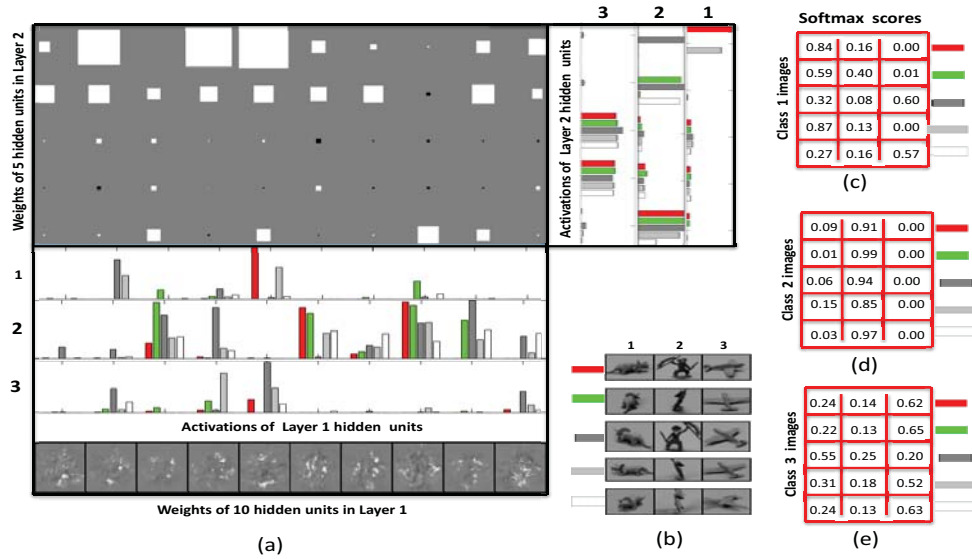


FIGURE 17: The weights were trained using two stacked L_1/L_2 -NCSAEs. RFs learned from the reduced NORB dataset are plotted as images at the bottom part of (a). The intensity of each pixel is proportional to the magnitude of the weight connected to that pixel in the input image with negative value indicating black, positive values white, and the value 0 corresponding to gray. The biases are not shown. The activations of first layer hidden units for the NORB objects presented in (b) are depicted on the bar chart on top of the RFs. The weights of the second layer AE are plotted as a diagram at the topmost part of (a). Each row of the plot corresponds to the weight of each hidden unit of second AE and each column for weight of every hidden unit of the first layer AE. The magnitude of the weight corresponds to the area of each square; white indicates positive, grey indicates zero, and black negative sign. The activations of second layer hidden units are shown as bar chart in the right-hand side of the second layer weight diagram. Each column shows the activations of each hidden unit for five color-coded examples of the same object. The outputs of Softmax layer for color-coded test objects with class labels (c) "fourlegged animals" tagged as class 1, (d) "human figures" as class 2, and (e) "airplanes" as class 3 [1].

and in addition, to a lesser degree, when the 3rd hidden neuron reacts to input by words of the 3rd column of words. So, in tandem, the dominant columns 1, 4 and then also 3 are sets of words that trigger the category *earn*.

Analysis of the term words for the topic *acq* leads to a similar conclusion. This topic also resonates with the two dominant hidden neurons 5 and 3 and somewhat also with neuron 2. These neurons 5 and 3 are driven again by the columns of words 1,4, and 3. The difference between the categories is now that to a lesser degree, the category *acq* is influenced by the 6th column of words. An interesting point is in contribution of the 3rd column of words. The column connects only to the 4th hidden neuron but weights from this neuron in the output layer are smaller and hence less significant than for any other of the five neurons (or rows) of the output weight matrix. Hence this column is of least relevance in the topical categorization.

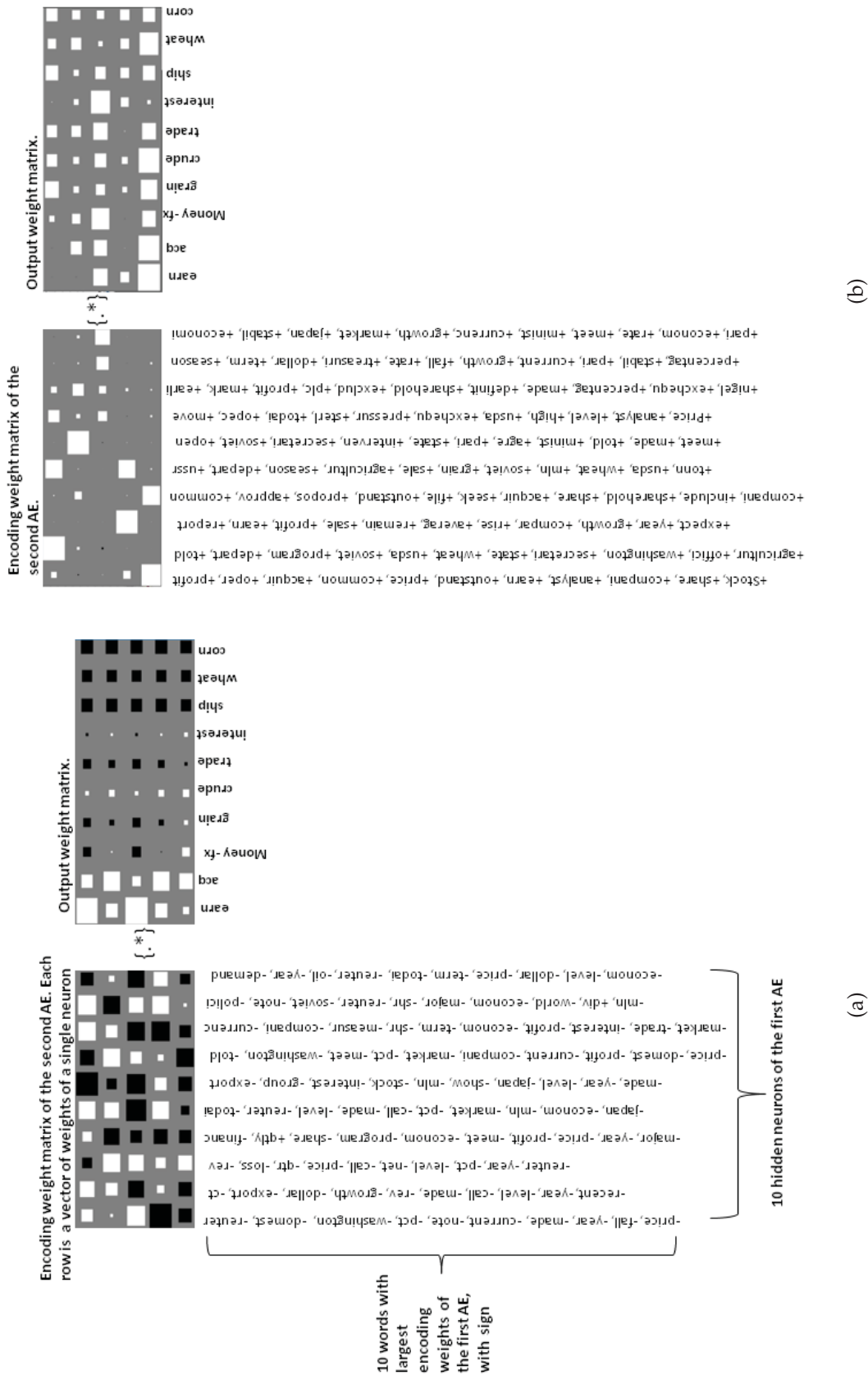


FIGURE 18: Deep network trained on Reuters-21578 data using (a) DpAE, (b) L_1/L_2 -NCsAE. The area of each square is proportional to the weight's magnitude. The range of weights are scaled to $[-1,1]$ and mapped to the grayscale map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color [1].

3. Performance Evaluation on Supervised Learning

In this set of experiments, a deep network was constructed using two stacked L_1/L_2 -NCSAE and a softmax layer for classification to test if the enhanced ability of the network to shatter data into parts and lead to improved classification. Eventually, the entire deep network is fine-tuned to improve the accuracy of the classification. In this set of experiments, the performance of pre-training a deep network with L_1/L_2 -NCSAE is compared with those pre-trained with recent AE architectures. The MNIST and NORB data sets were utilized, and every run of the experiments is repeated ten times and averaged to combat the effect of random initialization. The classification accuracy of the deep network pre-trained with NNSAE [61], DpAE [72], DAE [75], AAE [77], NCAE, and L_1/L_2 -NCSAE using MNIST and NORB data respectively are detailed in Table 4. The network architectures are 784-196-20-10 and 1024-200-20-5 for MNIST and NORB dataset respectively. It is remarked that for training of AAE with two layers of 196 hidden units in the encoder, decoder, discriminator, and other hyperparameters tuned as described in [77], the accuracy was 83.67%. The AAE reported in Table 4 used encoder, decoder, and discriminator each with two layers of 1000 hidden units and trained for 1000 epochs. The classification accuracy and speed of convergence are the figures of merit used to benchmark L_1/L_2 -NCSAE with other AEs.

It is observed from the result that L_1/L_2 -NCSAE-based deep network gives an improved accuracy before fine-tuning compared to methods such as NNSAE, NCAE, DpAE, and NCAE. However, the performance in terms of classification accuracy after fine-tuning is very competitive. In fact, it can be inferred from the p-value of the experiments conducted on MNIST and NORB in Table 4 that there is no significant difference in the accuracy after fine-tuning between NCAE and L_1/L_2 -NCSAE even though most of the weights in L_1/L_2 -NCSAE are nonnegativity constrained. Therefore it is remarked that even though the interpretability of

TABLE 1: Classification accuracy on MNIST and NORB dataset [1]

Dataset		Before fine-tuning		After fine-tuning		
		Mean (\pm SD)	<i>p</i> -value	Mean (\pm SD)	<i>p</i> -value	# Epochs
MNIST	SAE	0.735 \pm 0.015	<0.001	0.977 \pm 0.0007	<0.001	400
	NCAE	0.844 (\pm 0.0085)	0.0018	0.974 (\pm 0.0012)	0.812	126
	NNSAE	0.702 (\pm 0.027)	<0.0001	0.970 (\pm 0.001)	<0.0001	400
	L_1/L_2 -NCSAE	0.847 (\pm 0.0077)	-	0.974 (\pm 0.0087)	-	84
	DAE (50% input dropout)	0.551 (\pm 0.011)	<0.0001	0.972 (\pm 0.0021)	0.034	400
	DpAE (50% hidden dropout)	0.172 (\pm 0.0021)	<0.0001	0.964 (\pm 0.0017)	<0.0001	400
	AAE	-	-	0.912 (\pm 0.0016)	<0.0001	1000
NORB	SAE	0.562 \pm 0.0245	<0.0001	0.814 \pm 0.0099	0.041	400
	NCAE	0.696 (\pm 0.021)	0.406	0.817 (\pm 0.0095)	0.001	305
	NNSAE	0.208 (\pm 0.025)	<0.0001	0.738 (\pm 0.012)	<0.001	400
	L_1/L_2 -NCSAE	0.695 (\pm 0.0084)	-	0.812 (\pm 0.0001)	-	196
	DAE (50% input dropout)	0.461 (\pm 0.0019)	<0.0001	0.807 (\pm 0.0015)	0.0103	400
	DpAE (50% hidden dropout)	0.491 (\pm 0.0013)	<0.0001	0.815 (\pm 0.0038)	<0.0001	400
	AAE	-	-	0.791 (\pm 0.041)	<0.0001	1000

the deep network has been fostered by constraining most of the weights to be non-negative and sparse, nothing significant has been lost in terms of accuracy. In addition, network trained with L_1/L_2 -NCSAE was also observed to converge faster than its counterparts. On the other hand, NNSAE also has nonnegative weights but with deterioration in accuracy, which is more conspicuous especially before the fine-tuning stage. The improved accuracy before fine-tuning in L_1/L_2 -NCSAE based network can be traced to its ability to decompose data more into distinguishable parts. Although the performance of L_1/L_2 -NCSAE after fine-tuning is similar to those of DAE and NCAE but better than NNSAE, DpAE, and AAE, L_1/L_2 -NCSAE constrains most of the weights to be nonnegative and sparse to foster transparency than for other AEs. However, DpAE and NCAE performed

slightly more accurate than L_1/L_2 -NCSAE on NORB after network fine-tuning.

C. Conclusion

This chapter addresses the concept and properties of special regularization of DL AE that takes advantage of non-negative encodings and at the same time of special regularization. It has been shown that by using both L_1 and L_2 to penalize the negative weights, most of them are forced to be nonnegative and sparse, and hence the network interpretability is enhanced. In fact, it is also observed that most of the weights in the Softmax layer become nonnegative and sparse. In sum, it has been observed that encouraging nonnegativity in NCAE-based deep architecture forces the layers to learn part-based representation of their input and leads to a comparable classification accuracy before fine-tuning the entire deep network and not-so-significant accuracy deterioration after fine-tuning. It has also been shown on select examples that concurrent L_1 and L_2 regularization improve the network interpretability. The performance of the proposed method was compared in terms of sparsity, reconstruction error, and classification accuracy with the conventional SAE and NCAE, and we utilized MNIST handwritten digits, Reuters documents, and the NORB dataset to illustrate the proposed concepts.

CHAPTER III

UNSUPERVISED NONREDUNDANT FEATURE EXTRACTION

Feature extraction through constrained learning of RFs offers special promise and has recently become one of the important tenets of DL [78]. In deep autoencoding, AE performs unsupervised learning to detect feature hierarchies which shatter the data and generate features. In this process each added AE layer adds one more abstract representation of inputs, in effect producing a cascade of encodings. Further, the architectural complexity and the excessive number of weights and units are often built in into the DL data representation by design and are deliberate [79], [78], [80], [81].

Observations from previous studies indicate that over-sized DL architectures typically result in largely over-determined (or over-complete) systems [82, 83]. Unless special ad-hoc precautions are implemented to achieve acceptable accuracy such as de-noising, contraction of layers, and elimination of initially assumed and inherent redundancies, the generalization abilities of DL resulting methods suffer [82]. The resulting architectures may therefore not be the most computationally efficient due to their size, computational complexity, and due to their over-representation of data. Such suboptimal architectures sometime may learn local, or isolated features, especially with shallower architectures which have a limited capacity to combine inputs/features.

To address the over-representation of data mapping into the DL multilayer architectures, layers can be trained under specific and well-defined sets of constraints that remove a number of training limitations. The subsequent subsections discuss the purpose and specific techniques of defining the constraints. The con-

straints criteria refer to the RFs (or simply filters) originally introduced in [71].

A. Filtering Redundancy Elimination in Autoencoder-based Deep Networks

In general, as a result of learning for minimum error of reconstruction or classification, the RFs manifest themselves as quasi basis functions that are usually sparse and of the same dimensionality as the input layer (or, in general, of dimensionality of the preceding processing layer). It will be shown that a large number of filters are similar or even duplicative, thus creating unnecessary amount of filtering redundancy. The same features are therefore extracted multiple times [84]. In this section, the aim is to reduce the number of redundant RFs first in the offline setting, and in order to fully leverage on the proposed heuristic, their clustering is automated with the goal of reaching a predetermined number of filter clusters. It is remarked that the proposed approach requires final retraining of the AE to lower its reconstruction error. Further, we focus on filters that sort classes of images.

The most closely related approach to the described is the dropout technique - one of the recently introduced heuristics to sparsify the AEs to prevent overfitting. The key idea is to randomly drop units and their connections from the neural network during training [58]. Other related work is the k -sparse AE [85–87], which aims at reducing the number of filters by sorting the hidden units' activations and retaining k largest units, while setting the rest to zero. The algorithm that will be discussed in this section, on the other hand, aims at detecting the number of filter clusters according to the differentiation of filters based on their distances and the reconstruction error value. Hence, the proposed method leads to comparable reconstruction error with a reduced number of filters.

The proposed method is also applicable to AEs that extract non-negative latent features discussed in Chapters 1 and 2. In addition to using conventional AE, this concept will also be demonstrated using two AE architectures with non-

negative weights that produce additive decompositions layer-wise [1, 51, 61]. The novel way of obtaining a near-optimal number of filters for sparse representation of data will also be discussed.

1. Filter Clustering and Reduction

Training of SAEs indicate that resulting RFs are duplicative thus leading to a number of redundant RFs with the same feature to be extracted by two or more filters. In this section, two heuristics that aim at agglomerative static and dynamic clustering of filters as well as reduction of the hidden layer size are discussed. These criteria need to enforce:

1. **Static reduction of the number of redundant filters** while preserving and/or enhancing their sparsity. This reduction is analyzed and performed initially in the offline mode and for the filters that have been pre-trained. The essence of the pre-training is to avail the filters enough iterations.
2. **Dynamic reduction and reconciliation of filters** that is undertaken concurrently with their unsupervised learning. Such clustering of filters aims at automatically choosing a good similarity threshold for RFs for a given AE architecture and detecting the number of distinct filter clusters.

The above two criteria aim at producing a reduced set of filters. The term reduction refers here to computing their smallest number according to a specific chosen measure.

a. Static Reduction of the Number of Redundant Filters The objective in this section is removal of identical or very similar filters for the purpose of eliminating duplicative retrieval of features. Suitable similarity measures are needed to express the intra-filter distances between vectors \mathbf{V}_s that define the filters. Assuming $(\mathbf{V}_s, b_s), s=1, \dots, n'$, are weight vectors and biases, each \mathbf{V}_s corresponds to the s -th row of

the weight matrix $\mathbf{W}^{(1)}$ as in Fig. 2

$$\mathbf{V}_s = \mathbf{W}_s^{(1)} \quad s = 1, \dots, n' \quad (20)$$

Algorithm 1: Offline Feature Extraction (OFE)

```

1 function OFE (data, Autoencoder)
2   Input : data, Autoencoder
3   initialize:  $\{n'_0, pretrain\_iter, max\_iter\}$ 
4   initialize  $\mathbf{W}$  and  $\mathbf{b}$ :  $\mathbf{W}^{(init)}, \mathbf{b}^{(init)}$ 
5   set:  $\tau$ 
6    $\mathbf{W}, \mathbf{b} = \text{TRAIN\_AE}(\text{data}, n'_0, \mathbf{W}^{(init)}, \mathbf{b}^{(init)}, pretrain\_iter)$ 
7    $\mathcal{L} = \text{COMPUTE\_LOSS}(\text{data}, \mathbf{W}, \mathbf{b})$ 
8    $n'_{new}, \mathbf{W}^{(aggl)}, \mathbf{b}^{(aggl)} = \text{AGGLOMCLUSTERING}(\mathbf{W}, \mathbf{b}, \tau)$ 
9    $\mathbf{W}^{(new)}, \mathbf{b}^{(new)} = \text{Finetune\_AE}(\text{data}, n'_{new}, \mathbf{W}^{(aggl)}, \mathbf{b}^{(aggl)}, max\_iter)$ 
10 function AgglomClustering ( $\mathbf{W}, \mathbf{b}, \tau$ );
11   Input :  $\{\mathbf{W}, \mathbf{b}, \tau\}$ 
12   Output : Distinct filters and their biases  $\Rightarrow n'_{new}, \mathbf{W}^{(new)}, \mathbf{b}^{(new)}$ 

```

The set of n' vectors \mathbf{V}_s exhibits mutual distances as follows:

$$d_{sr} = \|\mathbf{V}_s - \mathbf{V}_r\| \quad s, r = 1, \dots, n'; \quad s \neq r \quad (21)$$

A number of similarity testing/clustering algorithms can be applied for elimination (and possibly merger) of originally developed redundant filters. Based on a comparative review, a clustering approach from [88, 89] has been adapted and reformulated for this purpose as shown in Algorithm 1.

Starting with each original filter as a potential cluster, agglomerative clustering is performed by merging the two most similar clusters C_i and C_j as long as the average similarity between their constituent filters is above a chosen cluster

similarity threshold denoted as τ [90,91]. The similarity threshold is an hyperparameter that has to be set in order to achieve optimal performance. The set of n' vectors \mathbf{V}_s exhibits mutual similarities as follows:

$$\text{similarity}(C_i, C_j) = \frac{\sum_{\mathbf{V}_s \in C_i, \mathbf{V}_r \in C_j} \text{NGC}(\mathbf{V}_s, \mathbf{V}_r)}{|C_i| \times |C_j|} > \tau \quad (22)$$

$$i, j = 1, \dots, n'_{\text{new}}; \quad s = 1, \dots, |C_i|, \quad r = 1, \dots, |C_j| \quad \text{and} \quad s \neq r$$

where the similarity between two filters is measured by Normalized Greyscale Correlation (NGC):

$$\text{NGC}(\mathbf{V}_s, \mathbf{V}_r) = \frac{\sum_{k=1}^n (p_k - \bar{p}_k)(q_k - \bar{q}_k)}{\sqrt{\sum_{k=1}^n (p_k - \bar{p}_k)^2 \sum_{k=1}^n (q_k - \bar{q}_k)^2}} \quad (23)$$

where $p_k \in \mathbf{V}_s$ and $q_k \in \mathbf{V}_r$; $\bar{p}_k = \sum_{k=1}^n p_k / n$ and $\bar{q}_k = \sum_{k=1}^n q_k / n$ and n is the size of the filter vector \mathbf{V}_s .

This clustering scheme guarantees that similar filters are grouped, and that the clusters stay compact [90]. That is, the constituent filters in each cluster stay as close as possible to one another or simply put, the intra-cluster distances are as small as possible.

b. Dynamic Reduction and Reconciliation of Filters The objective here is to discover c clusters in the set of n' original filters, where $c < n'$. Further evaluation is performed on how to replace filters that are within a single cluster with this cluster prototype or centroid's representation. The algorithm has been designed to heuristically set τ to develop novel representation that expresses aggregated properties of input data across the training set. The Algorithm 2 also tries to reduce the number of hidden units based on clustering of the input weights. It trains an AE with initial number of hidden units for *pretrain_iter* epochs, which must be chosen large enough to enable formation of duplicative filters. Number of hidden units is initialized as large as practical and τ is also initialized to a small value in order to fully activate the filter reduction heuristic. The reconstruction error is evaluated using the weights and biases obtained from the pre-training stage. The

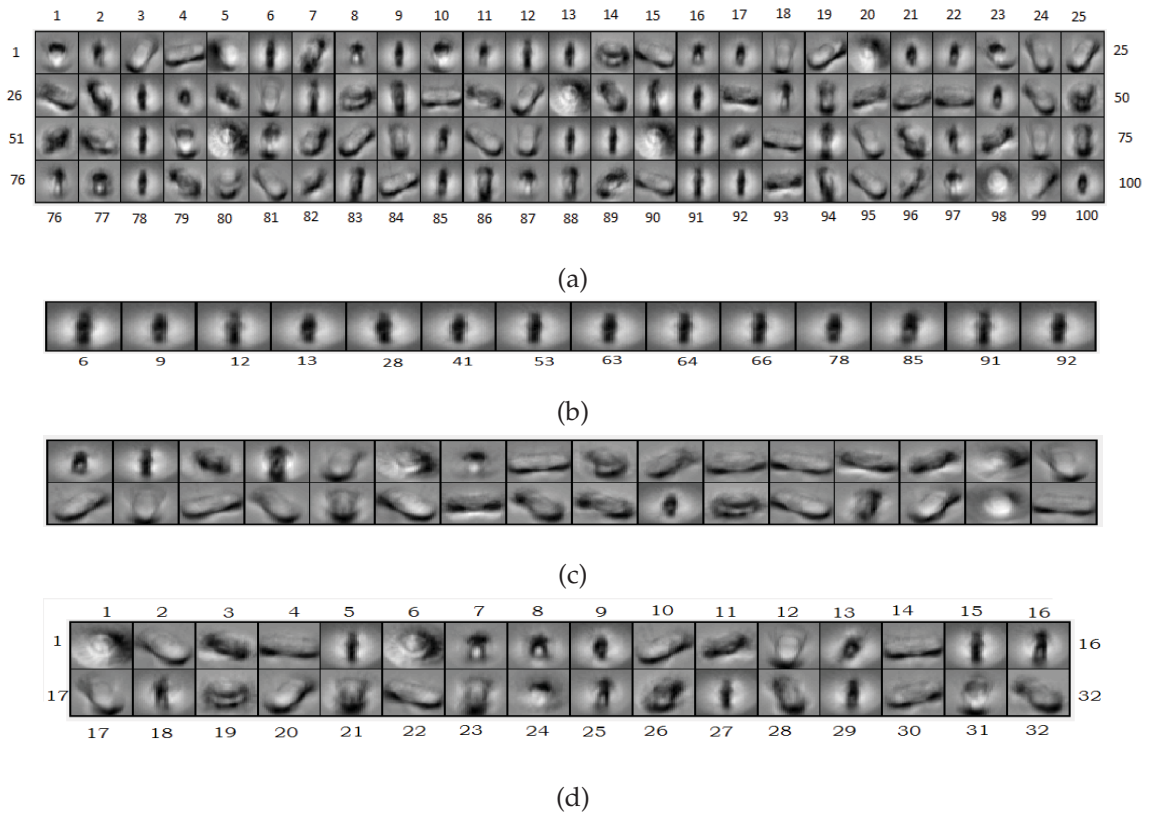


FIGURE 19: Filters learned from NORB data set using SAE with (a) 100 original filters (b) 14 examples of very similar filters with their corresponding indices at the bottom (c) 32 filters resulting from aggllo-SAE with $\tau=16$, and (d) 32 original filters. Black pixels indicate negative, and white pixels indicate positive weights [2].

weight vectors are then clustered and filters with similarity above the threshold τ are collapsed.

The resulting filters are fine-tuned for *scan_iter* epochs such that reconstruction error in (4) does not increase in comparison with the reconstruction error computed using previous τ . If the reconstruction error reduces, τ value is incremented by $\Delta\tau$ and the process is repeated. As mentioned above, the filter reduction becomes less active as τ increases, which in turn implies that reconstruction error should decrease due to increasing number of hidden units. At certain τ value and beyond, reconstruction error stops reducing which might be as a result of duplicative filters formation. Once no significant decrease in the reconstruction error associated with previous and current τ values is observed or the error starts to increase, Algorithm 2 stops and outputs the optimal number of filters and the resulting τ . The filters are then fine-tuned for *max_iter* epoch to ensure good reconstruction. It is worth mentioning that Algorithms 1 and 2 are alternative approaches and can be used independently. Also note that Algorithm 2 is an extended version of Algorithm 1 where acceptable RF cluster similarity threshold is set automatically to eliminate most redundant RFs, whereas in Algorithm 1 the hyperparameter τ is selected using trial and error. In this chapter, SAE, NCAE, and L_1/L_2 -NCAE denote training using the approach in Algorithms 1 and 2 as aggro-SAE, aggro-NCAE and aggro- L_1/L_2 -NCAE, respectively.

