REGULARIZED AUTOENCODERS

By

Kavya Gupta

Under the Supervision of

Dr. Angshul Majumdar

Indraprastha Institute of Information Technology, Delhi

October 2016

REGULARIZED AUTOENCODERS

By

Kavya Gupta

Under the Supervision of

Dr. Angshul Majumdar

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology

To

Indraprastha Institute of Information Technology, Delhi

October 2016

Table of Contents

Certificate

This is to certify that the thesis titled "Regularized Autoencoders" being submitted by Kavya Gupta to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

October 2016

Dr. Angshul Majumdar
Department of Electronics and Communication
Indraprastha Institute of Information Technology Delhi
New Delhi 110020

Acknowledgement

I would like to express my greatest gratitude to my supervisor Dr. Angshul Majumdar for providing invaluable guidance and encouragement at every step. These few words can't express my high regards and respect for him. He has been very supportive and his faith kept me going. He has constantly inspired me to do good work and have diligence in one's work.

I wish to express my gratitude to all my professors for the wonderful courses they taught and to IIITD in general for providing such a great environment to work. Very special thanks to Dr. Pankaj Jalote for making IIITD such a wonderful place for research in India.

I would like to thank all the members of SALSA lab for their support at different stages of my stay at IIITD. Valuable discussions, suggestions, and reviews helped me improve my work. And also I would like to thank my friends at IIITD who have supported me throughout my stay here and made it productive, fun and a wonderful experience. Every experience at IIITD is precious and helped me grow as a person.

Above all, I would like to thanks my parents and my brother for their non-stopping love, support, encouragement and positivity. This accomplishment would not have been possible without them.

Abstract

Autoencoders are Neural Networks trained in order to map input to its output. Autoencoders are designed to enable the network to copy input to output as close as possible. Following this, the network will be able to learn useful properties in the data and learns a representation of the data, which might be helpful in Classification, image recovery or any application where good feature set is detrimental. Autoencoders have been used for dimensionality reduction and feature learning and lately they are also being ventured into the field of generative modeling.

Copying the input to the output might seem trivial, but we are not typically concerned about the result of the decoder but the intrinsic representation of data which should be able to capture its salient features. In order to achieve this, different constraints can be enforced on the network learning of the autoencoders.

In this work we are interested in adding regularization terms to the basic Euclidean distance data fidelity term in order to get more defined and structured feature set which will further help in the task at hand. In other words, *Regularized Autoencoders* uses a cost function that not only copy input to output but also encourages the network to have other properties like sparsity, rank deficiency etc. and accordingly helps to regulate the capacity of the network to learn. Feature learning has become a norm for most applications and extracting good learned features have always been the ultimate aim of Machine Learning techniques. The main contribution of this work is to provide simple yet effective Machine Learning networks which can be widely used for a variety of applications and datasets.

This work explores regularization constraints on Undercomplete Autoencoders in two parts. The first part focuses on Modeling Redundancy in the network exploring the Sparsity and rank deficiency. Sparsity helps in learning (keeping) the important connections while trimming the irrelevant ones where as rank deficiency encourages linear dependency in the connections. The ensuing formulations are solved using Majorization-Minimization technique. The second part explores Modeling of Similarity in features using rank deficiency within classes which encourages linear dependency in the features belonging to the same class. For this part, Split Bregman technique is employed to solve the proposed formulation.

The performance of our methods is tested on two tasks – classification and Denoising. Thorough experiments showed that our proposed methods yield considerable better results for Gaussian denoising. For classification, we have opted a way of Deep learning and formed Stacked Autoencoders which significantly improves the classification results. The results are even better than highly tuned existing Deep Learning tools such as SDAEs and DBNs and the versatility of network is tested for different datasets. Our methods are also computationally simpler as compared to existing state-of-the-art tools which have enormous training times and requires a huge amount of data.

List of figures

List of Tables

CHAPTER 1

1. Introduction

Autoencoders are deterministic and unsupervised training models that learn a mapping from a dataset to itself. The encoding weights map the data into a latent space and decoding weights map the latent representation (generally a compressed representation) back to the original space – the weights are learned such that the reconstruction error is minimized.

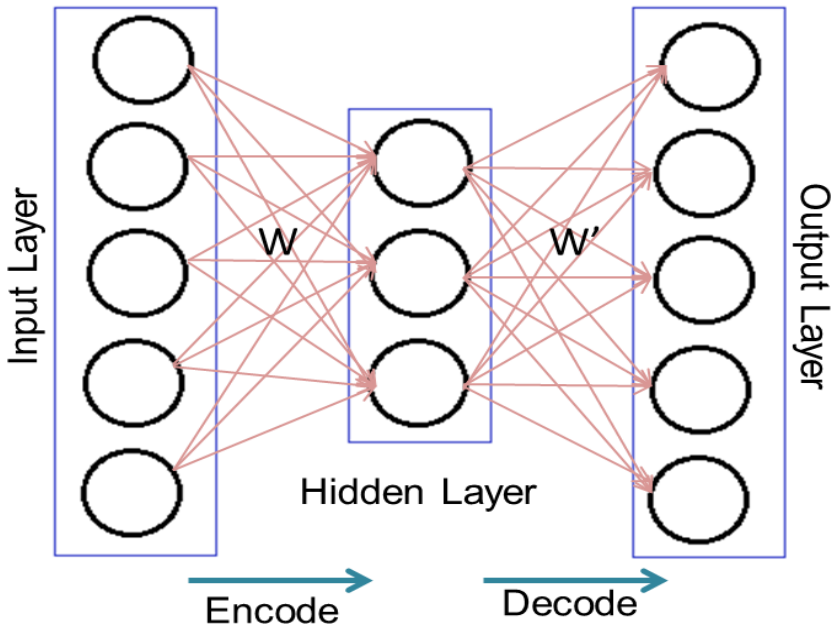


Figure 1.1: Basic Autoencoder Structure

Basically, autoencoders are approximators for the identity operation; therefore learning these weights might seem trivial; but by constraining the parameters (such as number of nodes or number of connections), interesting representations can be uncovered in the data. Most real datasets are structured i.e. they have a high degree of local correlations; usually, the autoencoder is able to exploit these correlations and yield compressed representations. However autoencoders are not usually used for compression, rather they are used for learning the representations which are later used for classification i.e. for feature learning. In the past, these structures have paved way for compression of natural signals like image, speech, and

video. Modeling these structures has also benefitted machine learning, e.g. the success of PCA in classification is a well-known example. Feature representation is one of the key factors in any machine learning problem; the goal is to identify features so that they best represent the data (in some sense) – autoencoders learn features so that the Euclidean norm is minimized.

Autoencoders can have a larger number of nodes in the representation layer than those at the input / output. It is believed that such networks can learn non-linear relationships in the data; however learning so many parameters requires a large amount of training data, which is not always available. This leads to overfitting. In recent times, instead of architecture with a large number of nodes in a single hidden layer, researchers are going deeper. Deep architectures too have a large number of parameters to learn, but the advantage is that the parameters can be learned layer-wise [1]; hence can be learned with limited training samples. It is well known how successful deep architectures have been for a variety of problems arising in image classification [2], [3] and speech processing [4], [5]. Proponents of deep learning believe that going deeper into the architecture captures more abstract and difficult representations and hence yield lower classification errors [6], [7], [8] and [9].

A basic Autoencoder learns the encoding and decoding weights by minimizing the l_2–norm between the input (training samples) and the output (training samples /corrupted training samples). Over time several variations have been proposed. One of the simpler variations is the denoising autoencoder [6], where the inputs are corrupted and the outputs are clean; the autoencoder basically learns to clean corrupted samples. In [6], it was empirically shown that such denoising autoencoders can generate more robust representation which improves classification. In [10] sparsity was introduced in terms of firing neurons, if the neurons are of high value (near about 1), it is allowed to be fired, the rest are not. In [11], only the top K high valued neurons are fired; in [12] only the neurons beyond a predefined threshold were fired. One study proposed regularizing the autoencoder by introducing a penalty so that the network representation; this is the Contractive Autoencoder (CAE) [13] – regularization helps combat the problem of overfitting.

If training data is not sufficient, all machine learning tools tend to overfit. One of the ways to combat overfitting in neural networks is based on randomly switching off and on the nodes –

DropOut [14], switching on and off the connections – DropConnect [15] or both DropAll [16]. These are stochastic regularization techniques and work well with statistical generative models. They have been successfully used in neural networks, but to the best of our knowledge have not been used in autoencoders which have a deterministic formulation. For deterministic formulations the most straightforward way to combat overfitting is to regularize the cost function; CAE is such an example. Classical studies in neural networks introduced sparsity in neural networks to combat overfitting, e.g. in Optimal Brain Damage (OBD) [17], all the low valued connections were set to 0 – equivalent to thresholding.

1.1. Aim of the thesis

With this work, we add different penalty terms to the basic Autoencoder formulation in order to regulate the learning capacity of the Autoencoders. We aim to learn structured feature set which will lead to improvement in the performance of the autoencoders.

In this work, we propose to introduce sparsity in the autoencoder connections by imposing sparsity promoting penalty on the connection weights; hence this work is analogous to the DropConnect method. It is motivated by the fact that a majority of natural signals are correlated. This concept is widely used by the machine learning and computer vision community; the best example is convolutional neural network (CNN) where local correlations are exploited by limiting the connections from previous layers to following ones. In a sense, this is a hand-picked sparse connection. The concept of local correlation and convolution follows from traditional image processing, where local similarity was exploited for various operations – edge detection, denoising, smoothing etc. More recent techniques in image processing are based on non-local similarities [18], [19]. The concept behind our work is based on the notions of non-local correlations. It is not possible to hand-pick connections for non-local correlations. In image processing applications, these correlations need to be searched for. In our work, we model these non-local correlations in terms of sparse weight. We expect that wherever there will be a correlation, the connection weight will be non-zero and when there is no similarity the weight will be zero.

In the last decade, there has been a plethora of work in signal processing related to the topic of sparse solutions. Sparsity usually arises when natural signals are "whitened" (decorrelated). In neural networks, theoretically all the network connections can be independent, but given the correlations within the data, and the need to capture it, the connections are usually never independent. That is the reason behind the success of OBD, where it was assumed that only the 'important' (high valued) connections need to be preserved while the rest can be switched off (put to 0). So in this work, we propose to use a new technique to capture the redundancy of network connections. We postulate that the connections are not really independent and when stacked as columns of a matrix will result in a low-rank structure. The low-rank penalty can be easily introduced into the existing autoencoder formulation by adding a nuclear norm (closest convex proxy of rank) penalty to the Euclidean cost function.

In this work we propose a new regularization penalty for the autoencoder – we add a nuclear norm term (on the learned features). This is particularly useful for classification problems where the training data is labeled. We postulate that training samples for each class may look widely different but autoencoders can learn a representation such that the features for the same class are similar (linearly dependent) to each other. If our assumption is true, features from the same class will be correlated with each other; when stacked as columns of a matrix, this would lead to a low-rank matrix (owing to linear dependency). For example, consider a simple problem of character recognition. The characters (especially when hand-written) may be tilted at various orientations; when the samples for each character are stacked as a matrix, it would not be low-rank – owing to the tilts they would not be linearly dependent. But if we can learn a mapping from the sample space to the feature space so that the features are aligned (same tilt), the features for one character (when stacked as columns) will form a low-rank matrix. The low-rank penalty on the features learns autoencoding weights that approximately map the samples to such a feature/latent space such that the representation (features) looks similar for each class. Low-Rank Representation (LRR) has been successful in uncovering similarities in the data; it has been successful in many applications related to signal processing, computer vision, and machine learning. There are several extensions to the basic LRR formulation, e.g. Structure Constrained Low-Rank Representation (SC-LRR) [20] and discriminative LRR [21]. However, this method is not related to the LRR technique.

1.2. Organization of the thesis

The rest of the thesis is organized as follows:-

Chapter 2:- Discusses the literature related to basic of Autoencoders, Stacked Autoencoders, and different variations of Autoencoders and we also discuss the connection of Autoencoders with Dictionary Learning.