B. Experimental Setup

This section discusses the performance of the proposed method in redundant filter reduction and reported for three benchmark image data sets: MNIST, NORB normalized-uniform dataset and the Yale face dataset. The input to the first layer of the AE is the vector of pixel intensities. The training parameters in Table 2 have been found experimentally for hyperparameter tuning of each of the

algorithm with various parameters that best minimize the cost function. In Algorithm 1, *pretrain_iter* and *max_iter* were both experimentally set to 200. Both AEs with RF clustering and those without clustering have very similar training time. Although clustering overhead is introduced by the proposed algorithm.

It is worth mentioning that the computational complexity of the proposed algorithm does not grow with increasing number of data points. Also, the method proposed does not require a fully trained network but a network partially trained for few epochs (*pretrain_iter*). For instance, SAE and aggro-SAE were both trained for 400 epochs. In the case of aggro-SAE, the network was first pre-trained for 200 epochs, then clustering was performed, followed by fine-tuning for another 200 epochs. It must be noted that the last 200 epoch is faster in aggro-network than its counterpart. This compensates for the clustering overhead. In Algorithm 2, hyperparameters *pretrain_iter*, *scan_iter*, and *max_iter* were set to 200, 50, and 150 respectively. It is remarked that this choice of iteration parameters should ensure that both traditional AEs and aggro-based ones were trained with similar training time. Also in experiments using Algorithm 2, $\Delta\tau$ was set to 0.5 and τ initialized to 1.

All experiments were performed on Intel(r) Core(TM) i7-6700 CPU @ 3.40Ghz and a 64GB of RAM running a 64-bit Windows 10 Enterprise edition. The software implementation has been with MATLAB 2015b, and LBFGS in minFunc [92] is used to minimize the objective function. The usage time in seconds is the time elapsed in seconds a fully trained deep network (DN) requires to classify all the test samples.

Algorithm 2: Automatic Feature Extraction (AFE)

```
1 function AFE (data, Autoencoder)
2   Input : data, Autoencoder
3   initialize:  $n'_0, pretrain\_iter, scan\_iter, max\_iter$ 
4   initialize:  $\mathbf{W}^{(init)}, \mathbf{b}^{(init)}, k, \tau, \Delta\tau$ 
5    $\mathbf{W}, \mathbf{b} = \text{TRAIN\_AE}(\text{data}, n'_0, \mathbf{W}^{(init)}, \mathbf{b}^{(init)}, pretrain\_iter)$ 
6    $\mathcal{L} = \text{COMPUTE\_LOSS}(\text{data}, \mathbf{W}, \mathbf{b})$ 
7   while  $\mathcal{L}^{(iter+1)} < \mathcal{L}^{(iter)}$  do {
8      $n'_{(iter+1)}, \mathbf{W}^{(aggl)}, \mathbf{b}^{(aggl)} = \text{AGGLOMCLUSTERING}(\mathbf{W}, \mathbf{b}, \tau)$ 
9      $\mathbf{W}^{(iter+1)}, \mathbf{b}^{(iter+1)} = \text{TRAIN\_AE}(\text{data}, n'_{(iter+1)}, \mathbf{W}^{(aggl)}, \mathbf{b}^{(aggl)},$ 
10     $scan\_iter)$ 
11      $\mathcal{L}^{(iter+1)} = \text{COMPUTE\_LOSS}(\text{data}, \mathbf{W}^{(iter+1)}, \mathbf{b}^{(iter+1)})$ 
12     if  $\mathcal{L}^{(iter+1)} \geq \mathcal{L}^{(iter)}$ :
13       return  $n'_{(iter)}$ 
14        $\tau \leftarrow \tau + \Delta\tau$  }
```

14 $\mathbf{W}, \mathbf{b} = \text{TRAIN_AE}(\text{data}, n'_{(iter)}, \mathbf{W}^{(iter)}, \mathbf{b}^{(iter)}, max_iter)$

TABLE 2

Parameter settings [2]

Parameters	SAE	NCAE	L_1/L_2 -NCAE
Sparsity penalty (β)	3	3	3
Sparsity parameter (p)	0.05	0.05	0.05
Weight decay penalty (α)	3e-3	-	-
Nonnegativity constraint penalty (α_1)	-	-	1e-4
Nonnegativity constraint penalty (α_2)	-	3e-3	3e-3
Maximum No. of Iterations	400	400	400

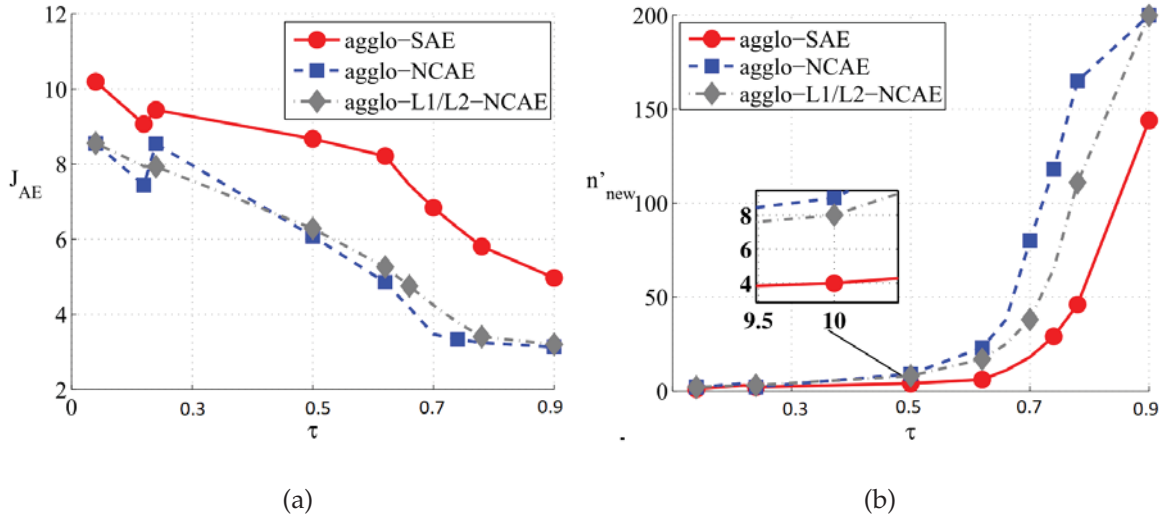


FIGURE 20: Performance of AE on the NORB dataset. (a) Reconstruction error vs. cluster similarity threshold. (b) Number of RFs vs. cluster similarity threshold for 196 initial filters [2].

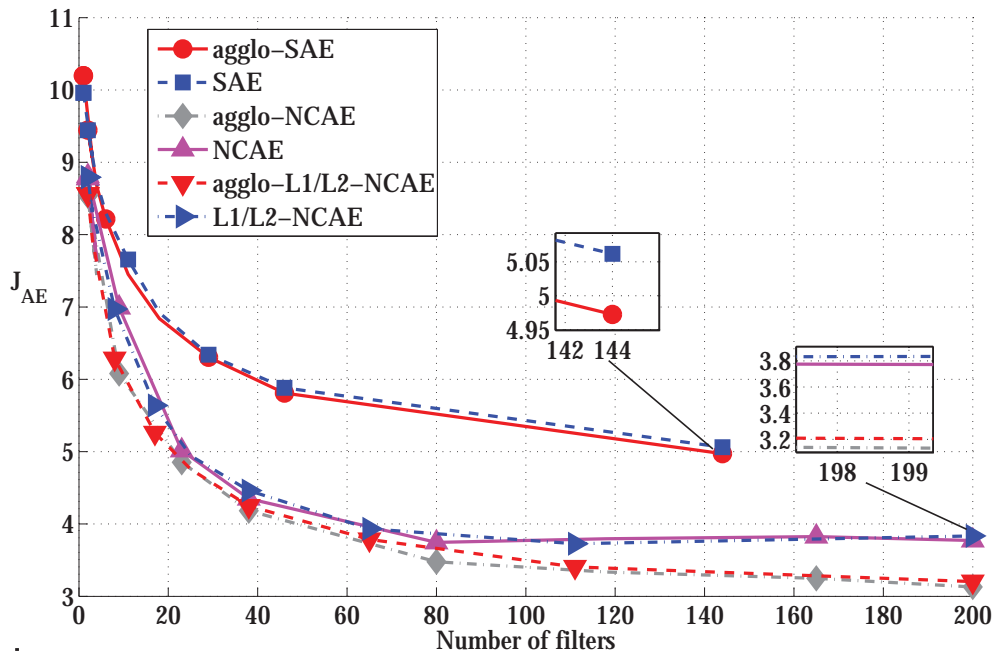


FIGURE 21: Reconstruction error using SAE, NCAE, and L_1/L_2 -NCAE trained with the same number of hidden units from experiment with aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE on NORB data [2].

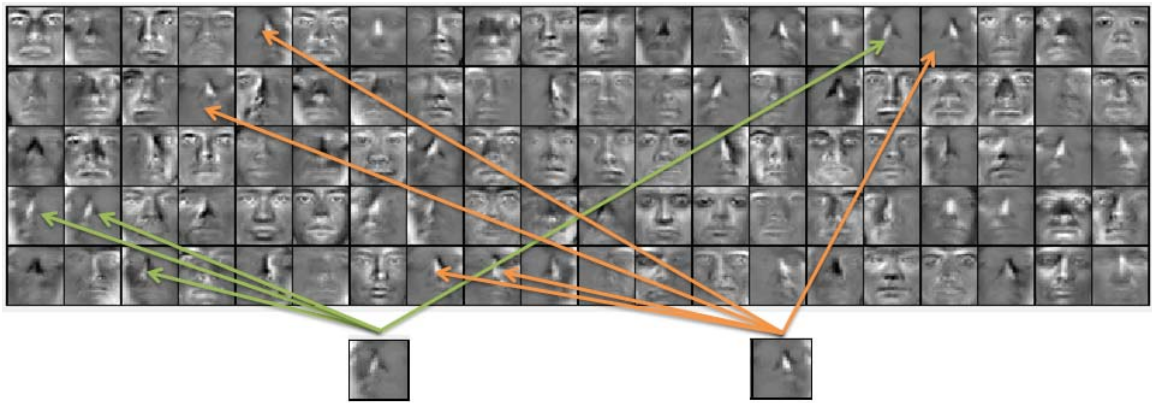


FIGURE 22: 100 receptive fields learned from Yale Face Dataset using SAE with examples of two duplicative RFs [2].

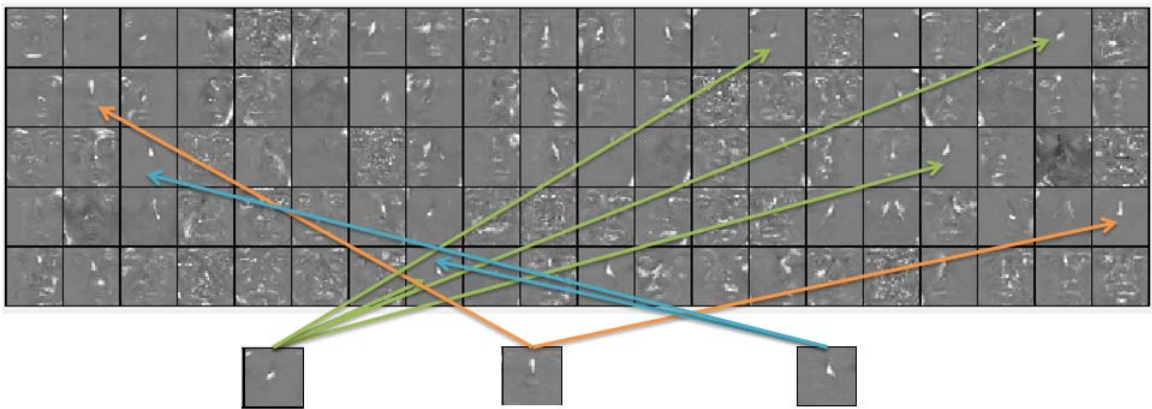


FIGURE 23: 100 receptive fields learned from Yale Face Dataset using NCAE with examples of duplicative RFs [2].

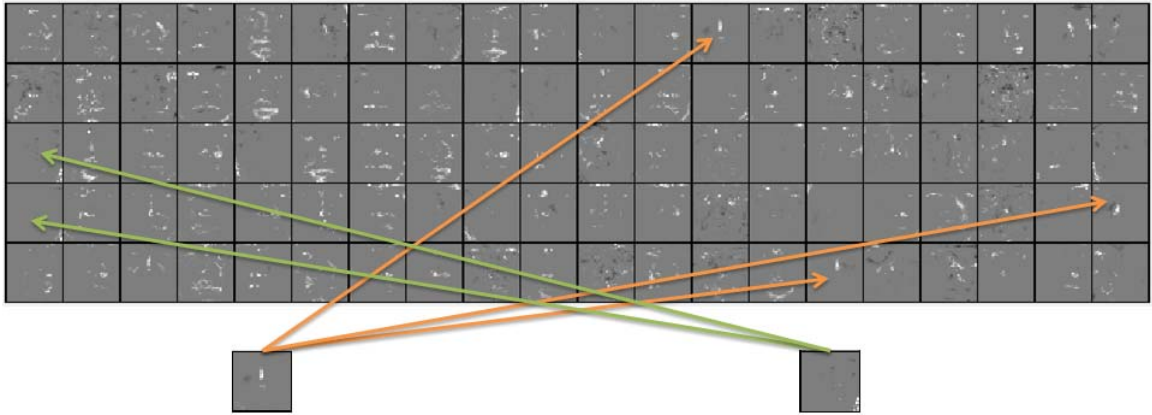
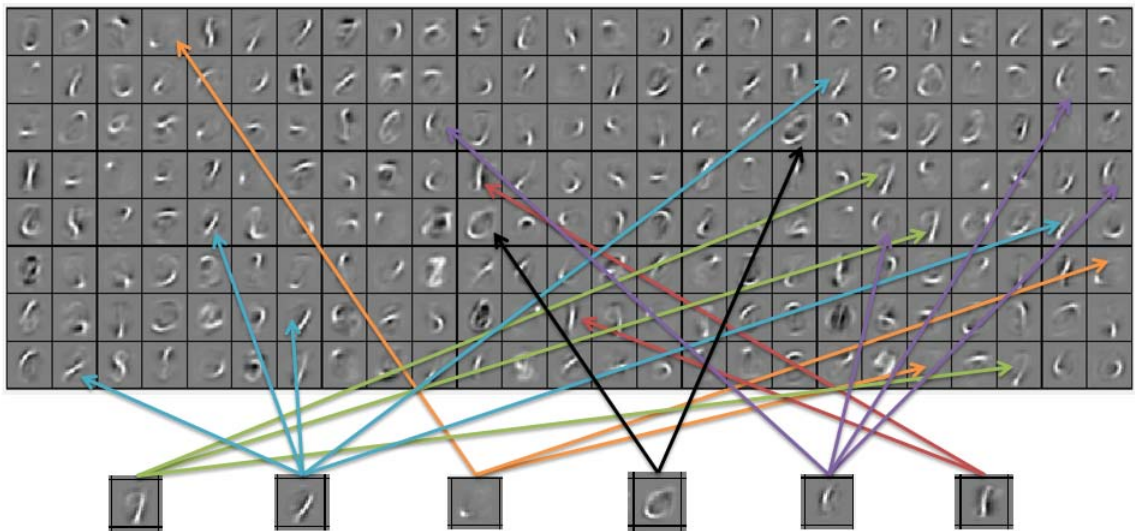
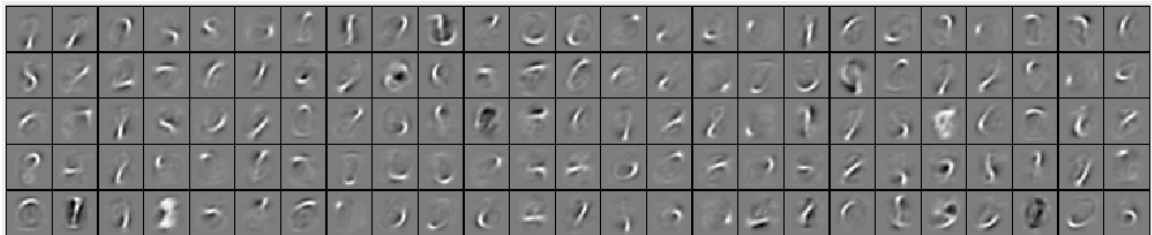


FIGURE 24: 100 receptive fields learned using L_1/L_2 -NCAE from Yale Face Dataset, with examples of duplicative RFs [2].



(a)



(b)

FIGURE 25: Filters learned from MNIST data set using SAE (a) 200 RFs with examples of duplicative filters, (b) 125 RFs for aggro-SAE with $\tau=0.6$ [2]

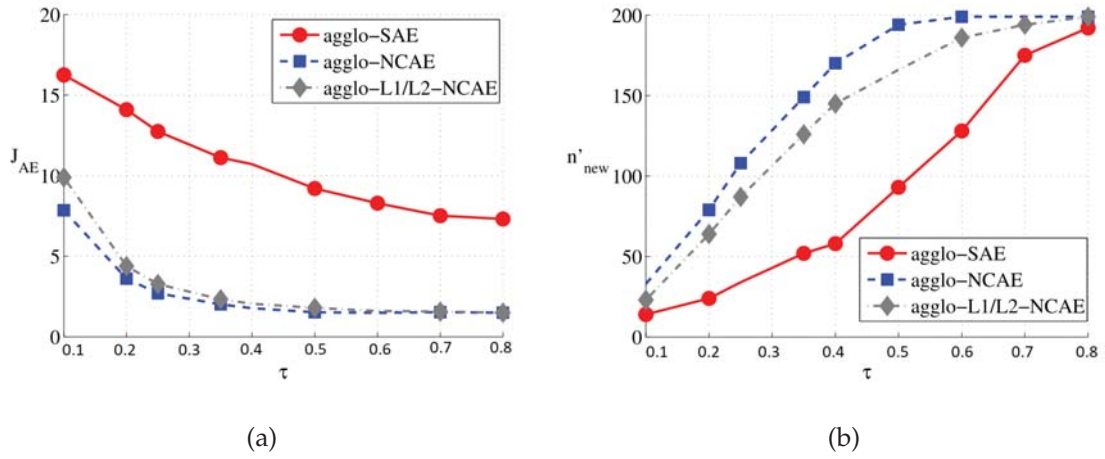


FIGURE 26: Performance of AE on the MNIST dataset vs. cluster similarity threshold (a) Reconstruction error (b) RF size (n'_{new}) using 200 initial filters [2].

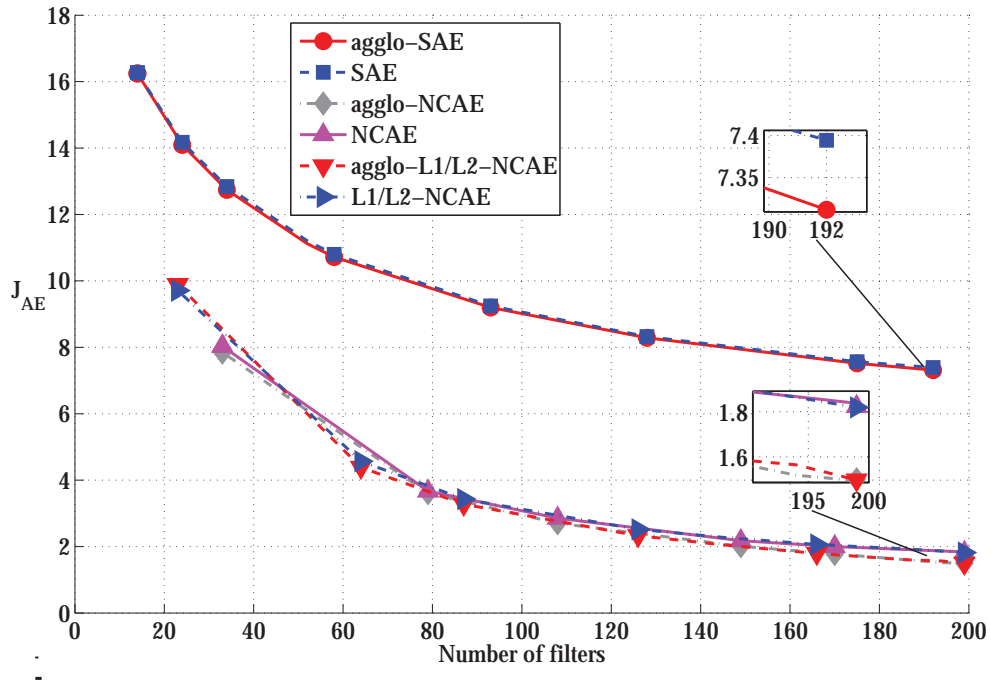


FIGURE 27: Reconstruction error using SAE, NCAE, and L_1/L_2 -NCAE trained with the same number of hidden units from experiment with aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE on MNIST data [2].

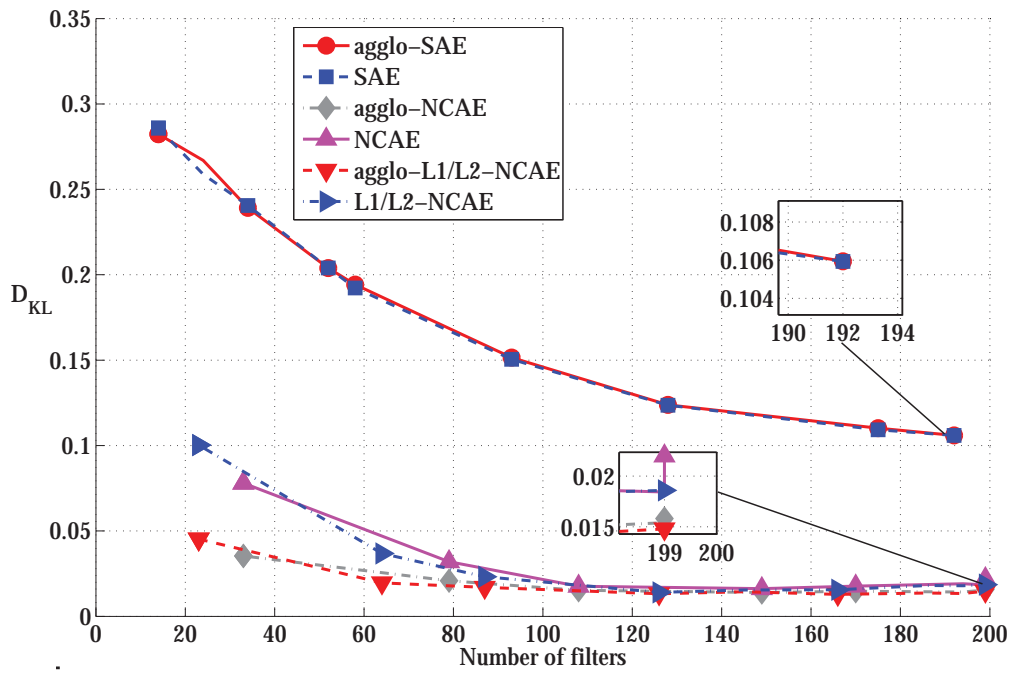


FIGURE 28: KL-Divergence sparsity measure with respect to a desired $p = 0.05$ using SAE, NCAE, and L_1/L_2 -NCAE trained with the same number of hidden units (n'_{new}) from experiment with aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE on MNIST data [2].

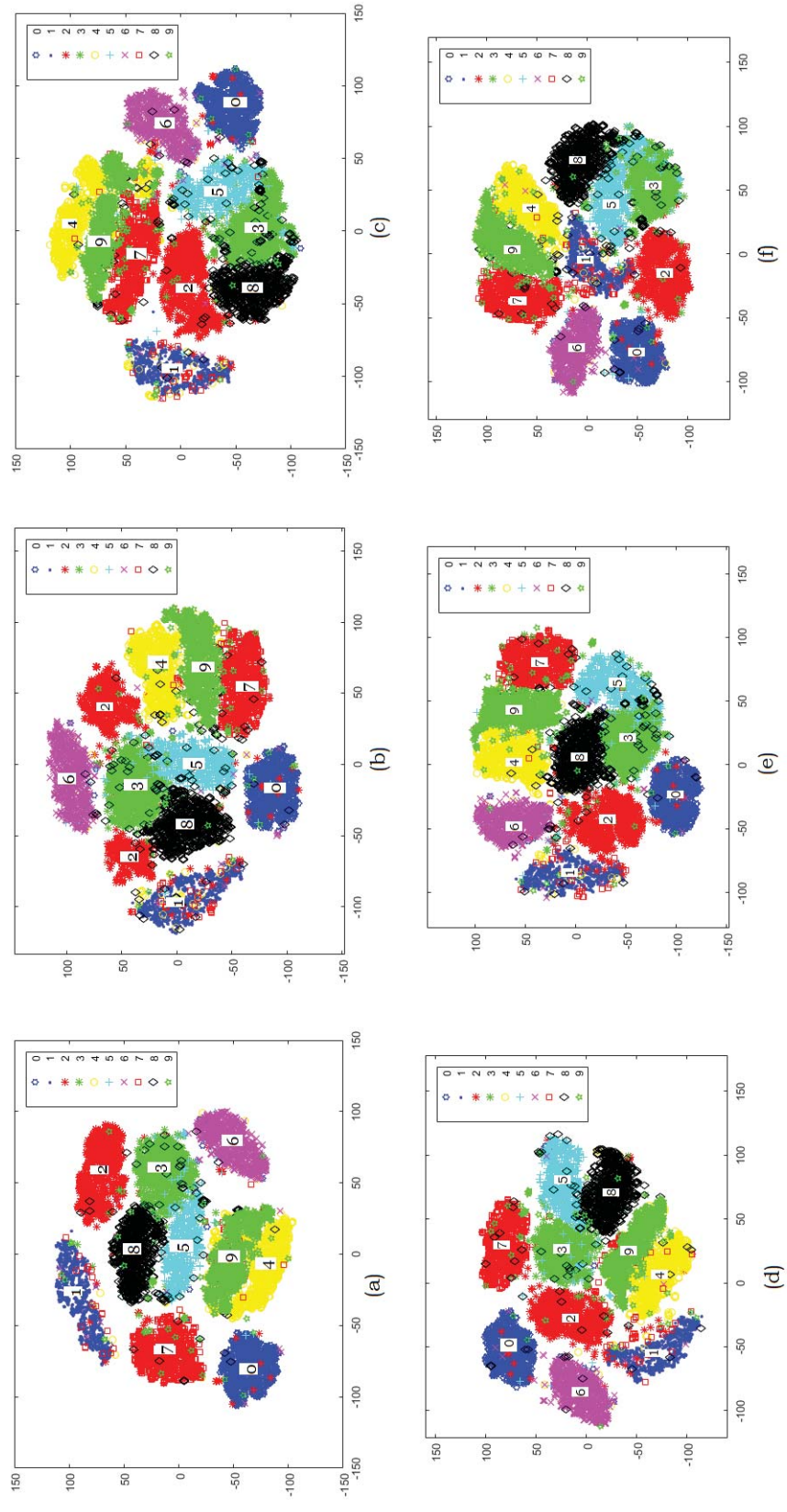


FIGURE 29: t-SNE projection [6] of 200D representations of MNIST handwritten digits using (a) SAE (b) NCAE (c) L_1/L_2 -NCAE, (d) 174D representations using aggllo-SAE (e) 153D representations using aggllo-NCAE, and (f) 172D representations using aggllo- L_1/L_2 -NCAE [2].

a. Unsupervised Feature Reduction via Filter Pruning In the first set of experiments, conventional SAE with 100 hidden units is trained using the NORB normalized-uniform dataset. To foster the claim that filter duplication is probable, the indexed RFs learned from NORB data are shown in Figure 19a. Figure 19b shows select RFs from Figure 19a. The visual inspection indicates that a good number of these filters are very similar. In order to eliminate the redundancy of RFs in Figure 19a, SAE is retrained with the Algorithm 1. Figure 19c shows the 32 RFs learned using aggro-SAE with similarity threshold $\tau = 0.73$. A quick inspection reveals that most duplicative filters in Figure 19a have been grouped in Figure 19c. In order to compare filter duplication and the consequent filtering redundancy with and without the agglomerative clustering approach, conventional SAE was trained with the same number of 32 hidden units. As shown in Figure 19d, a number of RFs resulting from SAE training with 32 hidden units can still be observed even though the network was trained with the same number of hidden units as produced by aggro-SAE heuristic.