Chapter 3:- In this chapter, we discuss the various datasets used for performance evaluation (Classification and Denoising) of our proposed methods and their comparison with other existing techniques.

Chapter 4:- This chapter discusses the first proposed formulation - sparsely connected Autoencoders. We derive algorithms for the formulations for both l_1-norm penalty and l_0-norm penalty. We also discuss Majorization-Minimization technique. Next, we give results of our proposed method and show our superiority of methods using classification and denoising performance and also through training times.

Chapter 5:- This chapter discusses the second proposed formulation – by modeling rank deficiency in the weights of Autoencoders. We derive an algorithm for the method and also briefly discuss the Singular Value shrinkage algorithm. We give results of our proposed method over various datasets and show our superiority of our method over others using classification and denoising performance and also through training times.

Chapter 6:- This chapter discusses the third proposed formulation – modeling rank deficiency within the classes in an Autoencoder. We derive algorithms for the formulation using Split Bregman Technique. Next, we give results of our proposed method and show our superiority of methods using classification.

The last section concludes the thesis.

CHAPTER 2

2. Literature Review

An autoencoder consists (as seen in Figure. 1.1) of two parts – the encoder maps the input to a latent representation, and the decoder maps the latent representation back to the data. For a given input vector (including the bias term) *x*, the latent space is expressed as:

$$h = Wx \tag{2.1}$$

Here the rows of *W* are the link weights from all the input nodes to the corresponding latent nodes. Usually, a non-linear activation function is used at the output of the hidden nodes leading to:

$$h = \phi(Wx) \tag{2.2}$$

The sigmoid function is popular; other non-linear activation functions (like tanh) can be used as well. Rectifier units in large neural networks employ linear activation functions (identity) – this considerably speeds up training.

The decoder portion reverse maps the latent variables to the data space.

$$x = W'\phi(Wx) \tag{2.3}$$

During training the problem is to learn the encoding and decoding weights – *W* and *W'*. These are learned by minimizing the Euclidean cost:

$$\arg\min_{W,W'} \left\| X - W'\phi(WX) \right\|_F^2 \tag{2.4}$$

Here $X = [x_1 | ... | x_N]$ consists all the training sampled stacked as columns. The problem (2.4) is clearly non-convex, but is smooth and hence can be solved by gradient descent techniques; the activation function needs to be smooth and continuously differentiable.

There are several extensions to the basic autoencoder architecture. Stacked autoencoders have multiple hidden layers – one inside the other (see Figure. 2.1). The corresponding cost function is expressed as follows:

$$\arg\min_{W_1,\ldots,W_L,W_1',\ldots,W_L'} \| X - g \circ f(X) \|_F^2 \tag{2.5}$$

Where $g = W_1'\phi(W_2'....W_{L'}(f(X)))$ and $f = \phi(W_L\phi(W_{L-1}.....\phi(W_1X)))$.

Solving the complete problem (2.5) is computationally challenging. Also learning so many parameters (network weights) lead to over-fitting. To address both these issues, the weights are usually learned in a greedy fashion – one layer at a time [7] i.e. the outermost weights are first learned. The features i.e. product of the learned weights multiplied with the input is used to learn the second layer and so on the deeper layers.

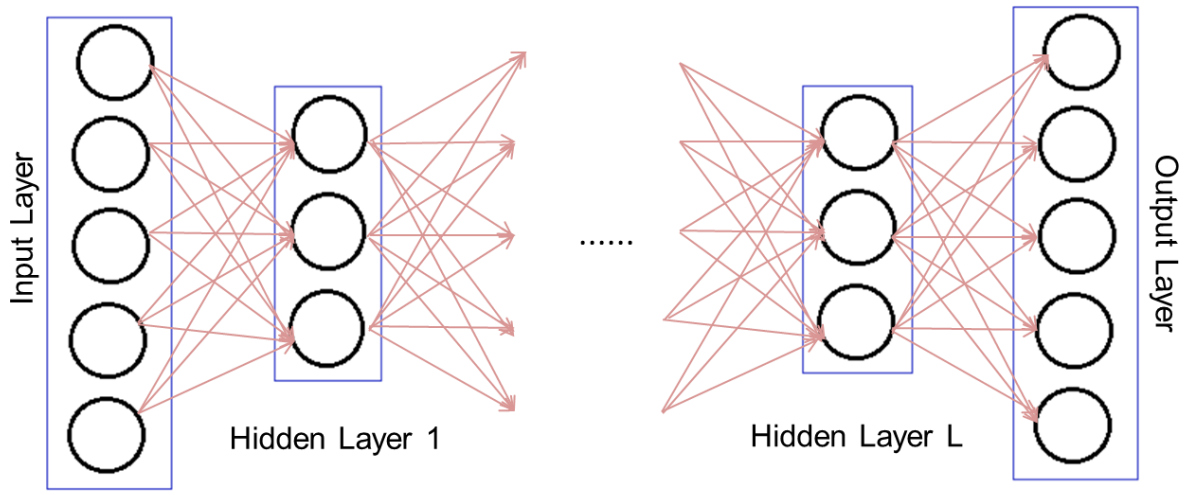


Figure 2.1: Stacked Autoencoder

Stacked Denoising Autoencoder (SDAE) [6] is a variant of the basic autoencoder where the input consists of noisy samples and the output consists of clean samples. Here the encoder and decoder are learned to denoise noisy input samples. The learned features appear to be more robust when learned by SDAE. In a recent work a Marginalized Denoising autoencoder was proposed [22], which does not have any intermediate nodes but learns the mapping from the input to the output. This formulation is convex (unlike regular autoencoders); the trick here is to marginalize over all possible noisy samples so that the dataset need not be augmented like SDAE. Such an autoencoder was used for domain adaptation.

Another variation for the basic autoencoder is to regularize it, i.e.

$$\arg\min_{(W)s} \| X - g \circ f(X) \|_F^2 + R(W, X) \tag{2.6}$$

The regularization can be a simple Tikhonov regularization however that is not used in

practice. It can be a sparsity promoting term [10], [11] and [12] or a representation penalizing term (Frobenius norm of the Jacobian) as used in the Contractive Autoencoder [13]. The regularization term is usually chosen so that they are differentiable and hence minimized using gradient descent techniques.

Sparse regularization penalties have also been applied to the basic autoencoder formulation. For example one can introduce sparsity in features by the following formulation:

$$\underset{(W)s}{\arg\min} \| X - g \circ f(X) \|_F^2 + \lambda \| f(X) \|_1 \tag{2.7}$$

here the nodes will be dropped at random which is analogous to Dropout [14] in Neural Networks.

In [11], instead of employing convex surrogate (l_1 -norm) exact sparsity was introduced as:

$$\underset{(W)s}{\arg\min} \| X - g \circ f(X) \|_F^2 \; s.t. \| f(X) \|_0 < k \tag{2.8}$$

here sparsity is introduced on the nodes such that top k-highest valued nodes(neurons) will be fired and the rest will be dropped. Solving (2.8) is NP-hard, but following techniques in greedy sparse recovery the authors of [11] solve (2.8).

2.1. Connection with Dictionary Learning

In dictionary learning, the objective is to learn a basis for representing the data. It simultaneously learns the basis and the representation / features of the data. Mathematically this is represented as:

$$X = DZ \tag{2.9}$$

Here X is the training data, D is the basis / dictionary and Z are the features. Both D and Z need to be learned. The simplest learning method is called Method of Optimal Directions (MOD) [23] which solves the following l_2-norm cost function.

$$\min_{D,Z} \| X - DZ \|_F^2 \tag{2.10}$$

This is a bilinear non-convex problem. It is usually solved using alternating least squares technique, i.e. *D* is solved assuming *Z* is fixed and then *Z* is solved assuming a fixed *D*. These two steps are carried out in every iteration until some local convergence criterion is satisfied.

Today, most successful dictionary learning methods impose a sparsity penalty on the features. Thus the problem is expressed as:

$$\min_{D,Z} \|X - DZ\|_F^2 \; s.t. \|Z\|_0 \leq \tau \tag{2.11}$$

The *l0*-norm imposes sparsity in *Z*. The ensuing problem (2.11) is solved neatly using K-SVD [24].

In signal processing, such a formulation is called the synthesis prior, where the learned features are supposed to be sparse. There is an alternate analysis prior formulation [25] – where the learned dictionary sparsifies the data, i.e. $D_A X$ is supposed to be sparse. We will use D_A to denote the analysis prior dictionary and D_S to denote the synthesis prior dictionary. The learning for Analysis dictionary learning is expressed as:

$$\min_{D,X_0} \|X - X_0\|_F^2 \; s.t. \|D_A X_0\|_0 \leq \tau \tag{2.12}$$

An autoencoder basically learns both the analysis and the synthesis mapping simultaneously. If we combine the analysis and the synthesis dictionary learning (ignoring sparsity penalty), we will get something like:

$$\min_{D_s,D_A} \|X - D_s(D_A X)\|_F^2 \tag{2.13}$$

This is the expression of a linear autoencoder. With sparsity penalties imposed on the features, the expression takes the form:

$$\min_{D_s,D_A} \|X - D_s(D_A X)\|_F^2 \; s.t. \|D_A X\|_0 \leq \tau \tag{2.14}$$

Or the alternate unconstrained formulation

$$\min_{D_s,D_A} \|X - D_s(D_A X)\|_F^2 + \lambda \|D_A X\|_0 \tag{2.15}$$

The first expression (2.14) is basically the linear version of the K-sparse autoencoder [11] while the second expression (2.15) is the linear version of the sparse autoencoder [12].

CHAPTER 3

3.Dataset Description

For performance evaluation of our work, we have compared our results with several state-of-the-art techniques on several benchmark datasets including image and speech dataset. This chapter will describe the datasets used for Classification and Denoising used throughout the work. The images in the image dataset are vectorized and this vectorized form of an image is used as training or testing sample.

3.1. For Classification (Digit Recognition) Task.

3.1.1. MNIST

The MNIST dataset that consists of 28x28 images of handwritten digits (English Numerals) ranging from 0 to 9 making a 10 class classification problem. The dataset has 60,000 samples for training and 10,000 samples for testing. It is a subset of a larger set available from NIST. We did not perform any pre-processing on the images.

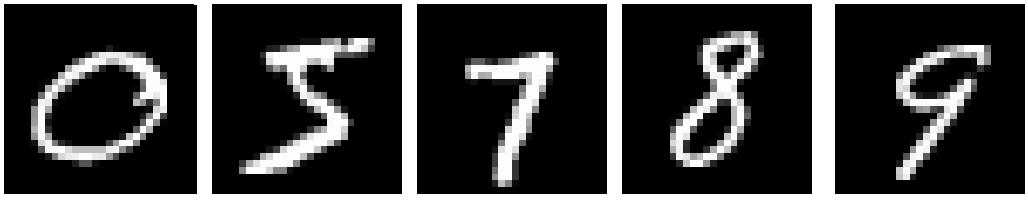


Figure 3.1: Numerals from MNIST Dataset.

The other datasets are variations of MNIST which again are 28x28 images with 0 to 9 labels and are more challenging primarily because they have fewer training samples (12,000) and a larger number of test samples (50,000). These datasets are simple variations in MNIST dataset or combination of variations in MNIST dataset. MNIST and its variations make for a feature vector of 784 (28x28) length.

3.1.2. MNIST basic

A smaller subset of MNIST dataset. Again no pre-processing has been applied to this dataset.

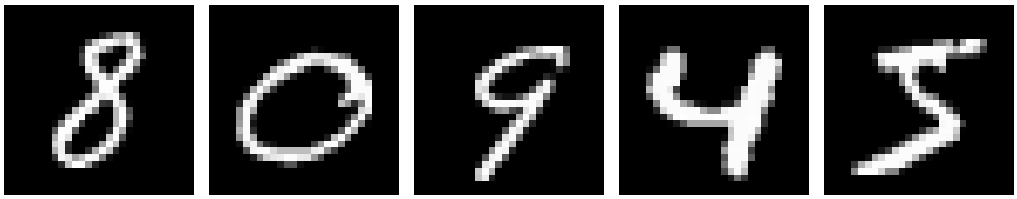


Figure 3.2: Numerals from MNIST basic Dataset.