Similarly, Nonnegativity Constrained Autoencoder (NCAE) [51] and L_1/L_2 -NCAE [1] with 200 hidden units each were trained using the NORB data. As observed in Figure 20a, both aggro-NCAE and aggro- L_1/L_2 -NCAE achieve better reconstruction accuracy than aggro-SAE. It can also be observed in Figure 20b that more distinct filters are produced as τ is varied in aggro-NCAE and aggro- L_1/L_2 -NCAE than in aggro-SAE. This indicates that imposing nonnegativity constraint on the network’s weights helps in learning an elevated number of distinct features, while also improving the reconstruction. Again, SAE, NCAE, and L_1/L_2 -NCAE were trained with the same number of hidden units that resulted from the experiment in Figure 20b. The error curves shown in Figure 21 reveal that networks trained with aggro-SAE, aggro-NCAE and aggro- L_1/L_2 -NCAE also yield a lower reconstruction error in comparison with those trained without agglomerative-clustering-

based approach. Figure 21 is averaged over ten experiments to show the statistical significance of improved reconstruction capability of aggro-AEs over the traditional AEs considered (SAE, NCAE and L1/L2-NCAE).

The second set of experiments is to show redundant feature extraction using AEs trained on Yale Face Dataset [93]. The database contains 11 images of 15 individuals, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink. The original size of each image is 320×243 with 256 gray levels per pixel. Each image is resized to 32×32 to reduce the computational time and normalize between 0 and 1 [94]. SAE, NCAE and L_1/L_2 -NCAE each with 100 hidden units were trained on this dataset and the RFs learned are shown in Figures 22, 23 and 24, respectively. It was observed that some of the filters are very similar and will thereby extract similar features. Again, it is observed that both types of nonnegativity-constrained AEs also learned duplicative filters that produce redundant filtering. This experiment shows that the tendency to learn redundant features is not specific to conventional SAE only but to a variety AE architectures.

In the third experiment, SAE, NCAE and L_1/L_2 -NCAE were trained using the MNIST digits dataset. A careful look at the 200 RFs of trained SAE in Figure 25a shows that many of the filters are duplicative and redundant, and thereby resulting in redundant over-representation of data and increased computational complexity. By visual inspection it can be observed that a good number of filters can be considered as redundant and eliminated with no significant loss of reconstruction accuracy. By deploying Algorithm 1, 75 redundant filters were eliminated and the resulting distinct agglomerative filters are shown in Figure 25b. In the case of aggro-SAE, it was observed that the reconstruction error does not decrease significantly beyond the similarity threshold of 0.7 in Figure 26a, which corresponds to approximately 172 filters in Figure 26b. No significant reduction in the reconstruc-

tion error was observed for aggro-NCAE and aggro- L_1/L_2 -NCAE for values of τ greater than 0.5 corresponding to 194 and 166 distinct filters, respectively. It is remarked that decreasing τ will decrease the number of RFs and hence increase the error. By implication, increasing τ will lead to the increase of duplicative RFs. The two bottom curves on Figure 26a also indicate that nonnegativity constraints yield better reconstruction quality on this data set. It is again shown in Figure 26b that imposition of nonnegativity constraints on the network’s weights results in networks with larger n'_{new} for a large range of similarity τ compared to SAE.

Similarly, SAE, NCAE, and L_1/L_2 -NCAE were trained with the same n'_{new} that resulted from the experiment in Figure 26b using the MNIST handwritten digits data. It can be observed in Figure 27 that the proposed agglomerative-based heuristic improves the reconstruction accuracy for all three AEs. Also, the sparsity has increased in aggro-NCAE and aggro- L_1/L_2 -NCAE, respectively, compared to their counterparts NCAE and L_1/L_2 -NCAE as shown in Figure 28.

However, no obvious sparsity improvement was noticed in the case of aggro-SAE. The proposed heuristic has been also evaluated based on the distribution of data in high level feature space. In this regard, t-distributed stochastic neighbor embedding (t-SNE) was used to project the high-level representations (that is, the hidden activations) of SAE, NCAE, L_1/L_2 -NCAE, aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE to 2D space [6]. The 2D projections of high-level (200D) representations of MNIST handwritten digits test set corresponding to hidden activities of SAE, NCAE, and L_1/L_2 -NCAE are respectively visualized in Figures 29a,b, and c. For the purpose of comparison, 174, 153, and 172D representations of MNIST handwritten digits for aggro-SAE, aggro-NCAE, and aggro- L_1/L_2 -NCAE are respectively depicted in Figures 29d,e, and f.

The t-SNE projections in Figure 29 no visible/obvious deterioration in the manifolds of the 174D representations of aggro-SAE compared to 200-D represen-

tations of SAE. This is an indication that more than 25 hidden activations are redundant and eliminating them does not deteriorate the manifold shown in Figure 29a and d. Similarly as shown in Figure 29b and e, approximately 50 hidden activations can be eliminated without overlapping the manifolds that enclose the 200-D representations of MNIST digits using NCAE. In a similar manner, elimination of more than 25 duplicative hidden activities has not adversely affected the manifolds of the digits' projections as shown in Figure 29f of aggro- L_1/L_2 -NCAE in comparison with Figure 29c of L_1/L_2 -NCAE.

b. Effect of Redundant Feature Pruning on Supervised Learning In the last set of experiments, a DN was tested using two stacked AEs and a softmax classification output to evaluate the effect of filter reduction on classification. The network was fine tuned by backpropagation algorithm to improve the classification accuracy. MNIST dataset was utilized for the first set of experiments, and it is noted that the filter reduction algorithm is only implemented on the first layer of the DN for experiments reported in Table 3. However, as will be shown below, this concept can also be applied to all layers of the deep architectures. For convenience of presentation on Tables 3-8, Λ_b and Λ_a are respectively defined in (24) and (25); $n'_{1,new}$ and $n'_{2,new}$ are the number of agglomerative filters in the first and second layer, respectively.

The classification accuracy and testing time are reported in Table 3 for the three benchmark AEs. Reported are averaged results of 10 independent trials and related mean values and standard deviation (SD) of 6 simulation series. It can be observed from the results that removing redundant filters in aggro-SAE-pretrained network does not deteriorate the classification performance of the DN. $\tau = 0.7$ was chosen based on the result in Fig. 26a, which shows that the reconstruction error does not significantly decrease beyond $\tau = 0.7$. This indicates that no matter how many filters are added beyond n'_{new} corresponding to $\tau = 0.7$, the performance

TABLE 3

Classification performance on MNIST dataset using initial network configuration 784-200-20-10 [2].

Architecture		Mean (\pm SD)	p-value	Usage time (\pm SD) (ms)
SAE ($n'_1=200$)	Λ_b	0.860 (± 0.007)	0.1517	59.9 (± 3.51)
	Λ_a	0.977 (± 0.0011)	0.8268	60.6 (± 2.51)
agglo-SAE ($\tau=0.7$)	$n'_{1,new}$	174 (± 2)	-	-
	Λ_b	0.855 (± 0.005)	-	59.2 (± 1.93)
	Λ_a	0.978 ($\pm 5.8e-4$)	-	54.0 (± 2.09)
NCAE ($n'_1=200$)	Λ_b	0.849 (± 0.008)	0.028	59.2 (± 2.14)
	Λ_a	0.975 ($\pm 8.94e-4$)	0.0645	55 (± 2.27)
agglo-NCAE ($\tau=0.35$)	$n'_{1,new}$	153 (± 3)	-	-
	Λ_b	0.852 (± 0.0061)	-	50.9 (± 2.44)
	Λ_a	0.974 (± 0.0016)	-	48.6 (± 1.29)
L_1/L_2 -NCAE ($n'_1 = 200$)	Λ_b	0.845 (± 0.205)	0.108	62.1 (± 4.13)
	Λ_a	0.974 (± 0.0014)	0.7374	56.3 (± 1.90)
agglo- L_1/L_2 -NCAE ($\tau=0.5$)	$n'_{1,new}$	172 (± 3)	-	-
	Λ_b	0.842 (± 0.0067)	-	58.2 (± 1.84)
	Λ_a	0.974 ($\pm 8.9e-4$)	-	53.1 (± 1.12)

$$\Lambda_b = \frac{\text{Number of correctly classified test cases before supervised fine-tuning}}{\text{Total number of test cases}} \quad (24)$$

$$\Lambda_a = \frac{\text{Number of correctly classified test cases after supervised fine-tuning}}{\text{Total number of test cases}} \quad (25)$$

where Λ_b is commonly referred to as accuracy before fine-tuning and Λ_a as accuracy after fine-tuning.

of the DN is not likely to improve. One of the direct aftermaths of filter reduction is the drastic reduction in training time and the usage time. It is apparent from the results that more than 6ms have been saved on the usage time, which is significant when dealing with very large datasets encountered in real world scenarios. However, for aggro-NCAE-pretrained DN, a slight improvement of $\approx 1\%$ in classification accuracy was observed before fine-tuning and this can be due to the features extracted in aggro-NCAE that are more sparse than those obtained using conventional NCAE. As can be inferred from the p-values, there is a significant difference in the performance before fine-tuning and no significant improvement observed after fine-tuning. Again $\tau=0.35$ for NCAE was chosen in connection with Fig. 26a.

The observations from experiments with DN pretrained using aggro- L_1/L_2 -NCAE also reveal that usage time is reduced and no significant difference was noticed in the accuracy before and after finetuning compared to the L_1/L_2 -NCAE-pretrained network. It is worthy of note that the hidden size of the second layer used in the first set of experiments was chosen to be 20 for computational reasons and negligible reduction of RFs number was noticed for most of τ values considered. However, the size of the hidden layers of all the AEs were chosen to be as

TABLE 4

Classification performance on MNIST dataset using initial network configuration 784-1000-20-10 [2].

Architecture		Mean (\pm SD)	p-value	Usage time (\pm SD) (ms)
SAE ($n'_1=1000, n'_2=20$)	Λ_b	0.8063 (\pm 0.0124)	0.0004	281.8 (\pm 25.0)
	Λ_a	0.9782 (\pm 0.000704)	0.0030	257.1 (\pm 34.2)
agglo-SAE ($\tau=0.7$)	$n'_{1,new}$	889 (\pm 6)	-	-
	$n'_{2,new}$	17 (\pm 1)	-	-
	Λ_b	0.8514 (\pm 0.0106)	-	147.9 (\pm 2.5)
	Λ_a	0.9819 (\pm 9.2e-4)	-	146.7 (\pm 0.35)
DpAE (20% dropout)	Λ_b	0.7314 (\pm 0.0363)	0.0017	209.7 (\pm 30.9)
	Λ_a	0.9585 (\pm 0.0107)	0.0090	203.4 (\pm 38.5)

large as the hidden layer size of the first AE in subsequent experiments and near-optimal number of RFs were obtained using the proposed heuristic.

In the next large-scale experiment reported in Tables 4-7, the effect of removing redundant filters on network performance is demonstrated using the MNIST dataset. Every run of the experiments is repeated five times and averaged for statistical significance. The classification performance of the DN pre-trained with DpAE [72] was used as a benchmark. In Table 4, the number of hidden units in the first and second layers were set to 1000 and 20 respectively. It can be observed from the results that more than 100 redundant filters have been removed in the first layer and 2 in second layer. An improved classification accuracy in aggro-SAE was also observed after eliminating the redundancy compared to SAE and DpAE counterparts. In addition, the usage times of SAE and DpAE-trained networks have been respectively reduced more than 40% and 28% in aggro-SAE. Similar trends were observed in the classification accuracy after fine-tuning when the number of hidden units of the first and second layer were respectively set to 1000 and 200 as detailed in Table 5.

In an attempt to investigate the effect of the proposed algorithm on overfitting, the number of hidden units of the first and second layer were increased to 500 and 500, respectively as in Table 6 and 1000-1000 as in Table 7. One of the key observations in Tables 6-7 is that as the AE's hidden layer size grows, the performance of SAE deteriorates. It was also observed that the use of dropout did not help in improving the performance. It is remarked that during the experiments, 20%, 30%, and 50% dropout fractions were tested and the one with the best output performance was reported. However, in these experiments, aggro-SAE outperforms both SAE and DpAE before and after fine-tuning. The performance of aggro-SAE could be traced to its ability to eliminate filtering redundancy and it is worth mentioning that such elimination might help in reducing overfitting. Table 8

TABLE 5

Classification performance on MNIST dataset using initial network configuration 784-1000-200-10 [2].

Architecture		Mean (\pm SD)	p-value	Usage time (\pm SD) (ms)
SAE ($n'_1=1000, n'_2=200$)	Λ_b	0.9604 (\pm 0.0014)	0.0151	324.1(\pm 39)
	Λ_a	0.9667 (\pm 0.0015)	0.0172	290.2(\pm 16)
agglo-SAE ($\tau=0.7$)	$n'_{1,new}$	874 (\pm 13)	-	-
	$n'_{2,new}$	95(\pm 2)	-	-
	Λ_b	0.9422 (\pm 0.0039)	-	162.3 (\pm 43)
	Λ_a	0.9826 (\pm 3.9e-4)	-	165.0 (\pm 35.6)
DpAE (20% dropout)	Λ_b	0.9612 (\pm 0.0024)	0.3743	193.4 (\pm 37.8)
	Λ_a	0.9768 (\pm 0.0013)	0.3738	209.2 (\pm 41.4)

TABLE 6

Classification performance on MNIST dataset using initial network configuration 784-500-500-10 [2].

Architecture		Mean (\pm SD)	p-value	Usage time (\pm SD) (ms)
SAE ($n'_1=500, n'_2=500$)	Λ_b	0.9595 (± 0.0012)	<0.0001	240.6 (± 10)
	Λ_a	0.9642 (± 0.0012)	0.0025	223.6 (± 16)
agglo-SAE ($\tau=0.7$)	$n'_{1,new}$	483 (± 3)	-	-
	$n'_{2,new}$	392 (± 3)	-	-
	Λ_b	0.9708 (± 0.0013)	-	220.8 (± 9.8)
	Λ_a	0.9785 (± 0.0046)	-	211.8 (± 20)
DpAE (20% dropout)	Λ_b	0.9640 (± 0.0074)	0.1777	253.7 (± 12.2)
	Λ_a	0.9704 (± 0.0112)	0.1777	227.1 (± 19.8)

reports the classification results on the small NORB data set and demonstrates that the network with agglo-SAE outperforms the two other networks before and after fine-tuning. It can be noticed that more than 50% of the filters in first layer and 75% in the second layer were redundant and removed with improved classification performance. Lastly, it was also observed that the results obtained in most of the experiments using Algorithm 2 are close to the results obtained using Algorithm 1. The conclusion drawn from this observation is that Algorithm 2 implements faster search of hyperparameter τ , hence it is more practical in the training phase.

TABLE 7

Classification performance on MNIST dataset using initial network configuration 784-1000-1000-10 [2].

Architecture		Mean (\pm SD)	p-value	Usage time (\pm SD) (ms)
SAE ($n'_1=1000, n'_2=1000$)	Λ_b	0.9647 (\pm 0.0012)	0.0004	480.6 (\pm 18.1)
	Λ_a	0.9689 (\pm 9.6e-4)	<0.0001	476.1 (\pm 57.9)
agglo-SAE ($\tau=0.7$)	$n'_{1,new}$	844 (\pm 6)	-	-
	$n'_{2,new}$	272 (\pm 10)	-	-
	Λ_b	0.9730 (\pm 8.9e-4)	-	298.8 (\pm 8.2)
	Λ_a	0.9812 (\pm 9.0e-4)	-	275.9 (\pm 17)
DpAE (20% dropout)	Λ_b	0.9464 (\pm 0.0513)	0.3124	403.4 (\pm 59.5)
	Λ_a	0.8483 (\pm 0.2701)	0.3336	378.2 (\pm 57.7)

TABLE 8

Classification performance on NORB dataset [2].

Architecture		Mean (\pm SD)	p-value	Usage time (\pm SD) (ms)
SAE ($n'_1=500, n'_2=500$)	Λ_b	0.8085 (\pm 0.0065)	0.3516	385.7 (\pm 38.8)
	Λ_a	0.8143 (\pm 0.0065)	0.0313	387.8 (\pm 34.6)
agglo-SAE ($\tau=0.65$)	$n'_{1,new}$	242 (\pm 14)	-	-
	$n'_{2,new}$	123 (\pm 8)	-	-
	Λ_b	0.8129 (\pm 0.0102)	-	159.1 (\pm 27.0)
	Λ_a	0.8303 (\pm 0.0087)	-	170.7 (\pm 31.8)
DpAE (20% dropout)	Λ_b	0.5042 (\pm 0.1377)	0.0062	443.0 (\pm 66.1)
	Λ_a	0.5090 (\pm 0.1208)	<0.001	475.2 (\pm 57.3)

C. Conclusion

This chapter proposes new techniques for data representation in the context of DL for stacked AEs by leveraging on the ability to agglomerate regularized sparse RFs and also by enhancing the feature generation process at the output layer via controlled feature compression. The performance of the proposed method in terms of decomposing data into parts and non-redundant feature extraction was compared for the conventional SAE with constrained AEs of type DpAE, NCAE, and L_1/L_2 -NCAE. The proposed technique uses agglomerative clustering which starts off by allocating each original filter to a separate cluster and merges two most similar clusters as long as the average *similarity* between their members is above a set threshold τ . This concept is illustrated using the NORB normalized-uniform object data set, MNIST handwritten digits data and Yale Face Dataset.

The results show that a large number of originally generated RFs are overlapping across these three data sets, creating unnecessary amount of filtering redundancy. By using the proposed methods, such redundancy can be controlled, eliminated and AEs are enabled to extract fewer and more distinctive features.

CHAPTER IV

REDUNDANCY-BASED FILTER PRUNING IN DEEP CONVOLUTIONAL NEURAL NETWORKS

Due to large-scale labeled data and efficient architectural design, deep neural networks (DNNs) in recent years have shown superior performance in many supervised learning tasks ranging from computer vision [73,95–98] to speech recognition [99–101] and natural language processing [102,103]. Over the past few years, the general trend has been that DNNs have grown deeper and larger, amounting to huge number of final parameters. Their flexibility and performance usually come with high computational and memory demands both during training and inference. A myriad of recent studies have shown that over-sized deep learning models typically result in largely over-determined (or over-complete) systems [2, 23, 104–107]. For instance, this over-complete representation is evidently pronounced in features learned by popular deep neural network architectures such as AlexNet [73] as emphasized by [24, 108].

The resulting oversized architectures may therefore be less computationally efficient due to their size, over-parameterization and their high inference cost. To account for the scale, diversity and the difficulty of data these models learn from, the architectural complexity and the excessive number of weights are often deliberately built in into the design of DNN models [79, 105]. These over-sized models have expensive training and inference costs especially for applications with constrained computational and power resources such as web services, mobile and embedded devices. In addition to good accuracy, many resource-limited applications would greatly benefit from lower inference cost [109, 110]. While the de-

mand for high computational inefficiency in the training phase has been alleviated with general-purpose computing engines otherwise known as Graphics Processing Units (GPUs) to accelerate computations but powerful GPUs are unavailable in hand-held devices.

Additionally, flexibility of DNN models may hinder their scalability and practicality, and may result in extracting highly redundant parameters with risk of over-fitting [111]. A symptom of learning replicated or similar features is that two or more processing units extract very similar and correlated information. From an information value standpoint, similar or shifted versions of features do not add extra information to the feature hierarchy, and therefore should be possibly suppressed. In other words, the activation of one unit should not be predictable based on the activations of other units of the same layer. However, enforcing dissimilarity of features in traditional way can be generally involved requiring computation of intractable joint probability table and batch statistics. To address this problem of over-representation, layers of deep and/or wide architectures have to be examined for possible redundancy to remove this limitation after training.

Knowing the level of redundancy in models is useful mainly for two reasons. First, information about the level of redundancy in models can be used for feature diversification in order to optimize their performance [?,24,111,112]. This is addressed in Chapter VI. Secondly, it can be used to build accurate inference-cost-efficient models via pruning for resource-limited applications to benefit greatly from lower inference cost and high accuracy [109,110]. This is important in practice because optimal architectures are unknown. However, pruning enables smaller model to preserve knowledge from a larger model. Since learning a complex function directly with a small suboptimal architecture might result in low accuracy, it is therefore necessary to first learn a task with larger architecture with many parameters followed by pruning redundant and less important features [113]. In

particular, model compression particularly via pruning is important for transferring deep learning models to resource-limited portable devices.

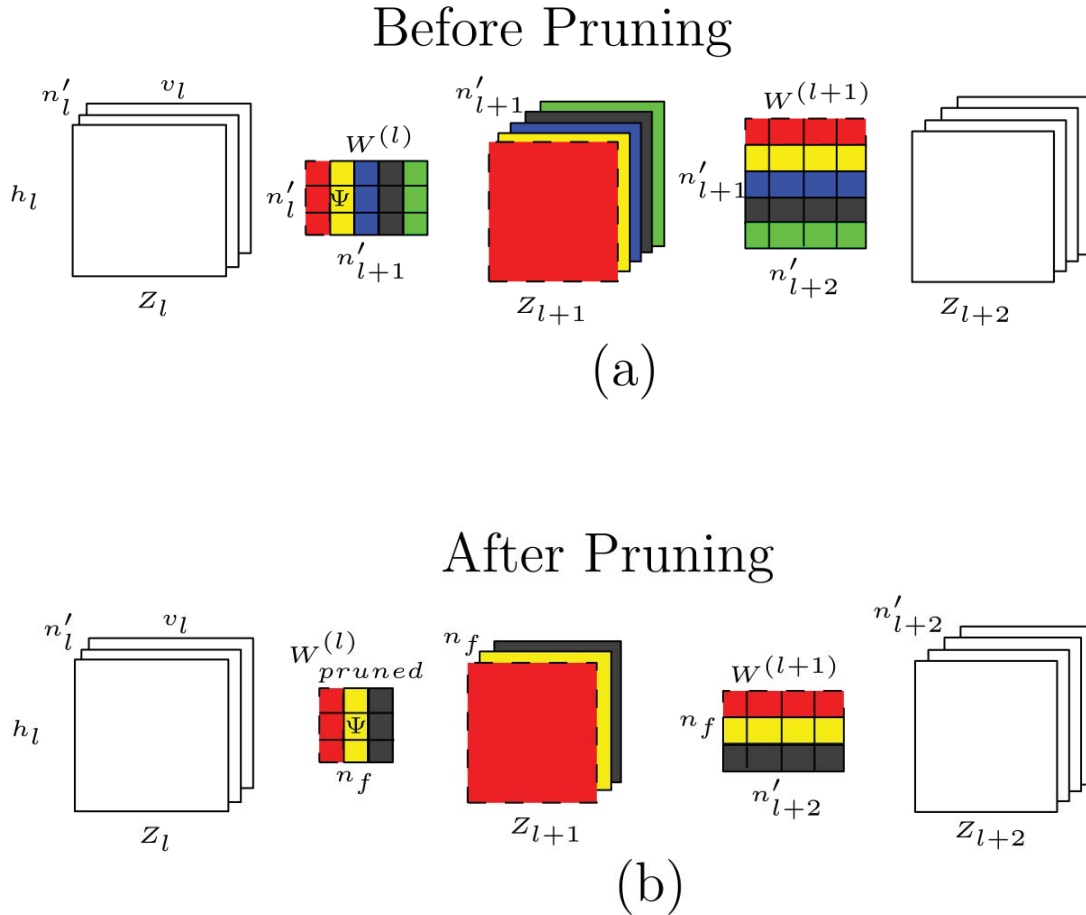


FIGURE 30: Pruning schema of l^{th} layer. (a) Assume filters red, blue, and green in the first, third, and fifth columns of $\mathbf{W}^{(l)}$, respectively, are very similar and are located in the same cluster (b) If filter red is sampled as the representative of the cluster, filters blue and green are redundant and their corresponding feature maps in Z_{l+1} and related weights in the next layer (third and fifth rows of $\mathbf{W}^{(l+1)}$) are all pruned [3].

Storage and computational cost reductions via model pruning techniques have a long history [114–119]. For instance, Optimal Brain Damage [114] and Op-

timal Brain Surgeon [115] use second-order derivative information of the loss function to prune redundant network parameters. Other related work include but is not limited to [113] which prunes based on particle filtering, [120] uses FFT to avoid overhead due to convolution operation, and [121] uses depth multiplier method to scale down the number of filters in each convolutional layer.

Feature redundancy has also been explored to construct a low rank basis of features that are rank-1 in the spatial domain. However, this method involves additional cumbersome optimization procedures. As demonstrated in [104], a fraction of the parameters is sufficient to reconstruct the entire network by simply training on low-rank decompositions of the weight matrices. HashedNets use a hash function to randomly group weights into hash buckets, so that all weights within the same hash bucket share a single parameter value for pruning purposes [122]. Redundant features have also been localized and pruned using simple thresholding mechanism [123].

Instead of localizing the redundant neurons in a fully-connected network, [116] compresses a trained model by identifying a subset of diverse neurons. Redundant feature maps are removed from a well trained network using particle filtering to select the best combination from a number of randomly generated masks [113]. With the assumptions that features are co-dependent within each layer, [124] groups features in hierarchical order. Driven by feature map redundancy, [125] factorizes a layer into 3×3 and 1×1 combinations and prunes redundant feature maps. Closely related to the proposed work, [109] sorts and prunes filters based on the sum of their absolute weights and [123] prunes weights with magnitude below a set threshold.

More recently, [126] trains another neural network as pruning agent which takes filter weights of the model to be pruned as input and outputs binary decisions to remove or keep filters. Using the concept of tensor factorization and re-

construction, [127] eliminates redundancy by pruning the feature maps instead of filter weights. The computational complexity of convolutional network has been reduced by filter-group convolution with tiny accuracy loss while mostly preserving diversity in feature representation. Network reduction problem has also been formulated as a binary integer optimization with a closed-form solution based on final response importance [128].

A. Convolutional Feature Clustering and Pruning

Typically, DNNs consist of input, output, and many intermediate processing layers. By letting the number of channels, height and width of input to the l^{th} layer be denoted as n'_l , h_l , and v_l , respectively. A layer (convolutional or fully-connected) in the network transforms input $Z_l \in \mathbb{R}^p$ into output $Z_{l+1} \in \mathbb{R}^q$, where Z_{l+1} serves as the input in layer $l + 1$. For convolutional neural network (CNN), p and q are given as $n'_l \times h_l \times v_l$ and $n'_{l+1} \times h_{l+1} \times v_{l+1}$, respectively. Whereas for a fully-connected network (FCN), p and q denote $n'_l h_l v_l \times 1$ and $n'_{l+1} \times 1$, respectively. A convolutional layer convolves Z_l with n'_{l+1} 3D filters $\chi \in \mathbb{R}^{n'_l \times k \times k}$, resulting in n'_{l+1} output feature maps (Z_{l+1}). Each 3D filter consists of n'_l 2D kernels $\zeta \in k \times k$. Unrolling and combining all features (or filters) in a single matrix results in kernel matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{m \times n'_{l+1}}$ where $m = k^2 n'_l$. In FCN, however, a layer operation involves only vector-matrix multiplication with kernel matrix $\mathbf{W} \in \mathbb{R}^{m \times n'_{l+1}}$, where $m = n'_l h_l v_l$. Additionally, $\mathbf{w}_i^{(l)}$, $i=1, \dots, n'_l$, denotes i^{th} feature in layer l , each $\mathbf{w}_i^{(l)} \in \mathbb{R}^m$ corresponds to the i -th column of the kernel matrix $\mathbf{W}^{(l)} = [\mathbf{w}_1^{(l)}, \dots, \mathbf{w}_{n'_l}^{(l)}] \in \mathbb{R}^{m \times n'_{l+1}}$.

In this section, two heuristics that aim at agglomerating and pruning convolutional features are introduced. The objective here is to discover n_f features that are representative of n'_l original features using agglomerative hierarchical clustering approach. Achieving effective clustering of features involves choosing suitable

similarity measures to express the inter-feature distances between features w_i that connect the feature map Z_{l-1} to feature maps of layer l . This technique is similar to that described for autoencoder in Chapter III. It starts by putting each feature vector \mathbf{w}_i in a separate potential clusters. Agglomerative clustering then merges the two most similar clusters C_a and C_b as long as the average similarity between their constituent feature vectors is above τ . The pair of clusters C_a and C_b exhibits average mutual similarities as follows:

$$\begin{aligned} \overline{SIM}_C(C_a, C_b) &= \frac{\sum_{\phi_i \in C_a, \phi_j \in C_b} SIM_C(\phi_i, \phi_j)}{|C_a| \times |C_b|} > \tau \\ a, b &= 1, \dots, n'_l; \quad a \neq b; \quad i = 1, \dots, |C_a|; \\ j &= 1, \dots, |C_b|; \quad \text{and} \quad i \neq j \end{aligned} \quad (26)$$

where $\phi_i = w_i / \sqrt{\|w_i\|^2}$, $SIM_C(\phi_1, \phi_2) = \frac{\langle \phi_1, \phi_2 \rangle}{\|\phi_1\| \|\phi_2\|}$ is the cosine similarity between two features and $\langle \phi_1, \phi_2 \rangle$ is the inner product of arbitrary feature vectors ϕ_1 and ϕ_2 , and τ is a set threshold.

It is remarked that the above similarity definition uses the graph-based-group-average technique, which defines cluster proximity/similarity as the average of pairwise similarities (that is, the average length of edges of the graph) of all pairs of features from different clusters. In this work, other similarity definitions such as the single-link and complete-link were also experimented with. Single-link approach defines cluster similarity as the proximity between the two closest feature vectors that are in different clusters. On the other hand, complete-link assumes that cluster proximity is the proximity between the two farthest feature vectors of different clusters. Group average proximity definition empirically yielded better performance compared to the other two definitions and thus, we report experimental results using average proximity approach.

1. Method A: Pruning of Redundant Filters

The redundant convolutional feature-based pruning is detailed in Algorithm 3 with the objective of grouping filters that are identical or very similar in weight space. The algorithm also aims at removing filters that are identical or very similar to eliminate duplicative retrieval of feature maps. The detection and removal of redundant filters is generally tractable especially from practical standpoint since the pruning heuristic uses one-shot pruning and retraining mechanism. As highlighted in Algorithm 3, the proposed pruning heuristic assumed a fully-trained model as input and filter grouping is performed at every layer of the model.

For a particular layer of the DNN model as in Figure 30, Algorithm 3 uses Algorithm 4 to group all the filters ϕ_i (columns of the kernel matrix $\mathbf{W}^{(l)}$) into n_f clusters whose average similarity among cluster members is above a set threshold τ while ensuring $n_f \leq n'$. One representative filter is randomly sampled from each of the n_f clusters. The output of Algorithm 4 is the list L_f of indices of clusters' representatives, which is equivalent to a subset of the indices of columns of $\mathbf{W}^{(l)}$. Algorithm 3 uses L_f to subset $\mathbf{W}^{(l)}$ and create a smaller kernel matrix $\mathbf{W}_{pruned}^{(l)}$. After obtaining all the new kernel matrices $\mathbf{W}_{pruned}^{(l)}$, Algorithm 3 constructs a smaller model initialized with $\mathbf{W}^{(l)pruned}$. In general, pruning a large fraction of filters generally results in performance deterioration. In fact, it is observed that some convolutional layers are extremely sensitive to pruning than others and this must be taken into consideration when pruning such layers and/or models. In most cases, restoring the performance after pruning, the pruned model is fine-tuned for prescribed number of epochs.

It must be noted that if filters are grouped into the same cluster because they have cosine similarity above τ , our approach in Algorithm 4 randomly chooses

Algorithm 3: Redundant Filter-based Pruning

```
1 for layer  $l$  in the trained model do
2   get: convolutional filters of  $l^{th}$  layer  $W^{(l)}$ 
3   set:  $\tau$ 
4   Extract: distinct filters in  $W^{(l)}$ 
5    $L_f, n_f = \text{FILTERCLUSTERING}(W^{(l)}, \tau)$ 
6   initialize:  $W_{pruned}^{(l)}$  of the pruned model
7    $k \leftarrow 0$ 
8   for  $i$  in  $L_f$  do
9     copy:  $i^{th}$  column of  $W^{(l)}$  into  $k^{th}$  column of  $W_{pruned}^{(l)}$ 
10     $k \leftarrow k + 1$ 
11  end for
12 end for
13 Construct the pruned model
14 Initialize the weights of  $l^{th}$  layer with  $W_{pruned}^{(l)}$ 
15 set:  $\tau, retrain\_epoch$ 
16 for prescribed number of  $retrain\_epoch$  do
17   fine-tune the pruned model
18 end for
```

one out of them as the cluster representative. Another approach considered in this work uses cluster centroid as the representative. However, it has been observed that the performance of this approach is similar to the random sampling in Algorithm 4. This further suggests that the cluster centroid is very close to all filters in the cluster. In this section, random selection is used in Algorithm 4 to inject some stochasticity in the selection process.

Algorithm 4: Localization and Pruning of Redundant Filters (Method A)

```

1 FILTERCLUSTERING():
2 Input:  $\{\mathbf{W}, \tau\}$ 
3   Scan for: cluster(s) of vectors in  $\mathbf{W}$  with similarity  $> \tau$ 
4   Randomly sample and tag one representative filter from each of the
       $n_f$  clusters as nonredundant
5 Outputs: List of Indices  $L_f$  of distinct filters and  $n_f$  in  $\mathbf{W}$ ;
6   return  $L_f, n_f$ ,

```

2. Method B: Pruning of Random n_f Filters

Here, Algorithm 5 is used to detect the number of n_f distinct filters in kernel matrix $\mathbf{W}^{(l)}$ of a given layer l . It then randomly samples n_f out of n_l' filters to construct the kernel matrix $\mathbf{W}_{pruned}^{(l)}$ of the pruned model. It is worth motivating and mentioning that Algorithms 4 and 5 are alternative approaches used by Algorithm 3.

The main difference is that Algorithm 4 randomly samples one filter out of every cluster of filters and prunes the remaining filters in all n_f clusters. Note that $n_f \leq n_l$, where n_l is the total number of filters in layer l . Algorithm 5 on the other hand uses filter clustering algorithm only to estimate n_f (the number of

Algorithm 5: Estimation and Random Pruning of n_f filters (Method B)

```
1 FILTERCLUSTERING():  
2 Input:  $\{\mathbf{W}, \tau\}$   
3   Scan for: cluster(s) of vectors in  $\mathbf{W}$  with similarity  $> \tau$  to estimate  $n_f$   
4   Randomly sample  $n_f$  filters  
5 Outputs: List of Indices  $L_f$  of randomly sampled filters and  $n_f$  in  $\mathbf{W}$ ;  
6   return  $L_f, n_f$ 
```

distinct filter clusters) and randomly prunes n_f out of n_l filters. In other words, Algorithm 4 localizes and prunes precisely the redundant filters, while Algorithm 5 just estimates how many filters to randomly prune.

B. Experiments

All experiments were performed on Intel(r) Core(TM) i7-6700 CPU @ 3.40Ghz and a 64GB of RAM running a 64-bit Ubuntu 14.04 edition. The software implementation has been in Pytorch library ¹ on two Titan X 12GB GPUs and the filter clustering was implemented in SciPy ecosystem [129]. The agglomeration of filters using hierarchical clustering is practical for very wide and deep networks even though the complexity of the agglomerative clustering algorithm itself is $O((n'_l)^2 \log(n'_l))$. In most network, $n'_l \leq 1000$ and number of layers is often less than 200. For instance, clustering VGG-16 feature vectors empirically takes on the average on our machine 14.1 milliseconds and this is executed only once during training. This amounts to a negligible computational overhead for most deep architectures.

The implementation of proposed filter pruning strategy is similar to that in [109] in the sense that when a particular filter of a convolutional layer is pruned,

¹<http://pytorch.org/>

its corresponding feature map is also pruned and the weights of the pruned feature map in the filter of the next convolutional layer are equally pruned. It must be emphasized that after pruning the feature maps of last convolutional layer, the input to the fully-connected layer has changed and its weight matrix must be pruned accordingly.

In the preliminary experiment, a multilayer perceptron was trained using MNIST digits [130]. Adam optimizer [131] with batch size of 128 was used to train the model for 400 epochs. The number of redundant feature was computed as $n_r = n'_l - n_f$ after the models have been fully trained.

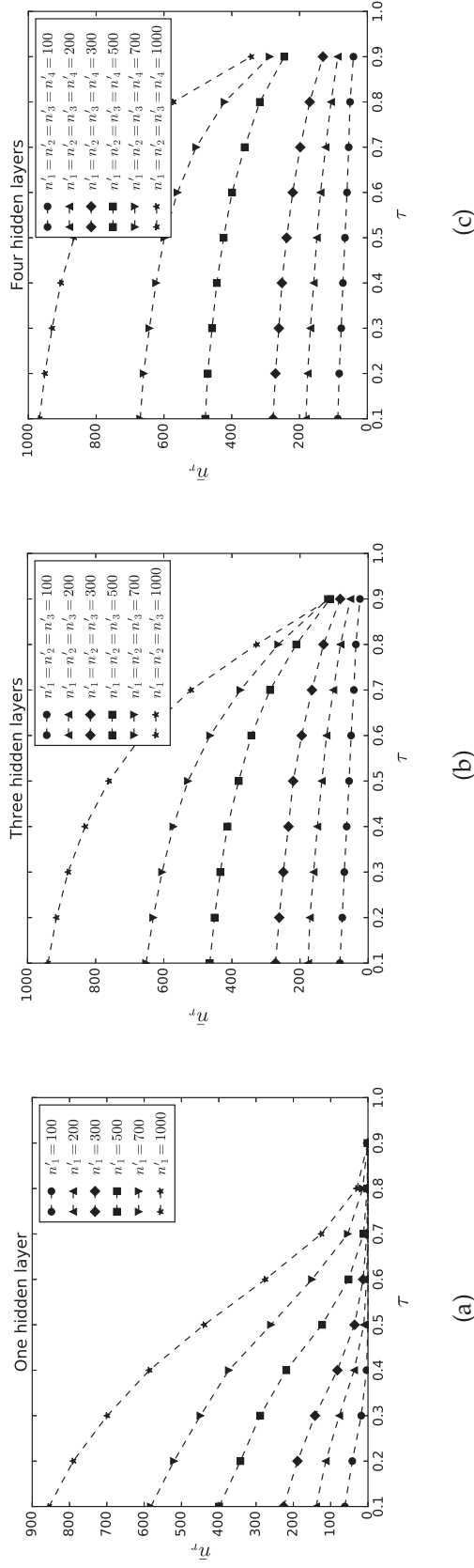


FIGURE 31: Average number of redundant features across all layers (\bar{u}_r) against threshold τ with (a) one (b) two (c) three, and (d) four hidden layers using MNIST dataset. Network width corresponds to the number of hidden units per layer and network depth corresponds to number of hidden layers. Networks with more than one hidden layer have equal number of hidden units in all layers.

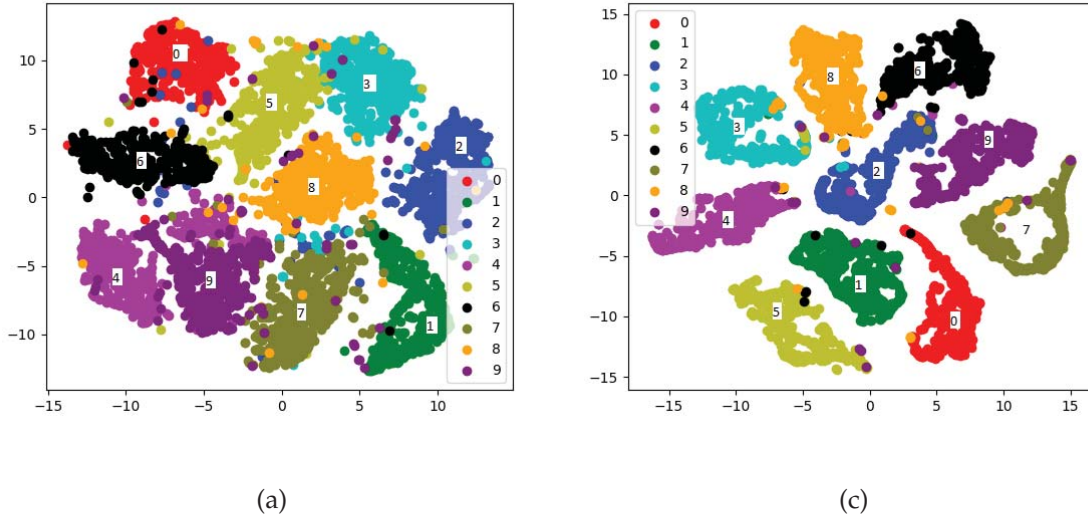


FIGURE 32: t-SNE projection [7] of the activation of last layer of network with (a) one and (b) four hidden layers using 5000 MNIST handwritten digits test samples. All Networks have 1000 hidden units in all layers and all layers use Sigmoid activation function.

Figures 31 a,b,c, and d show the performance of multilayer perceptron with one, two, three, and four hidden layer(s), respectively. The average number of redundant features across all layers of the network is denoted as \bar{n}_r . It can be observed in Figure 31 that both width (number of hidden units per layer) and depth (number of layers in the network) increase \bar{n}_r . As the number of hidden units per layer increases, \bar{n}_r grows almost linearly. Also, the higher the number of hidden layers in a network, the higher the average number of redundant features extracted and the higher the average feature pairwise correlations.

For instance, the network with one hidden layer and 100 hidden units does not have any feature pair with similarity above 0.4. However, as the depth increases (for two or more hidden layers) more feature pairs have similarity above 0.4. This observation is similar for other hidden layer sizes (200, 300, 500, 700, and 1000) and depth. In particular, as can be observed in Figure 31d that many feature

pairs in deep multilayer network (with four hidden layers) are almost perfectly correlated with cosine similarity of 0.9 even with just 100 hidden units per layer. Deep multilayer network was also evaluated based on the distribution of data in high level feature space. In this regard, t-distributed stochastic neighbor embedding (t-SNE) [7] was used to project the last hidden activations of a four-layer network and that of a single layer as shown in Figures 32a and b, respectively. The t-SNE projections show that network with four hidden layers has clustered activations compared to that of a single layer resulting in within class holes. This observation is pronounced for activations of digit 7.

CIFAR-10 dataset [132] was used in the second set of large-scale experiments to validate and retrain pruned models. The dataset contains a labeled set of 60,000 32x32 color images belonging to 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50000 and 10000 training and testing sets, respectively. FLOP was used to compare the computational efficiency of the models because its evaluation is independent of any underlying software and hardware. In order to fairly compare proposed method with state-of-the-art, the FLOP was only calculated for the convolution and fully connected layers. For CIFAR-10 dataset, the proposed redundant-feature-based pruning was evaluated on three deep networks, namely: VGG-16 [133] and two residual networks ResNet-56 and 110 [96]. The baseline accuracy for residual networks were obtained by following the procedures highlighted in [96].

1. VGG-16 on CIFAR-10

In this set of experiments, a modified version of the popular convolutional neural network known as the VGG-16 [133] was used. It has 13 convolutional layers and 2 fully connected layers. In the modified version, each layer of convolution is followed by a Batch Normalization layer [134]. A base model was trained for 350

epochs, with a batch-size of 128 and a learning rate 0.1. The learning rate was reduced by a factor of 10 at 150 and 250 epochs. After pruning, the pruned model was finetuned with learning rate of 0.001 for 80 epochs to adjust the weights of the remaining connections to regain the accuracy.

Figure 33 shows the number of nonredundant filters per layer for different

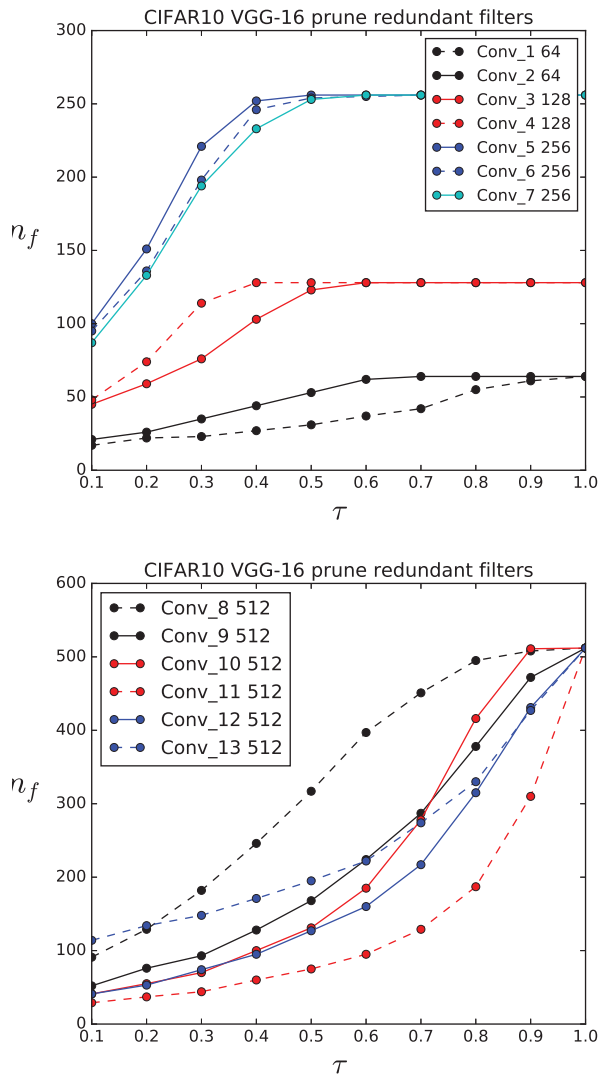
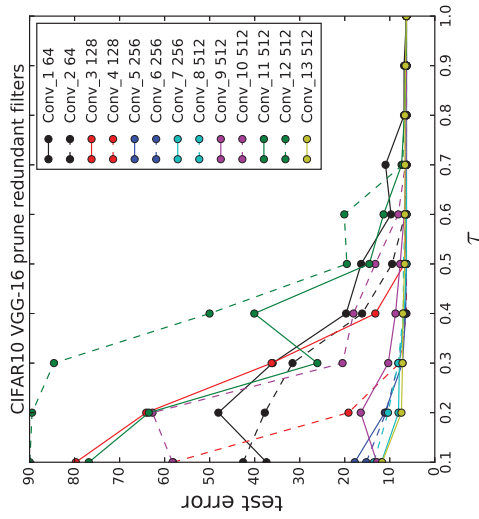


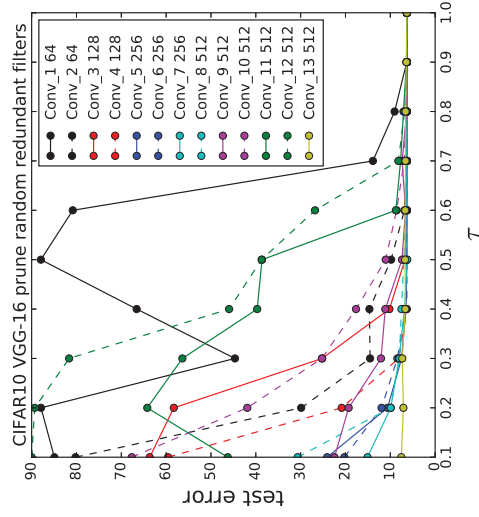
FIGURE 33: Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for VGG-16 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend [3].

τ values. As can be seen that some convolutional layers in VGG are prone to ex-

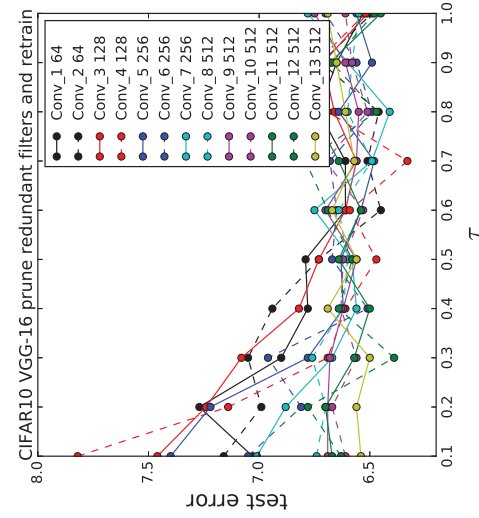
tracting features with very high correlation; examples of such as layer are layers 1, 11, 12, and 13. Another very important observation is that later layers of VGG are more susceptible to extracting redundant filters than earlier layers and can be heavily pruned. Figure 34(a) shows the sensitivity of VGG-16 layers to pruning and it can be observed that layers such as Conv 1, 3, 4, 9, 11, and 12 are very sensitive. However, as can be observed in Figure 34(c), accuracy can be restored after pruning filters in later layers (Conv 9, 11, and 12) compared to early ones (Conv 1, 3, and 4).



(a) Prune $n' - n_f$ redundant filters



(b) Prune $n' - n_f$ random filters



(c) Prune redundant filters and retrain

FIGURE 34: Sensitivity to pruning (a) redundant filters (b) random $n' - n_f$ filters, and (c) redundant filters and retraining for 30 epochs for VGG-16 [3].

For the final test score, the pruned model is finetuned on the entire training set. In the pruning stage, a grid search was performed over τ values within 0.1 and 1.0, and found 0.54 gave the least test error. Table 9 reports the pruning performance for $\tau = 0.54$ and it can be easily observed that more than 90% of most of the latter layer filters have been pruned and most of the sensitive earlier layers are minimally pruned. Figure 34(b) depicts the sensitivity of trained VGG-16 model to pruning using heuristic in Algorithm 5 that calculates the number of redundant filters ($n' - n_f$) and randomly prunes them.

As seen in Table 10, for $\tau = 0.54$ our approach in Algorithm 4 outperforms that Absolute filter sum approach [109], Network Sliming [135], Try-and-learn [126] and are able to prune more than 78% of the parameters resulting in 40% FLOP reduction and a competitive classification accuracy. In addition, when τ was tuned to 0.46, more than 40% FLOP reduction was achieved outperforming variational method [136], which is one of the state-of-the-art. It is suspected that proposed pruning approach outperforms other methods because it localizes and prunes similar or shifted versions of filters that do not add extra information to the feature hierarchy. This notion is reinforced from information theory standpoint that the activation of one unit should not be predictable based on the activations of other units of the same layer [137]. Another crucial observation is that heuristic A achieves a better accuracy than Method B because random pruning is suspected to remove dissimilar filters. It is strongly believe that Algorithm 4 performs better than 5 because of its precise ability to remove redundancy. However, Algorithm 4 is a bit slower than 5 and that is the trade-off.

2. RESNET-56/110 on CIFAR-10

The architecture of residual networks is more complex than VGG and also the number of parameters in the fully connected layer is relatively smaller and

layer	$v_l \times h_l$	#Maps	FLOP	#Params	#Maps	FLOP%
Conv_1	32×32	64	1.8E+06	1.7E+03	32	50.0%
Conv_2	32×32	64	3.8E+07	3.7E+04	58	54.7%
Conv_3	16×16	128	1.9E+07	7.4E+04	125	11.5%
Conv_4	16×16	128	3.8E+07	1.5E+05	128	2.3%
Conv_5	8×8	256	1.9E+07	2.9E+05	256	0%
Conv_6	8×8	256	3.8E+07	5.9E+05	254	0.8%
Conv_7	8×8	256	3.8E+07	5.9E+05	252	2.3%
Conv_8	4×4	512	1.9E+07	1.2E+06	299	42.5%
Conv_9	4×4	512	3.8E+07	2.4E+06	164	81.3%
Conv_10	4×4	512	3.8E+07	2.4E+06	121	92.4%
Conv_11	2×2	512	9.4E+06	2.4E+06	59	97.3%
Conv_12	2×2	512	9.4E+06	2.4E+06	104	97.7%
Conv_13	2×2	512	9.4E+06	2.4E+06	129	94.9 %

TABLE 9

Pruning performance on CIFAR dataset using VGG-16 model at $\tau = 0.54$ [3].