3.1.3. MNIST rot

The digits were rotated by an angle generated uniformly between 0 and 2π radians. Thus the factors of variation are the rotation angles and the factors of variation already contained in MNIST, such as handwriting style. No pre-processing has been done on this dataset as well.

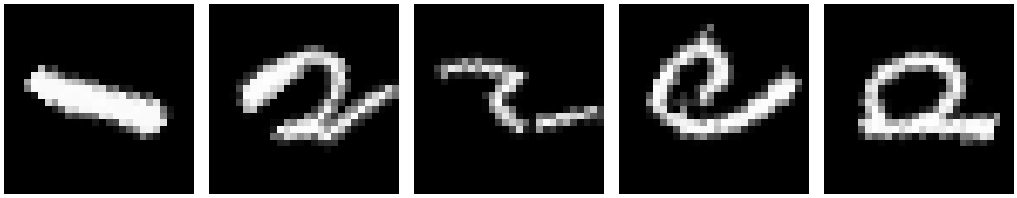


Figure 3.3: Numerals from MNIST rot Dataset.

3.1.4. MNIST back-rand

A random background was inserted in the digit image. Each pixel value of the background was generated uniformly between 0 and 255. Simple thresholding of the pixel values is done as part of processing before feeding to the network.

Figure 3.4: Numerals from MNIST back-rand Dataset.

3.1.5. MNIST back-image

A patch from a black and white image was used as the background for the digit image. The patches were extracted randomly from a set of 20 images downloaded from the internet. Patches which had low pixel variance (i.e. contained little texture) were ignored. Simple thresholding of the pixel values is done as part of processing before feeding to the network.

Figure 3.5: Numerals from MNIST back-image Dataset.

3.1.6. MNIST rot-back-image

The perturbations used in *mnist-rot* and *mnist-back-image* were combined. Simple thresholding of the pixel values is done as part of processing before feeding to the network.

Figure 3.6: Numerals from MNIST rot-back-image Dataset.

3.1.7. USPS

The dataset refers to numeric data obtained from the scanning of handwritten digits from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been de-slanted and size normalized, resulting in 16 x 16 greyscale images with labels from 0 to 9 making again a 10 class problem. There are 7291 training samples and 2007 test samples. The test set is considered notoriously "difficult". USPS makes for a feature vector of 256 (16x16) length.

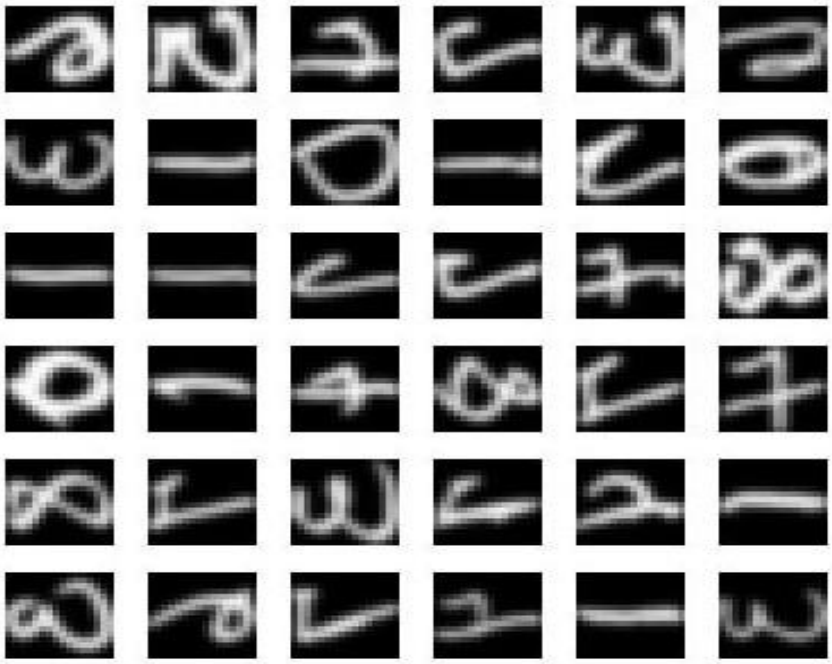


Figure 3.7: Numerals in original USPS dataset.

3.1.8. Devanagari Numerals

This dataset contains images of handwritten Devanagari Numerals. The images in the original dataset were of different sizes, so all the images were resized to 32x32. The labels are from 0 to 9, again a 10 class classification problem. There are 18783 training images and 3763 testing images. No other pre-processing was done on the images. Devanagari makes for a feature vector of 1024 (32x32) length.

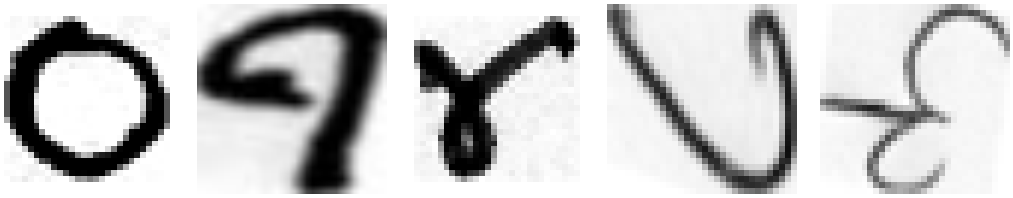


Figure 3.8: Numerals from Devanagari Dataset.

3.1.9. Bangla Numerals

This dataset contains images of handwritten Bangla Numerals. The images in the original dataset were of different sizes, so all the images were resized to 32x32. The labels are from 0 to 9, again a 10 class classification problem. There are 19392 training images and 4000 testing images. No other pre-processing was done on the images. Bangla makes for a feature vector of 1024 (32x32) length.