VGG-16 Model	% Accuracy drop	% FLOP Pruned	% Parameters Pruned
Methods			
[109]	0.40	34.2	64.0
[135]	-0.17	38.6	-
[126]	0.60	34.2	-
[136]	0.81	62.9	-
Ours-A ($\tau = 0.54$)	0.13	40.5	78.1
Ours-B ($\tau = 0.54$)	0.50	40.5	78.1
Ours-A ($\tau = 0.46$)	0.72	65.1	89.5

TABLE 10

Performance evaluation for three pruning techniques on CIFAR-10 dataset. Performance with the lowest test error is reported [3].

this makes it a bit challenging to prune a large proportion of the parameters. Both ResNet-56 and ResNet-110 have three stages of residual blocks for feature maps of differing sizes. The size ($v_l \times h_l$) of feature maps in stages 1,2, and 3 are 32×32 , 16×16 , and 8×8 , respectively. Each stage has 9 and 18 residual blocks for ResNet-56 and ResNet-110, respectively. A residual block consists of two convolutional layers each followed by a Batch Normalization layer. Preceding the first stage is a convolutional layer followed by a Batch Normalization layer². Only the redundant filters in first convolution layer of each block are pruned since the mapping for selecting identity feature maps is unavailable.

²The Pytorch implementation of ResNet56/110 in <https://github.com/D-X-Y/ResNeXt-DenseNet> was used as baseline models

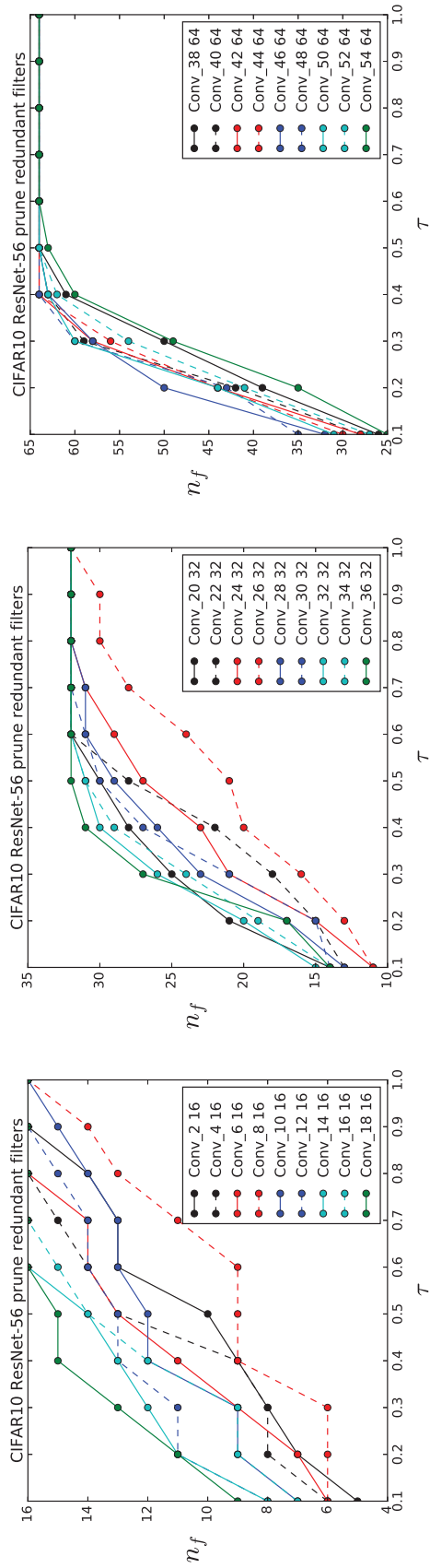


FIGURE 35: Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for ResNet-56 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend [3].

Model	Error %	FLOP	Pruned %	# Parameters	Pruned %
ResNet-56	6.61	1.25×10^8		8.5×10^5	
[109]	6.94	9.09×10^7	27.6%	7.3×10^5	13.7%
Ours-A	6.88	9.07×10^7	27.9%	6.5×10^5	23.7%
Ours-B	6.94	9.07×10^7	27.9 %	6.5×10^5	23.7 %
ResNet-110	6.35	2.53×10^8		1.72×10^6	
[109]	6.70	1.55×10^8	38.6%	1.16×10^6	32.4%
Ours-A	6.73	1.54×10^8	39.1%	1.13×10^6	34.2%
Ours-B	7.41	1.54×10^8	39.1%	1.13×10^5	34.2%

TABLE 11

Performance evaluation of three pruning techniques for ResNet 56/110 trained on CIFAR-10 dataset. Performance with the lowest test error is reported [3].

As can be observed in Figures 35 and 36 that convolutional layers in first stage are prone to extracting more redundant features than those of second stage, and the convolutional layers in the second stage are susceptible to extracting redundant filters than those of third block, which is contrary to the observations in experiments with VGG-16. In effect, more filters could be pruned from layers in first stage than latter stages without losing much to accuracy. More specifically, many layers in first stage of ResNet-56, such as Conv 2,8,10, and 26, have filters that are more than 80% correlated and could be easily pruned. Similarly, convolutional layers in the first stage of ResNet-110 exhibit similar tendency to produce more filters that are redundant. Due to differing redundancy tendencies at each stage, τ is customized for each of the stages. In pruning ResNet-56, τ is set to 0.253, 0.223, 0.20 as thresholds for stages 1,2, and 3, respectively. Similarly for ResNet-110 τ values 0.18, 0.12, and 0.17 were used.

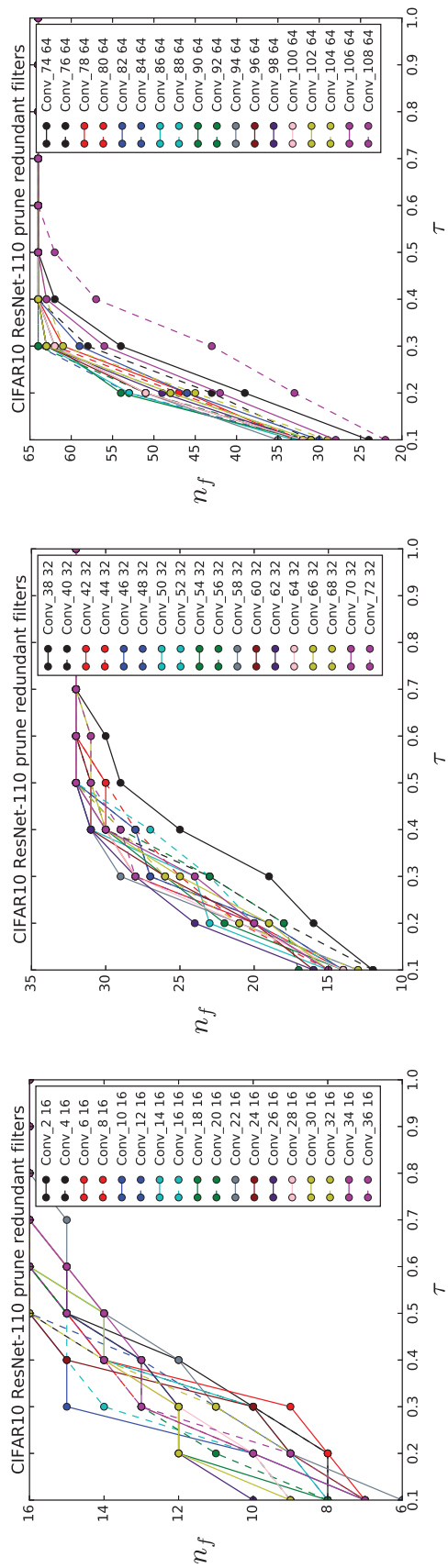


FIGURE 36: Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for ResNet-110 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend [3].

Figure 37 shows the sensitivity of ResNet-56 layers to pruning and it can be observed that layers such as Conv 10, 14, 16, 18, 20, 34, 36, 38, 52 and 54 are more sensitive to filter pruning than other convolutional layers. Likewise for ResNet-110, the layer sensitivity to pruning is depicted in Figure 38 and it can be observed that Conv 1, 2, 38, 78, and 108 are sensitive to pruning. In order to regain the accuracy by retraining the pruned model, these sensitive layers were also skipped while pruning.

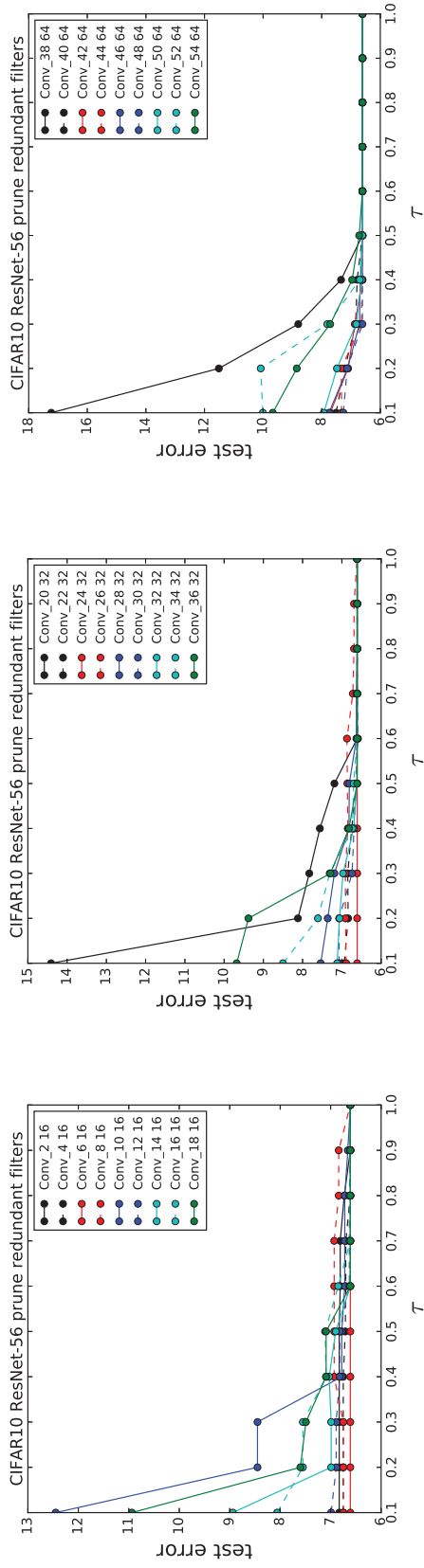


FIGURE 37: Sensitivity to pruning $n' - n_f$ redundant convolutional filters in ResNet-56 [3].

As seen in Table 11 for ResNet-56, redundant-feature-based pruning methods A and B have competitive performance in terms of FLOP reduction and outperform that in [109]. Proposed approach reduces the number of effective parameters by 10% with relatively better classification accuracy after retraining. However, the effective number of parameters pruned was marginally increased in ResNet-110 from 38.6% to 39.1%, resulting in approximately 2% increase.

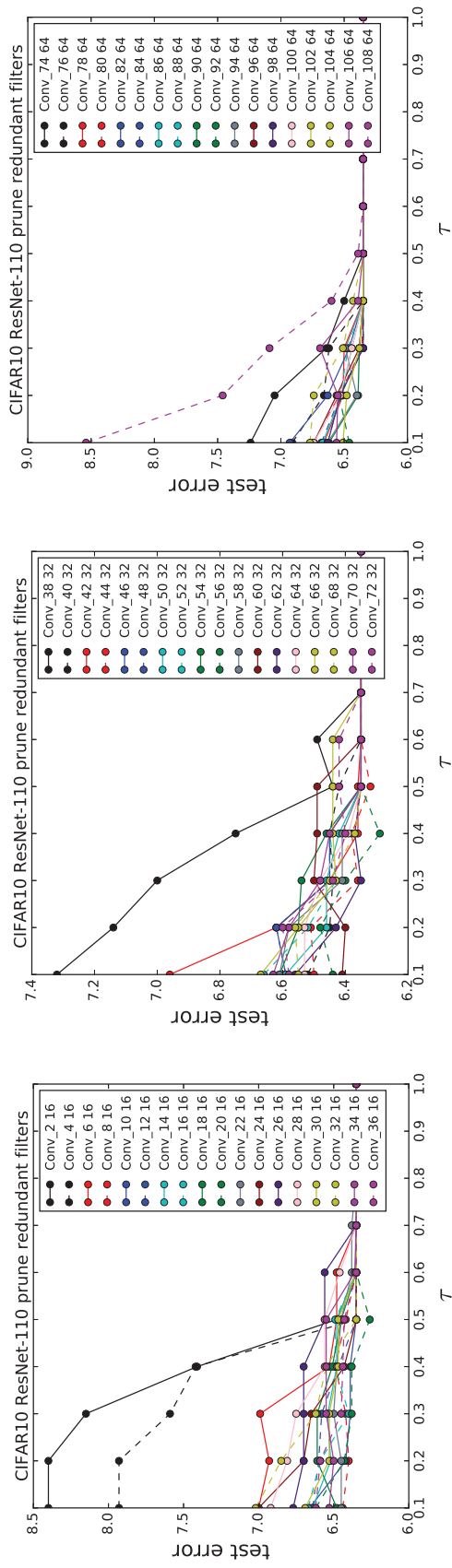


FIGURE 38: Sensitivity to pruning $n' - n_f$ redundant convolutional filters in ResNet-110 [3].

Model	FLOP	Pruned %	Time(s)	Saved %
VGG-16	3.13×10^8		1.47	
Ours-A	1.86×10^8	40.5%	0.94	34.01%
ResNet-56	1.25×10^8		1.16	
Ours-A	9.07×10^7	27.9%	0.96	17.2%
ResNet-110	2.53×10^8		2.22	
Ours-A	1.54×10^8	39.1%	1.80	18.9%

TABLE 12

FLOP and CPU time reduction for inference. Operations in convolutional and fully connected layer are considered for computing FLOP [3].

The inference times for original and pruned models are reported in Table 12. 10,000 test images of CIFAR-10 dataset were used for the timing evaluation conducted in Pytorch version 0.2.0_3 with Titan X (Pascal) GPU and cuDNN v8.0.44, using a mini-batch of size 100. It can be observed that %FLOP reduction also translates almost directly into inference CPU time savings.

3. Prune and Train from Scratch

In order to see the effect of copying weights from the original (larger) model to a pruned (smaller) model, two models (VGG-16 and ResNet-56) were pruned as described above, re-initialized their weights, and trained them from scratch. As shown in Table 13 that fine-tuning a pruned model is almost always better than re-initializing and training a pruned model from scratch. It is observed that already-trained filters may serve as good initialization for a smaller network which might on its own be difficult to train. Other observation from Table 13 is that redundant-feature-based pruning results in an architecture that attains a better performance than its counterpart in [109]. This suggests that redundant-feature-based pruning might be a potential approach to determining the architectural width of modern

Model	Error %
VGG-16	
Pruned [109]	6.60
Pruned (Ours-A)	6.33
Pruned-scratch-train [109]	6.88
Pruned-A-scratch-train (Ours-A)	6.79
ResNet-56	
Pruned [109]	6.94
Pruned (Ours-A)	6.88
Pruned-scratch-train [109]	8.69
Pruned-scratch-train (Ours-A)	7.66

TABLE 13

Performance on CIFAR dataset [3].

deep neural network.

C. Conclusion

Motivated by the observations of recent studies that modern deep neural network models often have large number of overlapping features amounting to unnecessary filtering redundancy and high inference cost. By grouping features at each layer according to a predefined measure in parameter space using agglomerative hierarchical clustering, it is shown in this chapter that redundancy can be eliminated, inference cost (FLOPS) is reduced by 60% for VGG-16, 28%/39% for ResNet-56/110 trained on CIFAR-10, and 28% for ResNet-34 trained on ImageNet database. To recover the accuracy after pruning, models were finetuned for a few iterations without the need to modify hyper-parameters.

CHAPTER V

FEATURE DIVERSIFICATION IN DEEP NEURAL NETWORKS

The expressiveness of deep neural networks, usually with huge number of trainable parameters, sometimes comes at a disadvantage when trained on limited amount of data due to their susceptibility to overfitting. To circumvent this problem, a plethora of regularization and initialization methods such as weight decay, dropout [58], and weight initialization [70] have been purported to ameliorate overfitting and convergence problems resulting from data scarcity and network size [24]. Moreover, recent advances in deep learning for image classification [96,138], language processing [102,103], speech synthesis and recognition [99–101] have been attributed to efficient regularization of randomly initialized deep and complex models trained with Stochastic Gradient Descent (SGD).

Over the last few decades, research focused on strategies for reducing overfitting and improving the capabilities of deep neural network. Examples of such strategies include Batch Normalization [134] that aims to minimize the internal covariance shift. Also, keeping similar variance at each layer’s input and output of deep network using initialization has shown to preserve signal propagation and improve generalization [68,70]. Orthonormal initialization coupled with output variance normalization has also been shown as decorrelating neural network’s initial weights for better convergence [139].

Another important and popular paradigm for reducing overfitting is regularization. In general, the two most commonly used regularization paradigms utilize the hidden activations, weights or output distribution. The first family of regularization strategy aims to extenuate the model complexity by using weight decay

[1, 140] to reduce the number of effective model parameters, or using dropout [58] to randomly drop hidden activations, or using DropConnect [141] to randomly drop weights during training. Even though these methods have shown improvements on generalization, they regularize in a manner that under-utilizes the capacity of the model. The second family of regularization methods focuses on improving the generalization without undermining the model's capacity. For instance, [23] presented pre-training algorithms to learn decorrelated features and [142] discusses decorrelated activations using incoherent training.

Other mechanisms that also fall in the second category are those that regularize the output distribution. In this sense, entropy-based regularizer with deterministic annealing was applied to train multilayer perceptrons for the purpose of avoiding poor initialization, local minima, and for improving model generalization [143]. Regularization has also been applied in form of label smoothing for estimating the marginalized effect of label-dropout during training. This, in effect, reduces overfitting by restricting the model from assigning full probability to each training sample and maintaining a reasonable ratio between the logits of the incorrect classes [110]. Label smoothing has also been achieved using a teacher model [144] instead of smoothing with uniform distribution as in [110]. Injecting label noise has equally been shown to have a tremendous regularizing effect [145]. Moreover, models with high level of overfitting have recently been shown to assign all output probability to a single class in the training set, thus giving rise to output distributions with low entropy - a phenomenon referred to as overconfidence [110]. Methods for effective regularization of overconfident networks have also been reported that penalize the confident output distribution [146].

As reinforced in Chapters IV and V that over-sized deep neural networks typically produce a high level of overfitting and usually rely on many redundant features that can be either shifted version of each other or be very similar with little

or no variations. For instance, this redundancy is evidently pronounced in features learned by popular deep neural network architecture such as AlexNet [73] as emphasized in [24, 108]. To address this redundancy problem, layers of deep and/or wide architectures have to be trained under specific and well-defined sets of constraints in order to remove this limitation during training. The most closely related work is the recently introduced regularization technique known as OrthoReg [24] that locally enforces feature orthogonality by removing interference between negatively correlated features. The key idea addressed is to regularize positively correlated features during training. In effect, OrthoReg reduces overfitting by enforcing higher decorrelation bounds on features. Proposed algorithms, on the other hand, aim at regularizing both negatively and positively correlated features according to their differentiation and based on their relative cosine distances. This way only features correlated above a certain correlation threshold are penalize, thus strengthening feature diversity as training progresses. This approach affords the flexibility of choosing the absolute correlation bound. Hence, the proposed method leads to elimination of redundancy of features and better generalization.

Other related work is [147], which aims at training neural networks for classification with few training samples by constraining the hidden units to learn class-wise invariant features, with samples of the same class having the same feature representation. It is remarked that the proposed methods have the flavor of the two aforementioned families of regularization in the sense that we aim to improve generalization without undermining the model's capacity by bounding the pairwise correlation of features and at the same time temporarily drop redundant feature maps during training.

The problem addressed in this chapter is four-fold: (i) an optimized algorithm that inhibits learning of redundant filters is proposed, thereby enforcing the extraction of diverse features (ii) using hierarchical agglomerative clustering

(HAC) to drop activations (or feature maps) of redundant features during training is also proposed, (iii) heuristics that eliminate the computational overhead introduced by HAC for very deep and/or wide neural networks by using the pairwise feature correlation to compute the fraction of the feature maps to be dropped during training is also proposed, and lastly (iv) the proposed regularization methods are shown to improve state-of-the-art models across many benchmark learning tasks and datasets.

A. Enhancing Feature Diversity by enforcing Dissimilar Feature Extraction

The objective here is to enforce constraints on the learning process by simply encouraging diverse feature learning and preventing the extraction of redundant features that are very similar or shifted version of one another. A symptom of learning replicated or similar features is that two or more processing units extract very similar and correlated information. From an information theory standpoint, similar or shifted versions of filters do not add extra information to the feature hierarchy, and therefore should be possibly suppressed. In other words, the activation of one unit should not be predictable based on the activations of other units of the same layer. It is remarked that convolutional filtering have found to greatly benefit from diversity or orthogonality of filters because it can alleviate gradient vanishing or exploding problems [69,148]. Enforcing feature dissimilarity in traditional way can be generally involved and would require computation of huge joint probability table and batch statistics which can be computationally intractable [24].

One tractable way of computing correlation between two features is by evaluating the cosine similarity measure (SIM_C) between them:

$$SIM_C(\mathbf{w}_1, \mathbf{w}_2) = \frac{\langle \mathbf{w}_1, \mathbf{w}_2 \rangle}{\|\mathbf{w}_1\| \|\mathbf{w}_2\|} \quad (27)$$

where $\langle \mathbf{w}_1, \mathbf{w}_2 \rangle$ is the inner product of arbitrary feature vectors \mathbf{w}_1 and \mathbf{w}_2 . The

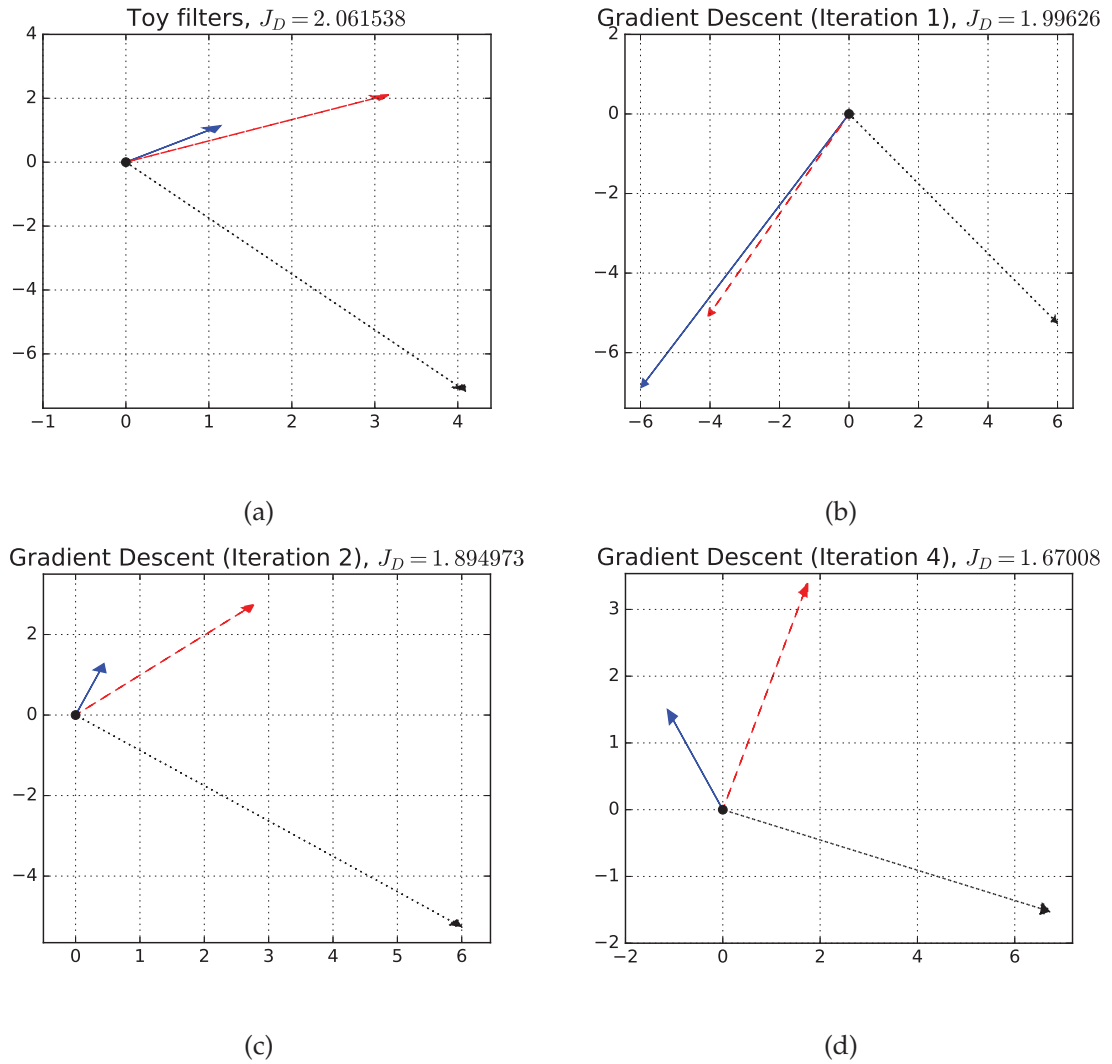


FIGURE 39: Illustration of effect of divReg with $\lambda = 10$ and $\tau = 0.1$ (a) on three toy filters in (b) iteration 1 (c) iteration 2 and (d) iteration 4 [4].

similarity between two feature vectors corresponds to correlation between them, that is, the cosine of the angle between the feature vectors in the feature space. Since the entries of the vectors can take both negative and positive values, SIM_C is bounded by $[-1,1]$. It is 1 when $\mathbf{w}_1 = \mathbf{w}_2$ or when \mathbf{w}_1 and \mathbf{w}_2 are identical. SIM_C is -1 when the two vectors are in exact opposite direction. The two filter vectors are orthogonal in feature space when SIM_C is 0. The corresponding distance measure is given as $D_C = 1 - SIM_C$.

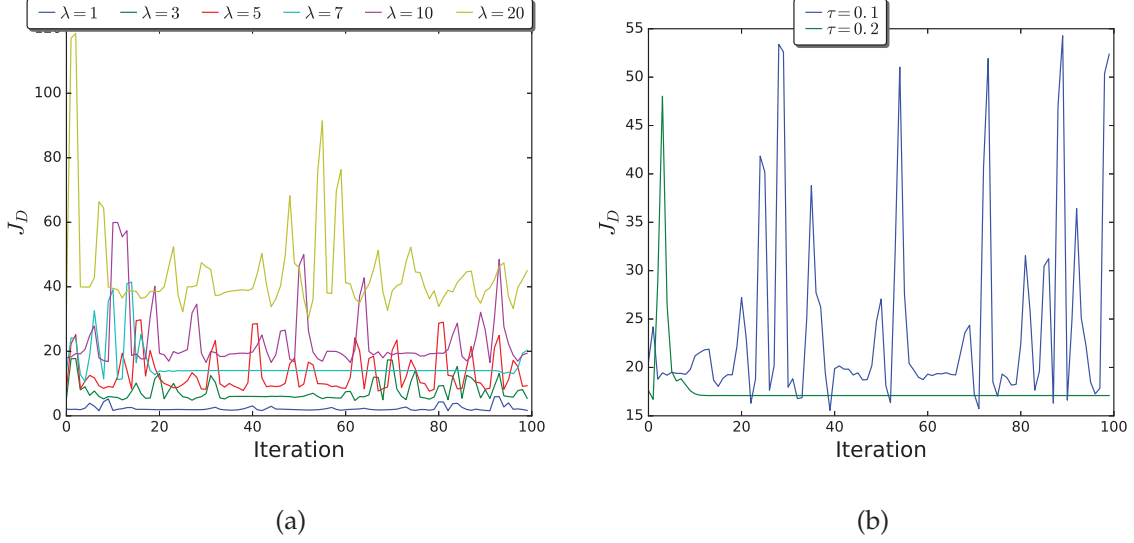


FIGURE 40: Effect of (a) diversity penalty factor λ and (b) thresholding parameter τ on diversity regularization cost J_D (Figure best viewed in color) [4]

1. Diversity Regularization

In order to minimize the extraction of redundant features during training, it is necessary to maximize the information encoded by each processing hidden units by incorporating a penalty term into the overall learning objective, here referred to as diversity regularization (divReg). The constraints induced as a result of diversity regularization term need to be reconciled with usual regularization through a judicious choice of appropriate penalty function. The diversity regularization cost (J_D) for l^{th} layer of the deep network is thus defined as:

$$J_D^{(l)}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n'} \sum_{j=1, j \neq i}^{n'} m_{i,j}^{(l)} (\text{SIM}_C(\mathbf{w}_i^{(l)}, \mathbf{w}_j^{(l)}))^2 \quad (28)$$

where \mathbf{w}_i are the weights connecting the activations of layer $l - 1$ to i^{th} neuron of layer l , n' is the number of neurons in layer l . $m_{i,j}$ is a binary mask variable defined as

$$m_{i,j} = \begin{cases} 1 & |SIM_C(\mathbf{w}_i, \mathbf{w}_j)| \geq \tau \\ 0 & i = j \\ 0 & otherwise \end{cases} \quad (29)$$

and $0 \leq \tau \leq 1$ is an hyperparameter. It is worthy to note that self correlation of each feature vector w_i has been discarded in (29). Also, both negative and positive correlations above the threshold τ are taken into consideration. This implies feature pair with $|SIM_C|$ below τ will not be penalized.

It is important to also note the importance and relevance of τ in (29). Setting $\tau = 0$ results in orthogonal feature set and this is in most cases neither desirable nor practical because some features are still required to be shared. For instance, if we consider a model trained on CIAFR-10 dataset [132] that has "automobiles" and "trucks" as two of its ten categories. If a particular lower-level feature describes the "wheel", then, it will not be out of place if two higher-level features describing automobile and truck share common feature that describes the wheel. The choice of τ determines the level of sharing allowed, that is, the degree of feature sharing across features of a particular layer. In other words, τ serves as a trade-off parameter that ensures some degree of feature sharing across multiple high-level features and at the same time ensuring features are sufficiently dissimilar.

By letting $\Phi \in \mathbb{R}^{n \times n'}$ contain n' normalized filter vectors (receptive fields) $\phi_i = \mathbf{w}_i / \sqrt{\|\mathbf{w}_i\|^2}$ as columns, each with n elements corresponding to connections from layer $l - 1$ to i^{th} neuron of layer l , then, J_D for all L layers can be rewritten as a sum:

$$J_D(\phi) = \sum_{l=1}^L \left(\frac{1}{2} \sum_{i=1}^{n'} \sum_{j=1}^{n'} \left(\Omega_{ij}^{(l)} \right)^2 \mathbf{M}_{ij}^{(l)} \right) \quad (30)$$

where $\Omega \in \mathbb{R}^{n' \times n'}$ denotes $\Phi^T \Phi$ which contains the inner products of each pair of columns i and j of Φ in each position i, j of Ω in layer l ; $\mathbf{M} \in \mathbb{R}^{n' \times n'}$ is a binary

mask for layer l defined in (31); L is the number of layers to be regularized.

$$\mathbf{M}_{i,j} = \begin{cases} 1 & \tau \leq |\Omega_{i,j}| \leq 1 \\ 0 & i = j \\ 0 & \text{otherwise} \end{cases} \quad (31)$$

In order to enforce diversity of features while training, the diversity regularization term (30) is added to the learning loss function $J(\theta; \mathbf{X}, \mathbf{y})$, where θ comprises of network's weights (\mathbf{W}) and biases (\mathbf{b}); \mathbf{X} , \mathbf{y} are the data matrix and label vector, respectively. The overall cost is then

$$J_{net} = J(\theta; \mathbf{X}, \mathbf{y}) + \lambda J_D(\phi) \quad (32)$$

where λ is the diversity penalty factor experimentally chosen to be 10. The weights are updated as below using the error backpropagation:

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \xi \frac{\partial}{\partial w_{i,j}^{(l)}} J_{net} \quad (33)$$

$$b_j^{(l)} = b_j^{(l)} - \xi \frac{\partial}{\partial b_j^{(l)}} J_{net} \quad (34)$$

where $\xi > 0$ is the learning rate and the gradient of the loss function is computed as in (35).

$$\frac{\partial}{\partial w_{i,j}^{(l)}} J_{net} = \nabla_{w_{i,j}^{(l)}} J(\theta; \mathbf{X}, \mathbf{y}) + \lambda \nabla_{w_{i,j}^{(l)}} J_D(\phi) \quad (35)$$

and

$$\nabla_{w_{i,j}^{(l)}} J_D(\phi) = \sum_{k=1}^n \Phi_{i,k}^{(l)} \Omega_{k,j}^{(l)} \mathbf{M}_{k,j}^{(l)} \quad (36)$$

2. Implications of imposing feature diversity

The graphical illustration of impact of diversity regularization on features is shown in Fig. 39. Since this illustration does not utilize training data to update

feature matrix $W^{(l)}$ in Eq.(33), $\nabla_{\mathbf{W}^{(l)}} J^{(l)}(\theta; \mathbf{X}, \mathbf{y})$ is thus set $\rightarrow 0$. The three 2D filters shown as vectors in Fig. 39a were synthesized for visual illustration and both τ and λ were set to be 0.1 and 10, respectively. $J_D(\phi)$ as a result of three initial filters evaluates to 2.062 using Eq.(30) with $L = 1$. Making a step along the gradient reduces the diversity regularization cost to 1.996 as shown in Fig. 39b. Likewise, the updated features after second and fourth iterations of gradient descent resulted in diversity regularization cost of 1.895 and 1.67 as shown in Figs. 39c and d, respectively. It is observed that at every iteration, the optimizer is forced to find features that are less similar in order to minimize $J_D(\phi)$.

Another crucial observation is that the filter distant from others in feature space is less regularized and has little influence on the regularization of other filters. The effect of both diversity penalty factor λ and thresholding parameter τ on diversity regularization cost J_D is shown in Figs. 40a and b, respectively. As expected, J_D increases as the value of λ is increased for $\tau = 0.1$. The effect of τ on J_D is also explored and it can be observed in Figs. 40b that when $\tau = 0.1$, the features are regularized more aggressively due to more feature-pair having similarity exceeding this threshold value and leading to a situation whereby feature vectors are heavily updated in every iteration leading to fluctuations of J_D . In contrast, when $\tau = 0.2$, features are heavily updated in the first fifteen iterations and subsequently converges into a local optimum.

B. Online Redundant Filter Detection and Dropout

This section introduces the concept of online agglomerative hierarchical clustering of features for detecting and dropping of N_r redundant features and their maps during training originally introduced in [88] and cited in [2] for pruning redundant features in unsupervised pretraining. In this section, two online

dropout heuristics considered both aim at online detection of redundant features and:

1. **dropping of redundant features maps** in the forward pass during training. Here, clustering of features aims at automatically detecting the features whose activations/maps will be dropped in each training epoch.
2. **random dropping of N_r feature maps** during training.

The above two criteria are alternative approaches and they both aim at temporarily dropping a set of feature maps during training. The term redundant reflects a choice of a specific chosen measure, SIM_C and of the τ value.

1. Online Filtering Redundancy Dropout

The objective here is to dropout feature maps that have identical or very similar features in the weight space according to well-defined similarity measure and a chosen cluster similarity threshold denoted as τ^* as shown in Algorithm 6. τ^* is an hyperparameter that has to be set in order to achieve optimal performance. The nitty-gritty of the redundant feature dropout procedure is detailed in Algorithm 6. It first initializes weights to small random numbers by following the method introduced in [70]. Training data are shuffled and split into batches in each epoch. The loss in (32) is computed on each batch of the training samples. The backpropagation algorithm computes the gradient of the loss with respect to all the model parameters. Weights and biases are updated using the update rules in (33) and (34), respectively. At the end of every epoch, the weights connecting the activations of layer $l - 1$ to neurons of layer l are examined for possible similarity. The objective here is to discover n_f clusters in the set of n' original weight vectors (or simply features), where $n_f \leq n'$. Upon detecting these distinct n_f clusters, a representative feature from each of these n_f clusters is randomly sampled without

replacement and the remaining set of features are tagged as redundant (S_R). This process continues for prescribed number of epochs.

The detection of redundant feature vectors is generally tractable especially from practical standpoint since the number of features in each layer is reasonably sized (mostly less than a thousand).

2. Online Redundancy-based Dropout

The complexity of agglomerative clustering in Algorithm 6 is $O((n')^2 \log(n'))$, which might sometimes make it impractical to deploy in online settings (that is, during training) especially for large n' and l . For instance, clustering 1024 feature vectors empirically takes on average on our machine (see specs in the Section C) 12 seconds and this is executed at least once in every epoch. This amounts to at least additional $(12 * l * nbepoch)$ seconds of computational overhead, where l is the number of layers and $nbepoch$ is number of epochs. However, the computational overhead is practical for relatively shallow network architectures.

To circumvent this problem for very deep and wide networks, Algorithm 7 is proposed to estimate the dropout fraction based on the number of feature pairs that are correlated above a set threshold τ^* . It uses cosine similarity with thresholding mechanism to dynamically set the dropout fraction of conventional dropout regularizer. This incorporates the redundancy information in the dropout mechanism. It is worth motivating and mentioning that Algorithms 6 and 7 are alternative approaches and should be used independently. The main difference between these algorithms is that Algorithm 6 uses hierarchical agglomerative clustering to detect and drop out the exact redundant features in each epoch, while Algorithm 7 estimates the number of feature maps to be randomly dropped at each epoch. Computationally, the dropout fraction of layer l in each epoch in Algorithm 7 is computed as the mean of the upper (or lower) triangular part of matrix

Algorithm 6: Online Redundant Feature Dropout (divReg-1)

- 1 {The parameters are: B_S - the batch size, ζ - learning rate, n' - number of filters, τ - diversity regularization correlation threshold, and τ^* - filter clustering similarity threshold}
 - 2 { θ is the vector of concatenated weights ($\mathbf{W}^{(l)}$) and biases ($\mathbf{b}^{(l)}$). Initialize θ from a normal distribution as proposed in [70]. Initialize dropout fraction α }
 - 3 $\theta \leftarrow \theta_0$ {Initial weight and biases}
 - 4 **for** prescribed number of epochs ($nbepoch$) **do**
 - 5 permute training samples
 - 6 **for all** batches of B_S train samples **do**
 - 7 $J_{net} \leftarrow$ loss on batch samples from eq. (32)
 - 8 $\Delta\theta \leftarrow$ compute gradient using eq. (33) and (34)
 - 9 {Make a step along the gradient}
 - 10 $\theta \leftarrow \theta - \zeta\Delta\theta$
 - 11 **end for**
 - 12 {Compute the set of redundant features}
 - 13 $S_R \leftarrow$ **FilterClustering10**
 - 14 {Drop activation maps corresponding to features in S_R }
 - 15 **end for**
 - 16 **FilterClustering10**:
 - 17 **Input:** $\mathbf{W}^{(l)}, \tau^*$
 - 18 Scan for cluster(s) of vectors in $\mathbf{W}^{(l)}$ with $\overline{SIM}_C > \tau^*$
 - 19 {Randomly sample and tag one representative feature from each of the n_f clusters as non-redundant}
 - 20 **Output:** Set of redundant features in $\mathbf{W}^{(l)}$
-

\mathbf{M}^* as in (37) below:

$$\alpha^{(l)} = \frac{\sum_{i=1}^{n'} \sum_{j=1}^{n'} \mathbf{M}^*(i, j)^{(l)}}{(n')^2 - n'} \quad (37)$$

where

$$\mathbf{M}^*(i, j) = \begin{cases} 1 & \tau^* \leq |\Omega(i, j)| \leq 1 \\ 0 & i = j \\ 0 & \text{otherwise.} \end{cases} \quad (38)$$

It must be noted that both Algorithms 6 and 7 are adaptive in the sense that they adapt accordingly in every epoch to varying number of redundant filters. Another crucial detail about Algorithm 7 is the initialization of α . Different initialization values [0, 0.25, 0.5, 0.75] were experimented with, and it is found different values work best for different datasets as will be detailed in Section IV. Unlike conventional dropout [58] that randomly drops a fixed number of units throughout the training process, the number of units dropped during training using Algorithms 6 and 7 adapts accordingly as training progresses. In this section, training under the diversity regularization in (30) without dropout as divReg while training using Algorithms 6 and 7 is denoted as divReg-1 and divReg-2, respectively.

Each of divReg-1 and divReg-2 can be used in tandem with the diversity regularization introduced in the previous section. However, they could also be deployed as stand-alone regularization tools in which case the regularization term in (30) is discarded by setting $\lambda = 0$. It must be noted that when using any of these procedures in conjunction with diversity regularization term (when $\lambda \neq 0$), the width of the similarity bound $[-\tau, \tau]$ must be chosen as large as possible to allow the detection of some similar features and also $\tau^* \leq \tau$.

Algorithm 7: Online Redundancy-dependent Dropout (divReg-2)

- 1 {The parameters are: B_S - the batch size, ζ - learning rate, n' - number of filters, α - dropout fraction, N_r - number of redundant filters, τ - diversity regularization correlation threshold, and τ^* }
 - 2 θ is the vector of concatenated weights ($\mathbf{W}^{(l)}$) and biases ($\mathbf{b}^{(l)}$).
 - 3 Initialize θ from a normal distribution as proposed in [70].
 - 4 $\theta \leftarrow \theta_0$ {Initial weight and biases}
 - 5 { Initialize α - dropout fraction}
 - 6 **for** prescribed number of epochs ($nbepoch$) **do**
 - 7 permute training samples
 - 8 $N_r \leftarrow 0$ {Initial number of redundant features}
 - 9 **for all** batches of B_S train samples **do**
 - 10 $J_{net} \leftarrow$ loss on samples in the batch b from eq. (32)
 - 11 $\Delta\theta \leftarrow$ compute gradient using eq. (33) and (34)
 - 12 {Make a step along the gradient}
 - 13 $\theta \leftarrow \theta - \zeta\Delta\theta$
 - 14 **end for**
 - 15 {Compute the binary mask $\mathbf{M}^{*(l)}$ in (38) for every layer}
 - 16 {Compute and update α in (37) for every layer l }
 - 17 **end for**
-

C. Experiments

Diversity regularization (divReg) was evaluated on MNIST dataset of handwritten digits [130], CIFAR-10 [132], and Stanford Natural Language Inference (SNLI) Corpus [149]. The software implementation has been in Keras library³ with Tensorflow [150] backend on two Titan X 12GB GPUs⁴.

1. Feature Evolution during Training

In the preliminary experiment, a multilayer perceptron with two hidden layers was trained using MNIST digits. Each layer has 1024 ReLU-activated hidden units and Adam optimizer [131] with batch size of 128 was used to train the model for 300 epochs and τ and ζ in divReg was both set to 0.05. The hyperparameters of OrthoReg was set as reported in [24]. Fig. 41 shows the distribution of pairwise correlation of first hidden layer features ($\Omega = \Phi^{(1)T} \Phi^{(1)}$) in the beginning and end of training. It can be observed that divReg was able to constrain the pairwise feature correlations between the desired bound (-0.05 and 0.05) compared to the highly correlated features extracted by unregularized counterpart. Although OrthoReg was able to eliminate all the positively correlated features using exponential squashing function, but it did so in a more rigid way which could lead to extraction of noisy features. Similarly in Figs. 42a and b, the pairwise feature correlations of the second hidden layer have been bounded by the set threshold for divReg, unconstrained for unregularized model, and negatively correlated with tight bound for OrthoReg.

Table 14 reports the performance of divReg along with four other regularization techniques. All reported results are average performance over 5 independent

³<https://keras.io/keras-the-python-deep-learning-library>

⁴Implementation of divReg can be found in <https://github.com/babajide07/Diversity-Regularization-Keras-Implementation>

trials alongside with their standard deviation. The results are separated and compared on the basis of the class of the regularization technique. It can be observed that Dropout outperforms L_1 in terms of test error and also in terms of generalization (as measured by the test-train error gap). The performance improvement of Dropout technique in terms of generalization over L_1 is statistically significant as shown by the p-value. Similarly, the performance of divReg is better than both L_2 and OrthoReg with respect to test error and generalization. Another keen observation is that the test-train error gap for divReg and L_2 regularization is very similar as inferred by the p-value, but the improvement in absolute test performance does seem to be statistically significant.

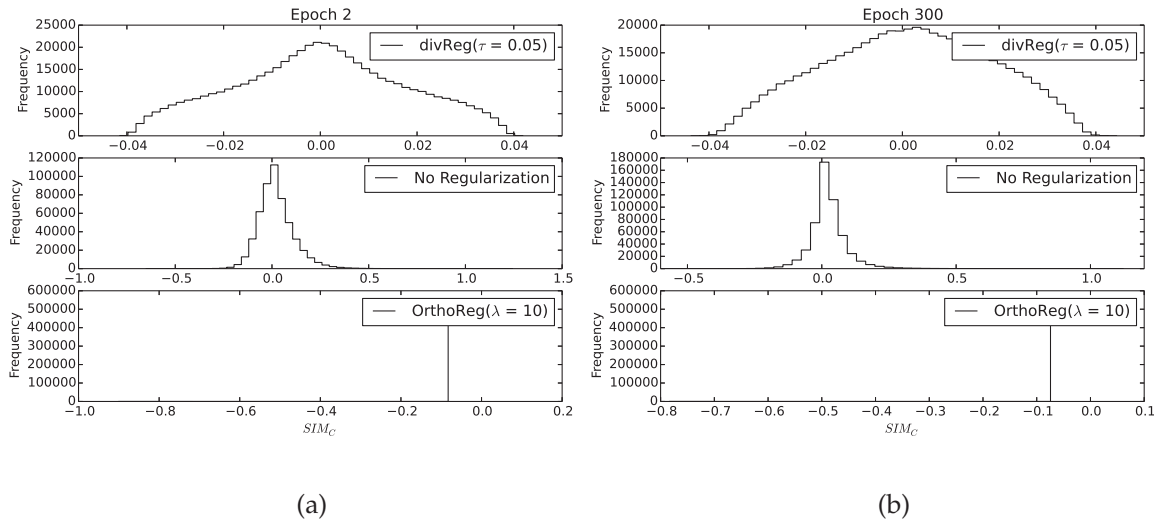


FIGURE 41: The distribution of pairwise feature correlation ($\Omega(1)$) in first hidden layer at (a) epoch 2 (b) epoch 300 [4]

For qualitative comparison, a sparse autoencoder (AE) with 256 ReLU-activated encoding units and 784 sigmoid-activated decoding units was trained on raw pixels of MNIST digits. The weights were initialized randomly by sampling from Gaussian distribution with zero mean and standard deviation of 0.003 based on

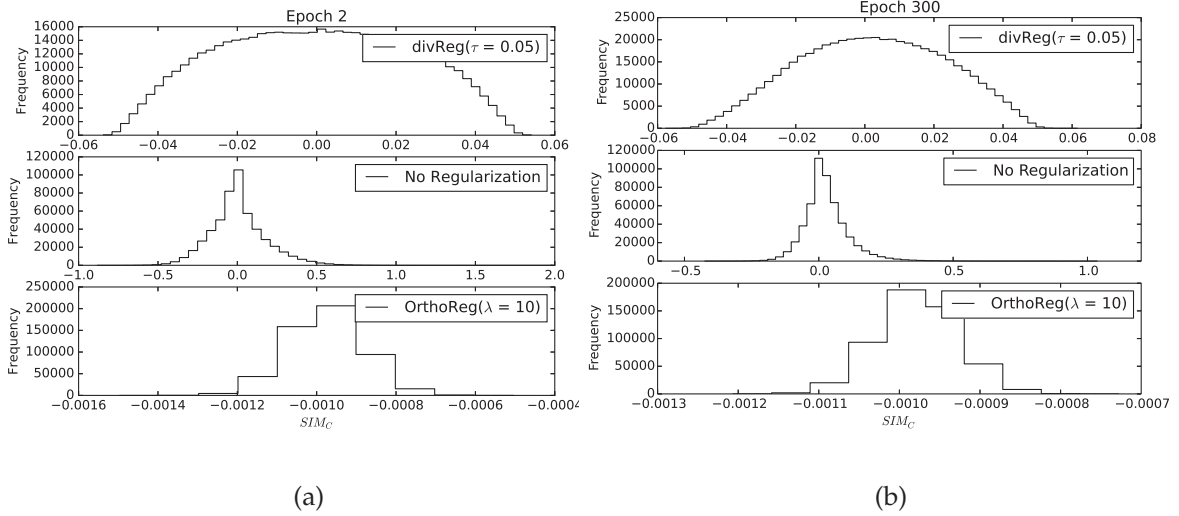


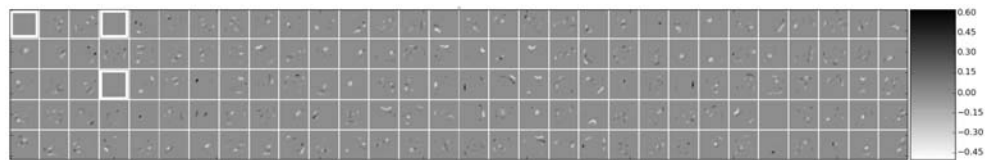
FIGURE 42: The distribution of pairwise feature correlation ($\Omega(2)$) in second hidden layer at (a) epoch 2 (b) epoch 300 [4]

[68]. The AE model was regularized with L_1 (using decay parameter 10^{-4}), dropout ($\alpha = 0.5$), OrthoReg (using angle-of-influence of 10), and divReg ($\tau=0.4$) regularization techniques and compared in terms of quality of the features learned. The features learned using each of the regularization method are shown in Fig. 25. One key observation is that L_1 and dropout regularization resulted in some dead filters as highlighted in Figs. 25a and b, whereas, the representations learned with OrthoReg looks noisy compared to those learned with divReg regularization.

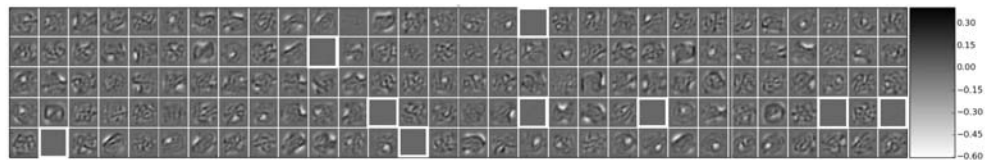
In a similar vein, multilayer perceptron was trained on MNIST with two ReLU-activated hidden layers and regularized by divReg-1. Number of hidden units per layer was set 1024 and parameters of the model was again optimized using Adam. As observed in Fig 44a, increasing τ^* yields increased number of dissimilar features because more and more features are considered occupying distinct clusters. Another interesting observation from this result is that earlier layers are generally prone to extracting more distinct features than latter layers with the same value of τ^* . Fig 44b is averaged over ten experiments to show the statistical

Regularization	train (%)	test (%)	test-train (%)	p -value
L_1	0.8791 ± 0.0947	1.9367 ± 0.0666	1.0575 ± 0.1411	0.0331
Dropout ($\alpha = 0.5$) [58]	0.4875 ± 0.0530	1.2250 ± 0.0071	0.7375 ± 0.0460	-
L_2	0.4550 ± 0.2280	1.7375 ± 0.1115	1.2825 ± 0.1505	0.0812
OrthoReg [24]	0.0167 ± 0.0212	1.6950 ± 0.0495	1.6783 ± 0.0283	0.0176
divReg	0.0535 ± 0.0658	1.3150 ± 0.0212	1.2615 ± 0.0445	-

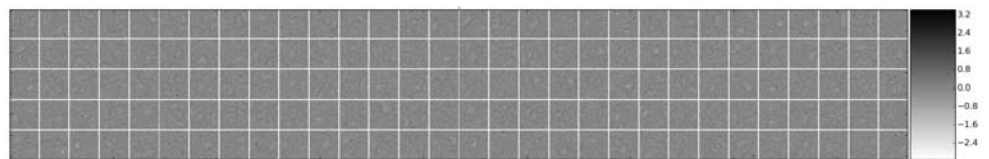
TABLE 14: Test-train error gap on MNIST [4]



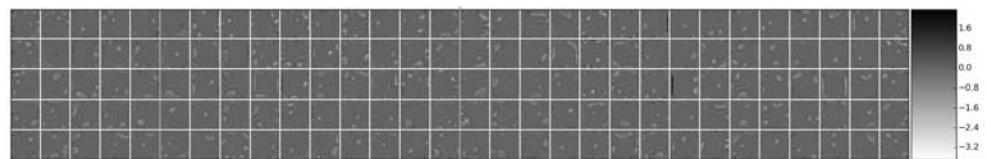
(a) L_1



(b) Dropout



(c) OrthoReg



(d) divReg

FIGURE 43: 150 out of 256 encoding features (left) learned from MNIST digit data set with autoencoders using (a) L_1 , (b) Dropout (c) orthoReg, and (d) divReg. The range of weights are scaled and mapped to the grayscale map (right) [4].