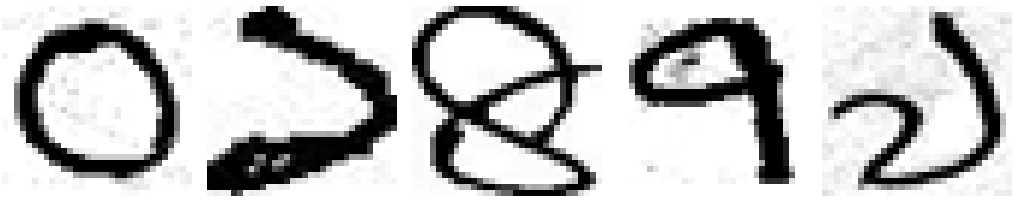


Figure 3.9: Numerals from Bangla Dataset.

3.2. For Classification (Spoken letter Recognition) Task.

3.2.1. ISOLET

ISOLET (Isolated Speech Letter Recognition) is a speech dataset of spoken English alphabets. The data was collected in 1994 by Ron Cole and Mark Fanty (Department of Computer Science and Engineering, Oregon Graduate Institute, Beaverton). It contains 6238 training samples and 1559 testing samples with 617 attributes and 26 classes. All attributes are continuous, real-valued attributes scaled into the range -1.0 to 1.0. The features include spectral coefficients; contour features, sonorant features, pre-sonorant features, and post-

sonorant features. The exact order of appearance of the features is not known. It is difficult classification problem with huge 26 classes. No pre-processing was done on the data.

3.3. For Denoising Task.

3.3.1. CIFAR-10

We perform denoising task on CIFAR-10 dataset. The CIFAR-10 dataset is composed of 10 classes of natural images with 50,000 training examples in total, 5,000 per class and 10,000 testing samples. Each image is an RGB image of size 32x32 (feature vector of 1024) taken from the tiny images dataset and labeled by hand. The images were first converted to greyscale and then zero mean Gaussian noise was added to both training and testing images. The noisy images served as the input to the autoencoders and the clean images were the output (Denoising Autoencoder).

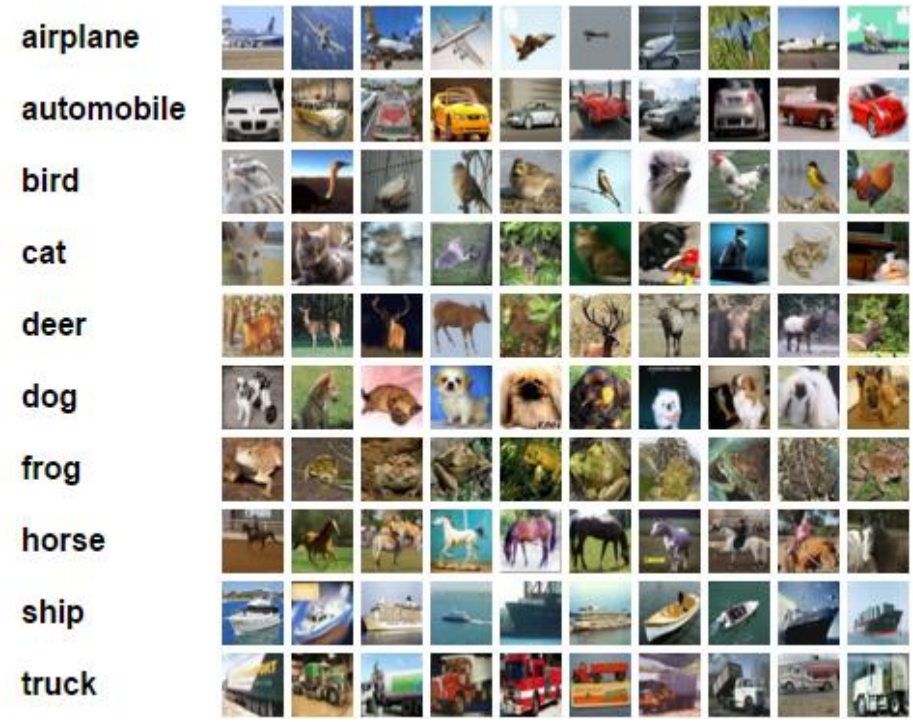


Figure 3.10: Images from CIFAR-10 Dataset.

PART-I

MODELING REDUNDANCY IN THE NETWORK

CHAPTER 4

Proposed Formulation -I

4. Sparsely Connected Autoencoder

4.1. Proposed Formulation

Autoencoders usually have a non-linear activation function. However, in [26] it was shown that an autoencoder usually operates in the linear region even if the activation function is non-linear. Therefore in this work, we will use a linear activation function. This allows us to derive a more efficient algorithm which is faster than its non-linear counterparts. We also show (experimentally) that the linear autoencoder yields better results than its non-linear counterpart. The basic formulation of an autoencoder with linear activation function is given by:

$$\arg\min_{W,W'} \left\| X - W'WX \right\|_F^2 \tag{4.1}$$

The basic autoencoder is prone to overfitting; especially when the number of training samples is limited. Denoising autoencoders use a stochastic regularization technique. However, given the Euclidean cost function of the autoencoder a more direct way to regularize it would be incorporate penalty terms to the basic formulation. For example, when we apply Tikhonov Regularization on the weights of the Autoencoder the formulation leads to:

$$\arg\min_{W,W'} \left\| X - W'WX \right\|_F^2 + \lambda \left(\left\| W \right\|_F + \left\| W' \right\|_F \right) \tag{4.2}$$

The Frobenius norm on the weights regularizes the network to have small values. The regularization prevents overfitting of the network.

We propose to regularize the autoencoder such that it has sparse connections both at the encoder and the decoder. The idea of trimming irrelevant connections in neural networks is not new; it was first proposed back in 1990 in the form of optimal brain damage [17]. In recent times, learning sparse structures in neural networks has gained momentum [27] and [28]. However, to the best of our knowledge, there is no work that learns autoencoders with

sparse connections. Prior studies in sparse autoencoder [10], [11] and [12] concentrate on sparse activities; not on sparse connections. In this respect, ours is the first work to propose sparsely connected autoencoders.