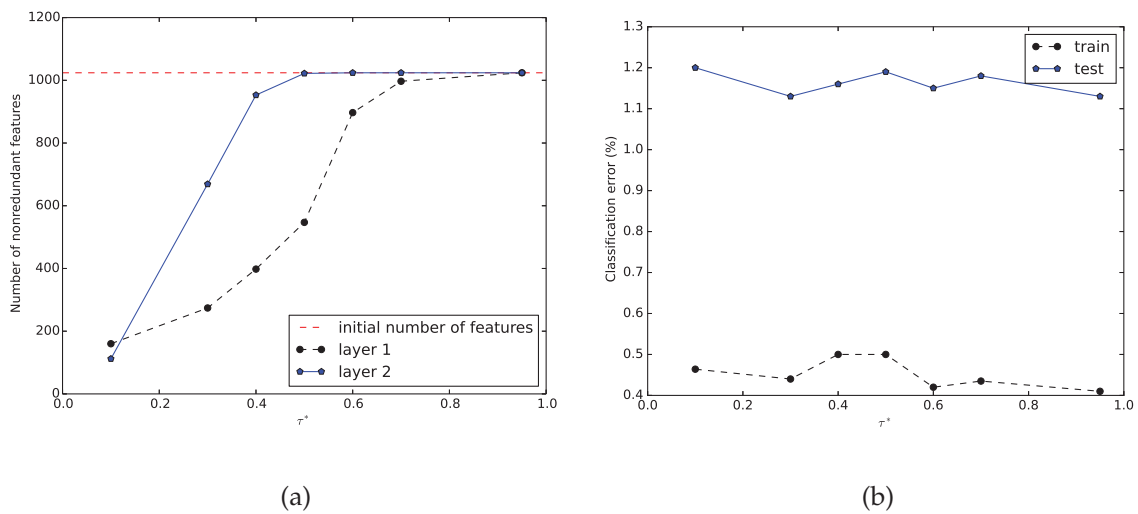


FIGURE 44: Performance of Multilayer Perceptron (with architecture 784-1024-1024-10) regularized using divReg-1 and trained on the MNIST dataset vs. threshold τ^* . (a) Number of nonredundant features for 1024 initial features. (b) percentage classification error [4]

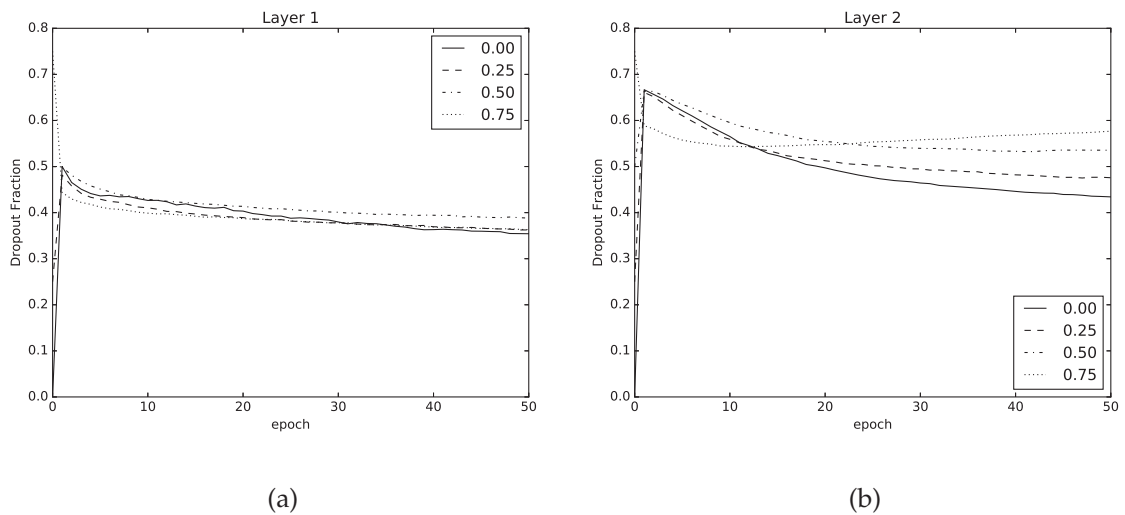


FIGURE 45: Evolution of dropout fraction (α) with divReg-2 using the MNIST dataset for four different initializations of α in (a) layer 1 and (b) layer 2. Source: [4]

significance. The error curves shown in Figs 44b reveal that networks trained using divReg-1 with $\tau^* = 0.3$ resulted in the lowest test-train error gap.

Model	# layers	size	test (%)
Unregularized [141]	2	800	1.40
DropConnect [141]	2	800	1.20
Dropout [58]	2	1024	1.28 ± 0.06
Dropout [58]	3	1024	1.25 ± 0.04
OrthoReg [24] (+ Dropout)	2	1024	1.38 ± 0.03
Label Smoothing [110] (+ Dropout)	2	1024	1.21 ± 0.06
Confidence Penalty [146] (+ Dropout)	2	1024	1.17 ± 0.06
DivReg-2	2	1024	1.15 ± 0.03
DivReg-1	2	1024	1.10 ± 0.02

TABLE 15

Test error(%) on MNIST. Source: [4]

As mentioned earlier, a crucial step in achieving good performance with divReg-2 is not only in the choice of τ^* but also in the initialization of adaptive dropout fraction α . Figs. 45a and b show the evolution of dropout fraction for four different α initializations (0.0, 0.25, 0.50, 0.75) as training progresses for first and second layers, respectively. For MNIST dataset, α initialized to 0.75 generalizes better than other initializations as shown in Fig. 46 by the test-train classification error gap. However, both the test and train accuracies are not as good as that initialized to 0.5, which has the best train and test error trade-off.

2. Diversity Regularized Image Classification

In the first set of experiments, multilayer perceptron with two ReLU-activated hidden layers and a softmax layer for classification is again tested to ascertain if

the enhanced ability to extract dissimilar features would lead to improved classification accuracy using MNIST dataset. 9000 images from MNIST data were randomly sampled from the training set as a held-out validation set for hyperparameter tuning and the network was retrained on the entire dataset using the best hyperparameter configuration. Adam optimizer with batch size of 128 was used for training the model for 300 epochs; τ and τ^* were set to 0.3 and 0.05 in divReg1 and divReg-2, respectively. The dropout fraction α in divReg-2 was initialized to 0.5. Every run of the experiment is repeated five times and averaged to combat the effect of random initialization. The classification errors of model trained with divReg were compared with state-of-the-art regularization techniques as detailed in Table 15. It is observed from the result that the model trained using divReg-1 and divReg-2 outperforms all other benchmark regularizers. The results also show that dropping the maps of redundancy filters in divReg-1 leads to a better generalization but introduces computational overhead comparable to divReg-2 with similar performance.

In the second set of large scale image classification experiments, the current state-of-the-art densely connected convolutional neural network (DenseNet) [151] was trained on CIFAR-10 dataset to see effect of extracting dissimilar features on classification performance. Again, 6000 images were randomly sampled from the training set as a held-out validation set for hyperparameter tuning and the network was retrained on the entire dataset using the best hyperparameter configuration. Due to GPU memory constraints, 100-layer DenseNet with a growth rate of 12 was used. The model was trained with stochastic gradient descent (SGD) with batch-size of 64 for 150 epochs and ζ initialized to 0.1 and scheduled to 0.01, 0.1, 0.01, 0.001, 0.01, 0.001, and 0.0001 in epochs 20, 24, 44, 84, 104, 114, and 130, respectively as shown in Fig. 47. For a fair comparison, the results presented in Table 16 are for models trained without data augmentation. Hyperparameters τ and

* τ in divReg2 was also set to 0.5 and 0.2, respectively. The dropout fraction α was initialized to 0.1. The implementation of DenseNet with bottleneck (BC) is limited to architecture with approximately 0.8M trainable parameters due to memory constraints. The classification performance of the deep networks regularized with Dropout [58], Label smoothing [110], confidence penalty [146], and OrthoReg [24] were used as benchmark. The experiments were repeated five times and averaged. It can be observed in Table 16 that divReg-2 outperforms all other regularizations considered - an indication that extraction of dissimilar features and redundancy-based adaptive dropout improve generalization of very deep neural network models.

Model	# layers	# parameters	test (%)	# epochs
Residual CNN [96]	110	1.7M	13.63	300
Stochastic Depth Residual CNN [152]	110	1.7M	11.66	500
Densely Connected CNN (with Dropout) [151]	40	1.0M	7.00	300
Densely Connected CNN (with Label Smoothing [110] + Dropout)	40	1.0M	6.89	300
Densely Connected CNN (with Confidence Penalty [146] + Dropout)	40	1.0M	6.77	300
Densely Connected CNN-BC (with Dropout) [151]	100	0.8M	6.8 (± 0.057)	150
Densely Connected CNN-BC (with OrthoReg [24] + Dropout)	100	0.8M	11.2 (± 0.125)	150
Densely Connected CNN-BC (with DivReg-2)	100	0.8M	6.3 (± 0.083)	150

TABLE 16: Test error(%) on CIFAR-10 without data augmentation. Source: [4]

Model	Top-1 (%)	Top-5 (%)	# Epochs
ResNet-34 [96]	26.77	8.56	90
ResNet-34 + OrthoReg	33.21	12.42	90
ResNet-34 + divReg	26.33	8.0	90

TABLE 17: Validation error on ImageNet. Source: [4]

In the third set of experiments involving large-scale image classification, experiments were performed on the 1000-class ImageNet 2012 dataset [153] which contains about 1.2 million training images, 50,000 validation images, and 100,000 test images (with no published labels). The results are measured by top1/top-5 error rates [153]. ImageNet dataset was used to train a residual network known as ResNet-34 [96], which has four stages of residual blocks and uses the projection shortcut when the feature maps are down-sampled. The model was trained for 90 epochs, with a batch-size of 200 and a learning rate 0.1. Each layer of the model is regularized using divReg with $\tau = 0.4$. It can be observed in Table 17 that divReg also outperforms both OrthoReg regularization method and unregularized counterpart.

3. Diversity Regularized Natural Language Inference

In the last set of experiments, the efficacy of diversity regularization in enhancing the efficiency of models used for understanding semantic relationship between two sentences and recognizing textual entailment was demonstrated. This task involves determining whether two observed sentences, first one is known as the premise and the other referred to as the hypothesis, are contradictory, not related (neutral) or entailing. In this series of experiments, models are evaluated on textual entailment recognition task using Stanford Natural Language Infer-

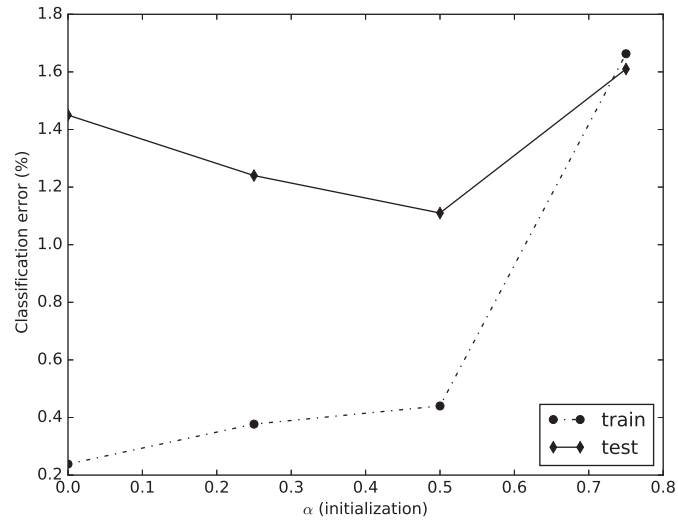


FIGURE 46–Performance evaluation using divReg-2 on MNIST dataset for four different initializations of α . Source: [4]

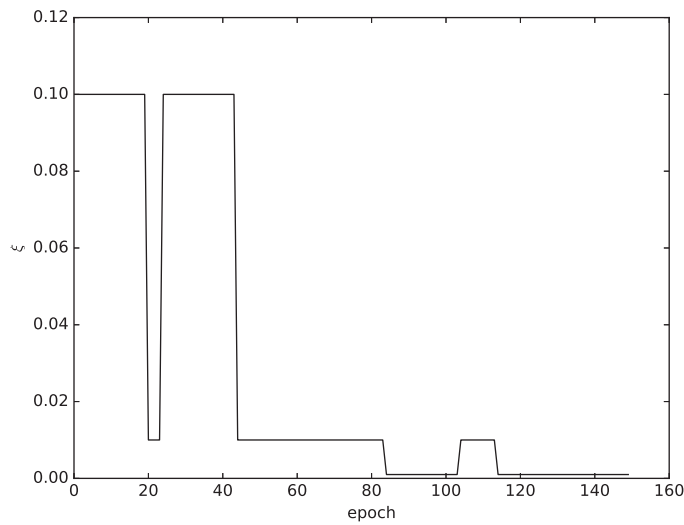


FIGURE 47–Learning rate (ζ) schedule for experiments on CIFAR-10 dataset. Source: [4]

Premise	Hypothesis	label
A soccer game with multiple males playing	Some men are playing a sport	E
A man inspects the uniform of a figure in some East Asian country	The man is sleeping	C
A smiling costumed woman is holding an umbrella	A happy woman in a fairy costume holds an umbrella	N

TABLE 18: A select examples from SNLI dataset where E, C, and N represent Entailment, Contradiction, and Neutral, respectively. Source: [4]

ence (SNLI) dataset [149]. The original dataset contains 550,152 duos of premise-hypothesis sentences and their corresponding labels as training set, 10,000 as validation set, and 10,000 as testing set. After the removal of sentence-pair with unknown labels, 549,367 pairs for training, 9,842 for validation and 9,824 for testing were obtained. Select examples from SNLI dataset are shown in Table 18.

300-dimensional word embeddings from the pretrained 300D Glove 840B vocabulary [154] was extracted from SNLI dataset, each for both the premise and hypothesis sentences and fed them through a ReLU "translation" layer. The maximum sequence length was chosen to be 42 and the embeddings of words not in the vocabulary are set to zero in accordance with [155]. The pretrained Glove embedding layer contains more than 12 million parameters, which was fixed during training to avoid overfitting [156] and computational overhead. The LSTM model with 300 hidden units was used to encode the premise and hypothesis sentences and the resulting two 300D embeddings are concatenated and fed into three layers

of fully-connected units with ReLU activations. The output of the last layer is fed into Softmax layer for classification.

The overall model was trained using the Adam optimizer with batch-size of 512 for 100 epochs while τ and τ^* in divReg-2 was also set to 0.5 and 0.2, respectively. The dropout fraction α for both recurrent (LSTM) and fully-connected layers was initialized to 0.2. The experiment was also performed by replacing LSTM with a GRU. Results were benchmarked with recent sentence encoding-based models and experimental results were illustrated in Table 19. It is remarked that the parameters of the Glove embedding layer were not included in the number of parameters computed in Table 19.

As can be observed from the results, diversity regularization and adaptive dropout significantly improved the performance of both the baseline LSTM and GRU models. By initializing dropout fraction of both recurrent and fully-connected units to 0.2, the model was able to figure out the suitable dropout fraction in accordance with differentiation of features. In addition, setting τ to 0.5 ensures no feature pair have cosine similarity greater than 0.5. Another important observation is that OrthoReg sometimes extracts noisy features in an attempt to decorrelate features, which explains why the performance of some models deteriorates. Deep Gated Attn BiLSTM (D-GAB) encoders [157] is the state-of-the-art sentence encoding-based model for SNLI dataset with test accuracy of 85.5%. However, D-GAB was not regularized using diversity regularization because it has more than 11 million parameters requiring larger memory than those compared in Table 19.

Model	# parameters	test (%)
300D LSTM (recurrent dropout) + 3 x 600D ReLU + OrthoReg	1.9 M	77.4
300D LSTM encoders [158]	3.0 M	80.60
300D LSTM (recurrent dropout) + 3 x 600D ReLU	1.9 M	82.7
300D SPINN-PI encoders [158]	3.7 M	83.2
600D (300+300) BiLSTM encoders [156]	2.0 M	83.3
300D NTI-SLSTM-LSTM encoders [155]	4.0 M	83.4
300D LSTM (recurrent dropout) + 3 x 600D ReLU + divReg2	1.9 M	83.9
300D GRU (recurrent dropout) + 3 x 600D ReLU	1.7 M	83.0
300D GRU (recurrent dropout) + 3 x 600D ReLU + OrthoReg	1.7 M	80.8
300D GRU (recurrent dropout) + 3 x 600D ReLU + divReg2	1.7 M	84.3

TABLE 19: Test accuracy (%) on SNLI dataset. Source: [4]

D. Diversity Regularized Adversarial Learning (DiReAL)

The training of GAN can be abstracted as a non-cooperative game between two players namely the generator G and discriminator D . The discriminator tries to distinguish if the generated sample is from the real (p_{data}) or fake data distribution (p_z), while G tries to trick D into believing that generated sample is from p_{data} by moving the generation manifold towards the data manifold. The discriminator aims to maximize $\mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})]$ when the input is sampled from real distribution and given a fake image sample $G(\mathbf{z})$, $\mathbf{z} \sim p_z(\mathbf{z})$, it is trained to output probability, $D(G(\mathbf{z}))$, close to zero by maximizing $\mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$. The generator network, however, is trained to maximize the chances of D producing a high probability for a fake image sample $G(\mathbf{z})$ thus by minimizing $\mathbb{E}_{\mathbf{z} \sim p_z}[\log(1 - D(G(\mathbf{z})))]$.

Since discriminator D is commonly parameterized as deep neural networks and relies on many redundant filters, it is regularized during training to provide more stable gradient to update both G and D . Diversity regularizer enforces constraints on the learning process by simply encouraging diverse filtering and discourages D from extracting redundant filters. The idea behind diversifying features is that in addition to gradient information provided by D , additional diversity loss with more stable gradient is provided to refine both G and D as shown in Fig.48. The diversity loss encourages weights of D to be diverse by pushing them towards the nearest orthogonal manifold. Proposed diversity regularization provides more efficient gradient flow, a more stable optimization, richness of layer-wise features of resulting model, and improved sample quality compared to benchmarks and baseline. The diversity regularization ensures the column space of $\Phi^{(l)}$ for l^{th} layer of the discriminator does not concentrate in few direction during training thus preventing them to be sensitive in few and limited directions.

In this experiment, a deep convolutional GAN (DCGAN) in [159] was trained

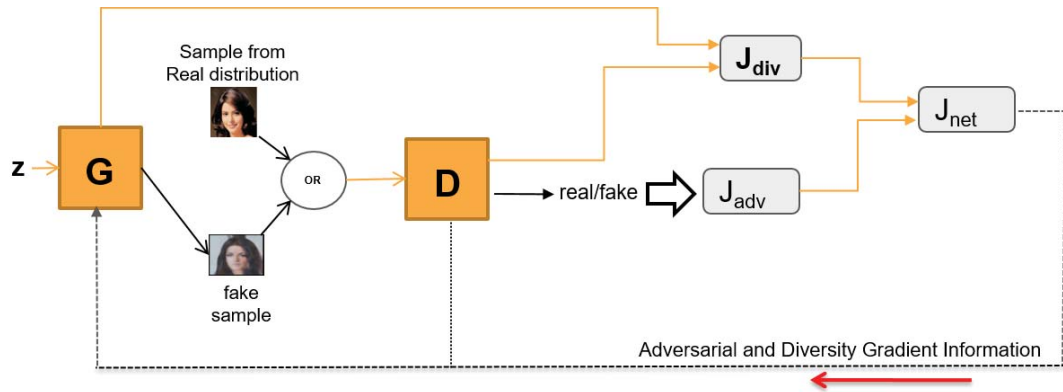


FIGURE 48 – Schema of Diversity Regularized Adversarial Learning (DiReAL)

using MNIST digits. A batch size of 64 was used to train the model for 100 epochs and τ in divReg was set to 0.5. Fig. 49 shows the diversity loss of both generator and discriminator for DiReAL and unregularized counterpart. It can be observed that divReg was able to minimize the pairwise feature correlations compared to the highly correlated features extracted by the unregularized counterpart. Specifically, divReg was able to steadily minimize the diversity loss as training progresses compared to the unregularized DCGAN, where extraction of similar features grows with epoch of training, thus increasing the diversity loss. The divergence between discriminator output for real handwritten digits and generated samples over 30 batches for regularized and the unregularized networks is shown in Fig. 50. The divergence was measured using the Wasserstein distance measure [160] and it can be observed that the regularizing effect of divReg stabilizes the adversarial training and prevents mode collapse. For unregularized network, however, the mode started to collapse around 45th epoch. Closer look into the diversity of the generator in Fig. 49a, it is evident that just around the epoch of collapse the generator starts extracting more and more redundant filters. It is suspected that divReg was able to stabilize the training by pushing features to lie close to the orthogonal manifold, thus preventing learned features from collapsing to an undesirable manifold.

Fig. 51 shows the handwritten digit samples synthesized with and without divReg and it can be observed that diversification of features is beneficial for stabilizing adversarial learning and ultimately improving the samples' quality.

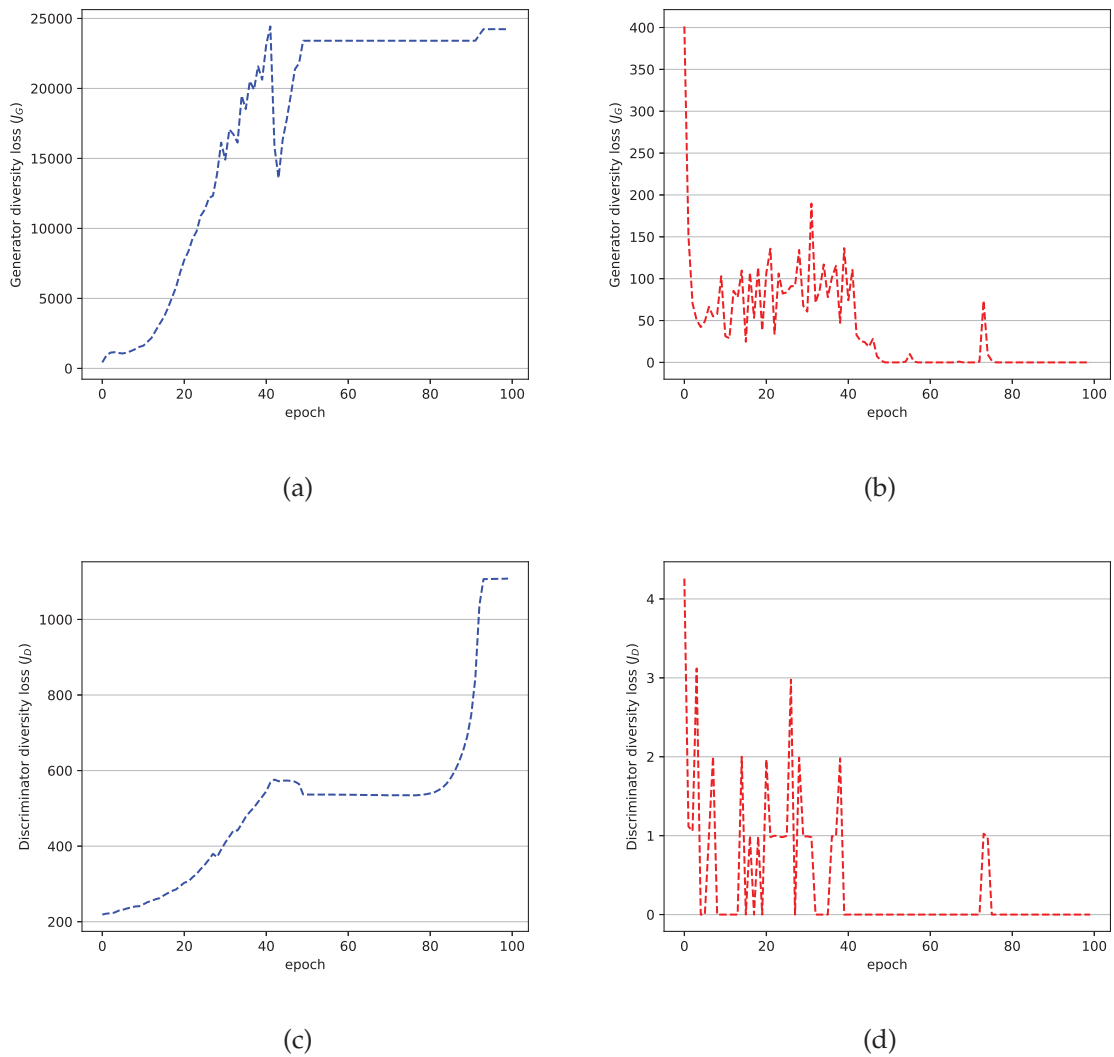


FIGURE 49: Diversity loss of (a) generator with no regularization (b) generator with diReAL (c) discriminator with no regularization, and (d) discriminator with DiReAL trained on MNIST dataset.

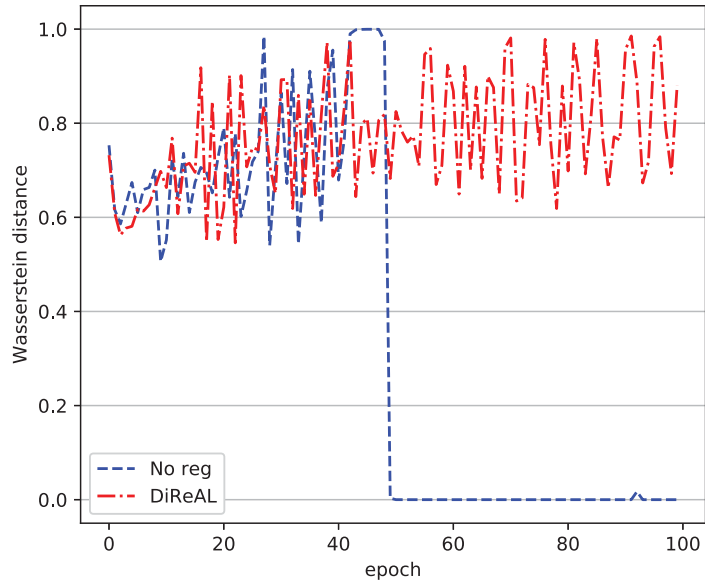


FIGURE 50 – Divergence, as measured by Wasserstein distance, between the discriminator output for real and synthesized samples

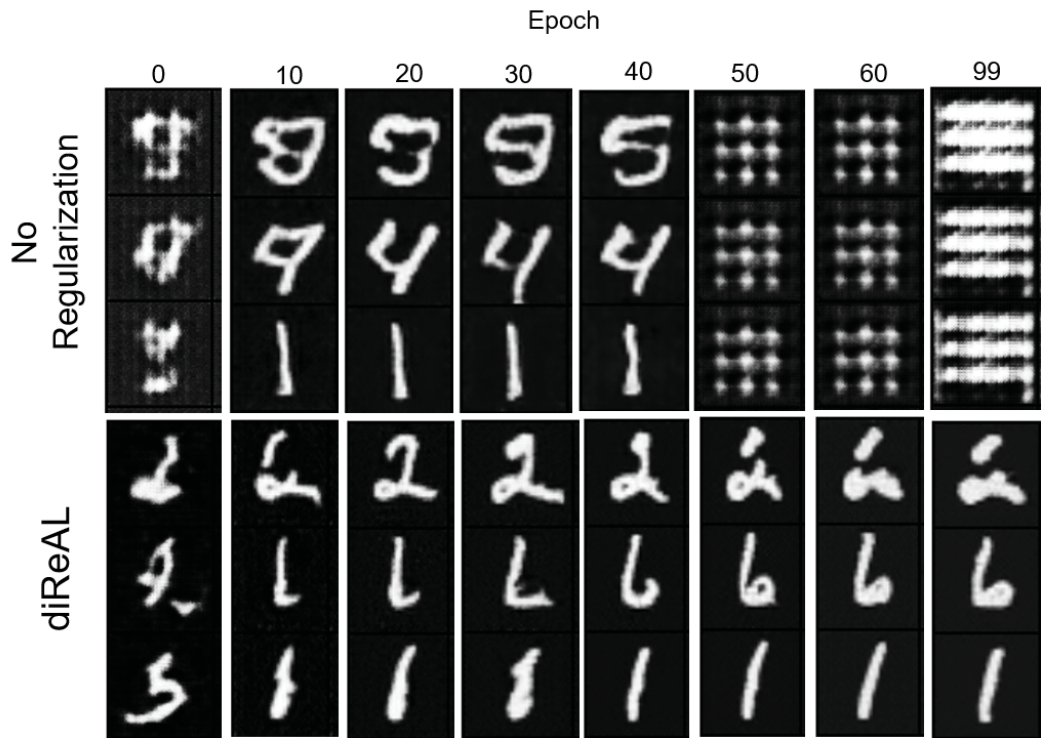


FIGURE 51 – Synthesized hand-written digits with and without diversity Regularization.