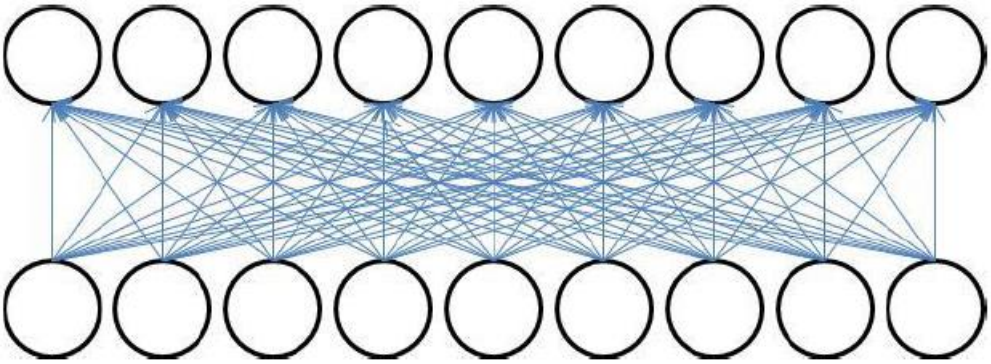


Figure 4.1: Dense Connections

Just as a human brain does not require all its neurons for a specific task, we postulate that an autoencoder does not need to utilize all its connections either. The issue of maintaining important connections without over-fitting is taken care of, if we have sparse weights. The portions which are not useful for representation are pruned and only the important connections in the network are maintained. Such a sparse connection is easily achieved from the following proposed formulation:

$$\arg\min_{W,W'} \|X - W'WX\|_F^2 + \lambda\left(\|W\|_{1/0} + \|W'\|_{1/0}\right) \tag{4.3}$$

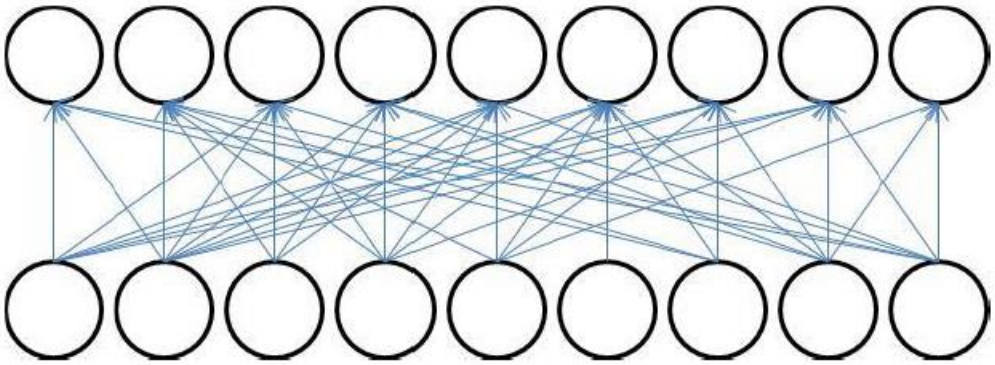


Figure 4.2: Sparse Connections

We have abused the notation a bit, the subscript 1/0 denotes either an l_1-norm or an l_0-norm and is defined on the vectorial representation of the weights. The l_1-norm is convex and has been widely used in recent times by Compressed Sensing. But the l_1-norm does not ideally yield sparse weights, the l_0-norm does. Unfortunately, l_0-norm minimization is an NP-hard problem. However, there are approximate techniques to solve such l_0-minimization problems.

Autoencoders with non-linear activation function are solved using gradient descent techniques. Such techniques cannot be directly applicable for our proposed formulation. This is because the l_1/l_0-norm penalties are not differentiable everywhere. In this work, we follow a Majorization-Minimization approach to solve the said problem. Majorization-Minimization Technique is demonstrated in the next section.

4.1.1. Majorization-Minimization

Let, $J(x)$ be the function to be minimized. Start with an initial point (at $k=0$) x_k (Fig. 4.3(a)). A smooth function $G_k(x)$ is constructed through x_k which has a higher value than $J(x)$ for all values of x apart from x_k, at which the values are the same. This is the Majorization step. The function $G_k(x)$ is constructed such that it is smooth and easy to minimize. At each step, minimize $G_k(x)$ to obtain the next iterate x_{k+1} (Fig. 4.3(b)). A new $G_{k+1}(x)$ is constructed through x_{k+1} which is now minimized to obtain the next iterate x_{k+2} (Fig. 4.3(c)). As can be seen, the solution at every iteration gets closer to the actual solution.

For convenience we express the problem (4.3) in a slightly different manner in terms of transposes:

$$\arg\min_{H} \left\| X^T - X^T H^T \right\|_F^2 + R(H) \tag{4.4}$$

Here $H=W'W$ and $R(H)$ denotes the penalty.

Only the least squares part need to be majorized; the penalty terms are not affected.

$$J(H) = \left\| X^T - X^T H^T \right\|_F^2 + R(H) \tag{4.5}$$

For this minimization problem, $G_k(x)$, the majorizer of $J(x)$ is chosen to be:

$$G_k(H) = \left\| X^T - X^T H^T \right\|_F^2 + (H^T - H_k^T)^T (aI - XX^T)(H^T - H_k^T) \tag{4.6}$$

where a is the maximum eigenvalue of the matrix XX^T and I is the identity.