E. Conclusion

This chapter addresses the concept and properties of special regularization of deep neural networks which initially produce a variety of complex receptive fields. Proposed approach takes advantage of initially extracting diversified features and eliminating features based on select redundancy measures. The performance of the proposed regularization in terms of extracting diverse features and improving generalization was compared with recent regularization techniques on select tasks using state-of-the-art deep learning models. The results show that if not properly constrained, deep neural network models are capable of extracting very similar features thereby creating unnecessary amount of filtering redundancy. By using the proposed methods, such redundancy can be controlled, eliminated and networks are enabled to extract more distinctive features. It has also been shown on select examples that concurrent extraction of diverse features and redundant feature dropout improve model generalization. Generative models such as generative adversarial networks have also been shown to benefit from feature diversification by using stable diversity loss to stabilize adversarial learning. These concepts are illustrated using MNIST handwritten digits, CIFAR-10, Celeb-A, STL-10, ImageNet, and Stanford Natural Language Inference Dataset.

CHAPTER VI CONCLUSIONS

Receptive fields optimization (RFO) in deep neural networks for improved interpretability, performance, and computational efficiency has been detailed in this dissertation. RFO is capable of enhancing feature extraction process and unearthing most important latent representations that are useful for many discriminative and generative tasks. For discriminative purposes, optimization of receptive fields could help reveal what is important/unimportant in data and/or model for task such as classification, regression, clustering, and compression. For generative intent, RFO can help alleviate some of the problems associated with training state-of-the-art generative models for data synthesis.

The task of optimizing receptive field in deep learning is an open-ended adventure as it stands and it is often tailored towards or customized for solving specific problems. As discussed in detail in Chapter III, RFO through imposition of nonnegative receptive fields can help alleviate the difficulty in building and developing an accurate interpretable autoencoder-based deep learning models. It was shown that by imposing nonnegativity-constraints on receptive fields, only few important negative weights for retaining model's performance are preserved and reduced in magnitude. This enables the extraction of additive part-based data decomposition.

Two offline RFO methods were proposed in Chapter IV for reducing redundant receptive fields in unsupervised artificial neural network known as autoencoder. The proposed methods show that redundancy can be drastically reduced even when autoencoders are cascaded into deep networks. It was also shown that

removing redundant receptive fields improves the computational efficiency of the unsupervised feature extraction and reduces the effect of overfitting in the supervised phase. Post-training RFO was also detailed and extended in Chapter V into a family of deep convolutional neural networks for improving computational efficiency through network compression with minimal accuracy deterioration.

Finally, an online RFO method for preventing redundancy and imposing feature diversity during training was presented and discussed in detail in chapter VI. The regularization mechanism proposed inhibits the learning of redundant filters, thereby enforcing the extraction of diverse features. Additionally, hierarchical agglomerative clustering was adapted to drop activations (or feature maps) of redundant features during training for adapting the dropout fraction. Since agglomerative clustering is computationally expensive, a novel method was proposed based on the pairwise feature correlation to eliminate the computational overhead resulting from agglomerative clustering of features. This proposed method uses pairwise feature correlation to compute adaptive dropout fraction during training. The effectiveness of the algorithms were demonstrated across many learning tasks and benchmark datasets and shown to improve the state-of-the-art.

RFO methods detailed in this dissertation have many practical implications. First, one of the main obstacles limiting the application of deep neural networks in medicine, military, and business analytics is the fact that its resulting models are not sufficiently transparent or interpretable. This dissertation in part alleviates these limitations by instilling power of explanation/interpretations into resulting deep learning model. Second, it focuses on improving the computational efficiency of deep neural network for both supervised and unsupervised settings. It enables the use of accurate deep neural network models on computationally limited platforms such as mobile and embedded devices. In addition to good accuracy, the work in Chapters IV and V enables resource-limited devices to benefit greatly from

accurate deep neural network models with lower inference computational cost. It enables large-scale DNNs models to execute efficiently offline on mobile devices and medical wearables without the need for the conventional cloud-based solutions.

In sum, algorithms introduced in this dissertation are particularly valuable to so many aforementioned contemporary machine learning applications requiring the use of intelligent and computationally efficient models interpretable by domain users. The original contributions of the author of this dissertation include:

- The introduction of regularization method that balances the notion of interpretability and accuracy in deep autoencoding neural networks with induction of simultaneous sparsity and nonnegativity constraints.
- Analysis and effect of redundant receptive fields on performance, computational efficiency and interpretability of supervised and unsupervised deep neural network models.
- The introduction of two algorithms based on agglomerative clustering to automatically detect and eliminate redundancy in unsupervised deep autoencoding and supervised deep convolutional neural networks.
- The introduction of a novel online diversity regularization technique to inhibit the learning of redundant receptive fields for the purpose of enhancing the efficiency of deep learning models and for stabilizing the training of generative adversarial networks.
- The introduction of a redundancy-feature-based adaptive dropout technique to reduce the effect of overfitting in deep neural network models.

REFERENCES

- [1] B. O. Ayinde and J. M. Zurada. Deep learning of constrained autoencoders for enhanced understanding of data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(9):3969–3979, Sep. 2018.
- [2] Babajide O Ayinde and Jacek M Zurada. Nonredundant sparse feature extraction using autoencoders with receptive fields clustering. *Neural Networks*, 93:99–109, 2017.
- [3] Babajide O Ayinde and Jacek M Zurada. Building efficient convnets using redundant feature pruning. *arXiv preprint arXiv:1802.07653*, 2018.
- [4] B. O. Ayinde, T. Inanc, and J. M. Zurada. Regularizing deep neural networks by enhancing diversity in feature extraction. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–12, 2019.
- [5] Babajide O Ayinde and Jacek M Zurada. Discovery through constraints: Imposing constraints on autoencoders for data representation and dictionary learning. *IEEE Systems, Man, and Cybernetics Magazine*, 3(3):13–24, 2017.
- [6] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(11), 2008.
- [7] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
- [8] Shuyang Wang, Zhengming Ding, and Yun Fu. Feature selection guided auto-encoder. In *AAAI*, pages 2725–2731, 2017.

- [9] Babajide O Ayinde and Hashim A Hashim. Energy-efficient deployment of relay nodes in wireless sensor networks using evolutionary techniques. *International Journal of Wireless Information Networks*, pages 1–16, 2018.
- [10] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.
- [11] Abdel-rahman Mohamed, George E Dahl, Geoffrey Hinton, et al. Acoustic modeling using deep belief networks. *IEEE Trans. Audio, Speech & Language Processing*, 20(1):14–22, 2012.
- [12] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10, 1994.
- [13] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.
- [14] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1. *Vision Research*, 37(23):3311–3325, 1997.
- [15] Douglas Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, pages 827–832, 2015.
- [16] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

- [17] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [18] J. M. Zurada. *Introduction to Artificial Neural Systems*. West Publishing Co., St. Paul, MN, USA, 1992.
- [19] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.
- [20] Jan Chorowski. Learning understandable classifier models. *PhD Dissertation*, 2012.
- [21] Jan Chorowski and Jacek M Zurada. Extracting rules from neural networks as decision diagrams. *IEEE Transactions on Neural Networks*, 22(12):2435–2446, 2011.
- [22] Jan Chorowski and Jacek M Zurada. Learning understandable neural networks with nonnegative weight constraints. *IEEE transactions on Neural Networks and Learning Systems*, 26(1):62–69, 2015.
- [23] Yoshua Bengio and James S Bergstra. Slow, decorrelated features for pre-training complex cell-like networks. In *Advances in Neural Information Processing Systems*, pages 99–107, 2009.
- [24] Pau Rodríguez, Jordi González, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*, 2016.
- [25] Jasper Snoek, Ryan P Adams, and Hugo Larochelle. Nonparametric guidance of autoencoder representations using label information. *Journal of Machine Learning Research*, 13(Sep):2567–2588, 2012.

- [26] Koray Kavukcuoglu, Rob Fergus, Yann LeCun, et al. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1605–1612. IEEE, 2009.
- [27] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007.
- [28] Joseph F Murray and Kenneth Kreutz-Delgado. Learning sparse overcomplete codes for images. *The Journal of VLSI Signal Processing*, 45(1):97–110, 2006.
- [29] Babajide O Ayinde and Ahmed H Desoky. Lossless image compression using zipper transformation. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCVC)*, page 103. The Steering Committee of The World Congress in Computer Science, 2016.
- [30] Babajide O Ayinde. A fast and efficient near-lossless image compression using zipper transformation. *arXiv preprint arXiv:1710.02907*, 2017.
- [31] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696. ACM, 2009.
- [32] Jan Chorowski. Review of dimensionality reduction techniques. *Technical Paper (Internal Report)*, 2010.
- [33] Christopher J Rozell, Don H Johnson, Richard G Baraniuk, and Bruno A Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural computation*, 20(10):2526–2563, 2008.

- [34] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
- [35] Nicolas Gillis. The why and how of nonnegative matrix factorization. *Regularization, Optimization, Kernels, and Support Vector Machines*, 12(257), 2014.
- [36] Karthik Devarajan. Nonnegative matrix factorization: an analytical and interpretive tool in computational biology. *PLoS computational biology*, 4(7):e1000029, 2008.
- [37] Oyetunji E Ogundijo and Xiaodong Wang. A sequential monte carlo approach to gene expression deconvolution. *PloS one*, 12(10):e0186167, 2017.
- [38] Oyetunji E Ogundijo and Xiaodong Wang. Characterization of tumor heterogeneity by latent haplotypes: a sequential monte carlo approach. *PeerJ*, 6:e4838, 2018.
- [39] Oyetunji E Ogundijo and Xiaodong Wang. Bayesian estimation of scaled mutation rate under the coalescent: a sequential monte carlo approach. *BMC bioinformatics*, 18(1):541, 2017.
- [40] Oyetunji E Ogundijo, Abdulkadir Elmas, and Xiaodong Wang. Reverse engineering gene regulatory networks from measurement with missing values. *EURASIP Journal on Bioinformatics and Systems Biology*, 2017(1):2, 2016.
- [41] Babajide O Ayinde, Sami El Ferik, Salim Ibrir, Moez Feki, and Bilal A Siddiqui. Backstepping control of an electro-hydraulic servo system subject to disturbance and parameter uncertainty. In *GCC Conference and Exhibition (GCCCE), 2015 IEEE 8th*, pages 1–6. IEEE, 2015.

- [42] Tsung-Han Chan, Wing-Kin Ma, Chong-Yung Chi, and Yue Wang. A convex analysis framework for blind separation of non-negative sources. *IEEE Transactions on Signal Processing*, 56(10):5120–5134, 2008.
- [43] Fariar Shahnaz, Michael W Berry, V Paul Pauca, and Robert J Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006.
- [44] Babajide Odunitan Ayinde and Abdulaziz Y Barnawi. Differential evolution based deployment of wireless sensor networks. In *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference on*, pages 131–137. IEEE, 2014.
- [45] Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. Community discovery using nonnegative matrix factorization. *Data Mining and Knowledge Discovery*, 22(3):493–521, 2011.
- [46] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2011.
- [47] Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.
- [48] Jing Wang, Haibo He, and Danil V Prokhorov. A folded neural network autoencoder for dimensionality reduction. *Procedia Computer Science*, 13:120–127, 2012.
- [49] Omar Dekhil, Hassan Hajjdiab, Babajide Ayinde, Ahmed Shalaby, Andy Switala, Dawn Sosnin, Aliaa Elshamekh, Mohamed Ghazal, Robert Keynton, Gregory Barnes, et al. Using resting state functional mri to build a personalized autism diagnosis system. In *Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on*, pages 1381–1385. IEEE, 2018.

- [50] A. Ng. Sparse autoencoder. In *CS294A Lecture notes*, URL https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf, 2011. Stanford University.
- [51] E. Hosseini-Asl, J. M. Zurada, and O. Nasraoui. Deep learning of part-based representation of data using sparse autoencoders with nonnegativity constraints. *Neural Networks and Learning Systems, IEEE Transactions on*, 27(12):2486–2498, 2016.
- [52] Babajide O Ayinde, Ehsan Hosseini-Asl, and Jacek M Zurada. Visualizing and understanding nonnegativity constrained sparse autoencoder in deep learning. In *Rutkowski L., Korytkowski M., Scherer R., Tadeusiewicz R., Zadeh L., Zurada J. (eds) Artificial Intelligence and Soft Computing. ICAISC 2016. Lecture Notes in Computer Science, vol 9692*, pages 3–14. Springer, 2016.
- [53] H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area v2. *Advances in Neural Information Processing Systems*, 7:873–830, 2007.
- [54] V. Nair and G. E. Hinton. 3d object recognition with deep belief nets. *Advances in Neural Information Processing Systems*, pages 1339–1347, 2009.
- [55] G. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.
- [56] C. Poultney, S. Chopra, and Y. Cun. Efficient learning of sparse representations with an energy-based model. *Advances in Neural Information Processing Systems*, pages 1137–1144, 2006.
- [57] J Moody, S Hanson, Anders Krogh, and John A Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4:950–957, 1995.

- [58] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [59] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.
- [60] Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, pages 2791–2799, 2015.
- [61] A. Lemme, R. Reinhart, and J. Steil. Online learning and generalization of parts-based image representations by non-negative sparse autoencoders. *Neural Networks*, 33:194–203, 2012.
- [62] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.
- [63] Włodzisław Duch, Rudy Setiono, and Jacek M Zurada. Computational intelligence methods for rule-based data understanding. *Proceedings of the IEEE*, 92(5):771–805, 2004.
- [64] Bart Baesens, Rudy Setiono, Christophe Mues, and Jan Vanthienen. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management science*, 49(3):312–329, 2003.
- [65] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.
- [66] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

- [67] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.
- [68] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.
- [69] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.
- [70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1026–1034, 2015.
- [71] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.
- [72] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- [73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [74] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Pro-*

- ceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.
- [75] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *25th International Conference on Machine learning*, pages 1096–1103. ACM, 2008.
- [76] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–97. IEEE, 2004.
- [77] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.
- [78] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.
- [79] Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. *Large-Scale Kernel Machines*, 34(1):1–41, 2007.
- [80] L. Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3:e2, 2014.
- [81] S. Bengio, L. Deng, H. Larochelle, H. Lee, and R. Salakhutdinov. Guest editors introduction: Special section on learning deep architectures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1795–1797, 2013.
- [82] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8).

- [83] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.
- [84] Babajide O Ayinde and Jacek M Zurada. Clustering of receptive fields in autoencoders. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 1310–1317. IEEE, 2016.
- [85] Jiquan Ngiam, Zhenghao Chen, Sonia A Bhaskar, Pang W Koh, and Andrew Y Ng. Sparse filtering. In *Advances in Neural Information Processing Systems*, pages 1125–1133, 2011.
- [86] Jun Li, Heyou Chang, and Jian Yang. Sparse deep stacking network for image classification. *arXiv preprint arXiv:1501.00777*, 2015.
- [87] Yunlong He, Koray Kavukcuoglu, Yun Wang, Arthur Szlam, and Yanjun Qi. Unsupervised feature learning by deep sparse coding. *arXiv preprint arXiv:1312.5783*, 2013.
- [88] Bruce Walter, Kavita Bala, Milind Kulkarni, and Keshav Pingali. Fast agglomerative clustering for rendering. In *IEEE Symposium on Interactive Ray Tracing*, pages 81–86. IEEE, 2008.
- [89] Chris Ding and Xiaofeng He. Cluster merging and splitting in hierarchical clustering algorithms. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 139–146. IEEE, 2002.
- [90] Bastian Leibe, Ales Leonardis, and Bernt Schiele. Combined object categorization and segmentation with an implicit shape model. In *Workshop on Statistical Learning in Computer Vision, ECCV*, volume 2, page 7, 2004.

- [91] Swami Manickam, Scott D Roth, and Thomas Bushman. Intelligent and optimal normalized correlation for high-speed pattern matching. *Datacube Technical Paper*, 2000.
- [92] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.
- [93] Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.
- [94] Deng Cai, Xiaofei He, Yuxiao Hu, Jiawei Han, and Thomas Huang. Learning a spatially smooth subspace for face recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition Machine Learning (CVPR'07)*, 2007.
- [95] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.
- [96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [97] Islam Reda, Mohammed Ghazal, Ahmed Shalaby, Mohammed Elmogy, Ahmed AbouEl-Fetouh, Babajide O Ayinde, Mohamed AbouEl-Ghar, Adel Elmaghraby, Robert Keynton, and Ayman El-Baz. A novel adcs-based cnn classification system for precise diagnosis of prostate cancer. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 3923–3928. IEEE, 2018.
- [98] Islam Reda, Babajide O Ayinde, Mohammed Elmogy, Ahmed Shalaby, Moumen El-Melegy, Mohamed Abou El-Ghar, Ahmed Abou El-fetouh, Mo-

- ammed Ghazal, and Ayman El-Baz. A new cnn-based system for early diagnosis of prostate cancer. In *Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on*, pages 207–210. IEEE, 2018.
- [99] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.
- [100] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proc. of the 31st International Conference on Machine Learning*, pages 1764–1772, 2014.
- [101] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- [102] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.
- [103] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.
- [104] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

- [105] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.
- [106] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.
- [107] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *ICLR*, 2017.
- [108] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
- [109] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, pages 1–12, 2017.
- [110] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.
- [111] Jaehong Yoon and Sung Ju Hwang. Combined group and exclusive sparsity for deep neural networks. In *International Conference on Machine Learning*, pages 3958–3966, 2017.
- [112] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *ICLR*, 2016.

- [113] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.
- [114] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.
- [115] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.
- [116] Zelda Mariet and Suvrit Sra. Diversity networks. *ICLR*, 2016.
- [117] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *ICLR*, 2016.
- [118] Adam Polyak and Lior Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.
- [119] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *ICLR*, 2017.
- [120] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.
- [121] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

- [122] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.
- [123] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.
- [124] Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. *arXiv preprint arXiv:1605.06489*, 2016.
- [125] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016.
- [126] Qiangui Huang, Kevin Zhou, Suyu You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *Applications of Computer Vision (WACV), 2018 IEEE Winter Conference on*, pages 709–718. IEEE, 2018.
- [127] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.
- [128] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. *CVPR*, 2018.
- [129] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. Online accessed: 01-04-2018.

- [130] Yann LeCun. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [131] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [132] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- [133] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.
- [134] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.
- [135] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2755–2763. IEEE, 2017.
- [136] Bin Dai, Chen Zhu, and David Wipf. Compressing neural networks using the variational information bottleneck. *ICML*, 2018.
- [137] Pau Rodríguez, Jordi González, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. *ICLR*, 2017.
- [138] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

- [139] Dmytro Mishkin and Jiri Matas. All you need is a good init. *International Conference on Learning Representations*, 2016.
- [140] Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.
- [141] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proc. of the 30th International Conference on Machine Learning*, pages 1058–1066, 2013.
- [142] Yebo Bao, Hui Jiang, Lirong Dai, and Cong Liu. Incoherent training of deep neural networks to de-correlate bottleneck features for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6980–6984, 2013.
- [143] David Miller, Ajit V Rao, Kenneth Rose, and Allen Gersho. A global optimization technique for statistical classifier design. *IEEE Transactions on Signal Processing*, 44(12):3108–3122, 1996.
- [144] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [145] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing cnn on the loss layer. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4753–4762, 2016.
- [146] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

- [147] Soufiane Belharbi, Clement Chatelain, Romain Herault, and Sebastien Adam. Neural networks regularization through invariant features learning. *arXiv preprint arXiv:1709.01867*, 2017.
- [148] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. *ICLR*, 2017.
- [149] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.
- [150] Martín Abadi, Ashish Agarwal, Paul Barham, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [151] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.
- [152] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.
- [153] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [154] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

- [155] Tsendsuren Munkhdalai and Hong Yu. Neural semantic encoders. *arXiv preprint arXiv:1607.04315*, 2016.
- [156] Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. Learning natural language inference using bidirectional lstm model and inner-attention. *arXiv preprint arXiv:1605.09090*, 2016.
- [157] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Recurrent neural network-based sentence encoder with gated attention for natural language inference. *arXiv preprint arXiv:1708.01353*, 2017.
- [158] Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.
- [159] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
- [160] SS Vallender. Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786, 1974.

CURRICULUM VITAE

NAME: Babajide O. Ayinde

ADDRESS: 6703 Strawberry Ln, Apt. 216
Louisville, KY, 40214

EDUCATION & TRAINING:

B.S., Electronic and Electrical Engineering
Obafemi Awolowo University
2005 - 2010

M.S., Systems and Control Engineering
King Fahd University of Petroleum and Minerals
2013 - 2015

Ph.D., Electrical and Computer Engineering
University of Louisville
2015 - 2019

Data Science Summer Internship
State Farm, Bloomington IL
2017 (12 weeks)

Machine Learning Research Internship
Toyota InfoTechnology Center, Mountain View
2018 (13 weeks)

TEACHING: Computational Intelligence- Data Analysis Lab

AWARDS: Doctoral Dissertation Completion Fellowship
State Farm Hackathon Winner (2017)
State Farm Top-3 Innovation Idea Challenge Winner (2017)
University of Louisville Doctoral Fellowship
King Fahd Uni. of Pet. & Minerals Graduate Fellowship

PROFESSIONAL

SOCIETIES: Institute of Electrical and Electronic Engineers
National Society of Black Engineers

PUBLICATIONS

REFEREED JOURNALS

B.O. Ayinde and J. M. Zurada," Nonredundant Sparse Feature Extraction using Autoencoders with Receptive Fields Clustering", *Neural Networks*, Vol. 7, pp. 99-109, September, 2017

B.O. Ayinde and J. M. Zurada," Constrained Autoencoders for Data Representation and Dictionary Learning", *IEEE Systems, Man, and Cybernetics Magazine*, Vol. 3, Issue 3, pp. 13-24, July 2017

B.O. Ayinde and J. M. Zurada," Deep Learning of Constrained Autoencoders for Enhanced Understanding of Data" in *IEEE Trans. on Neural Networks and Learning Systems*, September 2018, Vol. 29, Issue 9, Pg. 3969 - 3979.

B.O. Ayinde, T. Inanc, and J. M. Zurada, "Regularizing Deep Neural Net-

works by Enhancing Diversity in Feature Extraction", in IEEE Trans. on Neural Networks and Learning Systems, January 2019, DOI: 10.1109/TNNLS.2018.2885972

B.O. Ayinde, T. Inanc, and J. M. Zurada, "Redundant Feature Pruning for Accelerating Trained Deep Neural Networks", Neural Networks, submitted in December, 2018 (in review)

Hashim, Hashim A., Babajide Odunitan Ayinde, and Mohamed A. Abido. "Optimal placement of relay nodes in wireless sensor network using artificial bee colony algorithm." Journal of Network and Computer Applications 64 (2016): 239-248.

Ayinde, Babajide O., and Hashim A. Hashim. "Energy-Efficient Deployment of Relay Nodes in Wireless Sensor Networks Using Evolutionary Techniques" International Journal of Wireless Information Networks, Volume 25, Issue 2, pp. 157-172, 2018.

CONFERENCE PAPERS

B. O. Ayinde and J. M. Zurada (2016, July). Clustering of receptive fields in autoencoders. In Neural Networks (IJCNN), 2016 International Joint Conference on (pp. 1310-1317). IEEE.

B. O. Ayinde, E. Hosseini-Asl, and J. M. Zurada (2016, June). Visualizing and understanding nonnegativity constrained sparse autoencoder in deep learning. In International Conference on Artificial Intelligence and Soft Computing (pp. 3-14). Springer, Cham.

B.O. Ayinde, T. Inanc, and J. M. Zurada, "On Correlation of Features Extracted by Deep Neural Networks", submitted on December 25, 2018 to 2019

International Joint Conference on Neural Networks.

B.O. Ayinde, R. Guo, H. Sun and K. Oguchi, " Efficient Shadow Detection and Removal using Synthetic Data with Domain Adaptation", submitted to CVPR, June 2019, Los Alamitos, CA, USA.

R. Guo, B.O. Ayinde, H. Sun and K. Oguchi, "Monocular Depth Estimation Using Synthetic Images with Realistic Constraint*", submitted to International Conference on Robotics and Automation (ICRA), 2019.

B. O. Ayinde and A. H. Desoky (2016, January). Lossless Image Compression using Zipper Transformation. In Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV) (p. 103). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

B. O. Ayinde, S. El-Ferik, S. Ibrir, M. Feki, and B. A. Siddiqui (2015). "Backstepping control of an electro hydraulic servo system subject to disturbance and parameter uncertainty", IEEE 8th GCC Conference and Exhibition, (pp. 1-6)

S. El Ferik, B. O. Ayinde, S. Ibrir, and M. Feki. "Backstepping-based output feedback control of an electro-hydraulic servo system." In Systems, Signals and Devices (SSD), 2015 12th International Multi-Conference on, pp. 1-6. IEEE, 2015.