One can check that at $H=H_k$ the expression $G_k(H)$ reduces to $J(H)$. At all other points it is larger than $J(H)$; the value of 'a' assures that the second term is positive definite.

$$\begin{aligned}
&G_k(H) = \left\| X^T - X^T H^T \right\|_F^2 + (H^T - H_k^T)^T (aI - XX^T)(H^T - H_k^T) + R(H) \\
&\Rightarrow XX^T - 2XX^T H^T + HXX^T H^T + (H^T - H_k^T)^T (aI - XX^T)(H^T - H_k^T) + R(H) \\
&\Rightarrow XX^T + H_k(aI - XX^T)H_k^T - 2(XX^T + H_k(aI - XX^T))H_k^T + aHH^T + R(H) \\
&\Rightarrow a(-2BH^T - HH^T) + C + R(H) \\
&\text{where } B^T = H_k^T + \frac{1}{a} X(X^T - X^T H_k) \\
&C = XX^T + H_k(aI - XX^T)H_k^T
\end{aligned}$$

Using the identity $\left\| A - D \right\|_2^2 = A^T A - 2A^T D + D^T D$, one can write

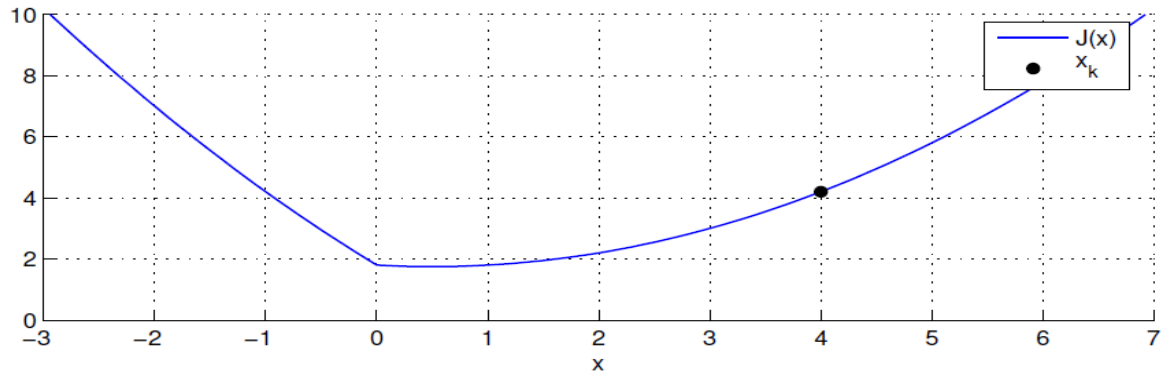
$$\begin{aligned}
&G_k(H) = a\left\| B - H \right\|_2^2 - aB^T B + C + R(H) \\
&= a\left\| B - H \right\|_2^2 + R(H) + K
\end{aligned}$$

where K consists of terms independent of x.

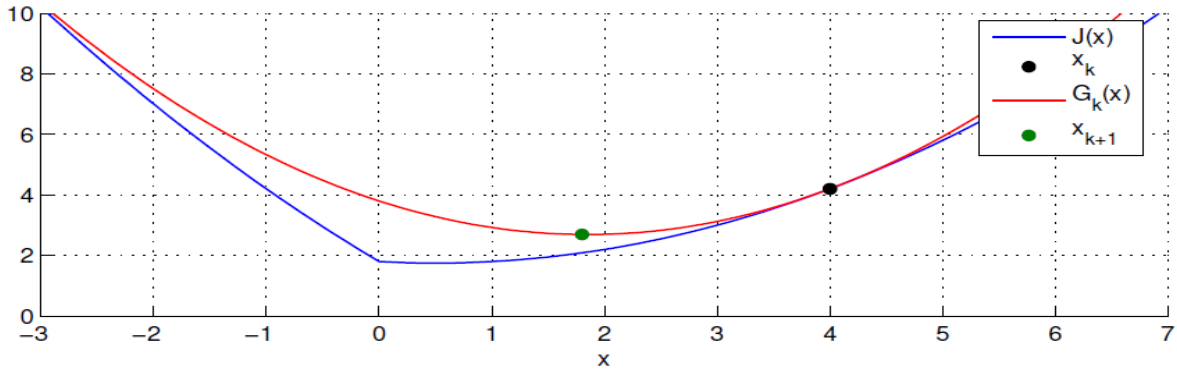
Therefore, minimizing $G_k(x)$ is the same as minimizing the following:

$$G_k(H) = \left\| B - H \right\|_2^2 + \frac{1}{a} R(H) \tag{4.7}$$

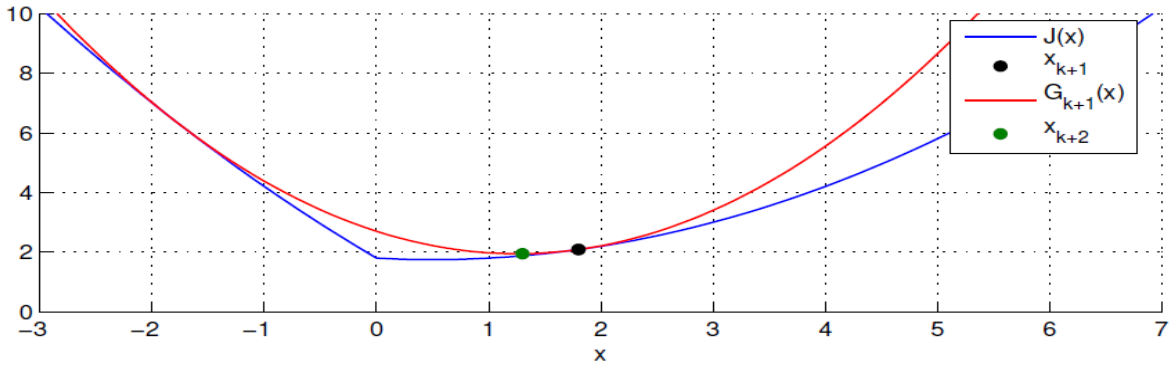
Where $B^T = H_k^T + \frac{1}{a} X(X^T - X^T H_k)$ and $a \geq \max eig(XX^T)$. This update is known as the Landweber iteration.



(a)



(b)



(c)

Figure 4.3: Majorization-Minimization

[29]

4.1.2. l_1-*norm* penalty

First, we derive the algorithm for solving the *l1*-norm minimization problem.

$$\arg\min_{W,W'} \left\|B - W'W\right\|_F^2 + \lambda\left(\left\|W\right\|_1 + \left\|W'\right\|_1\right) \tag{4.8}$$

This is a bilinear problem and we propose to solve it alternately, i.e. we fix *W'* and solve *W* and then solve *W'* assuming *W* is fixed. These two steps are done in every iteration.

$$W_{k+1} = \arg\min_{W} \left\|B - W_k'W\right\|_F^2 + \frac{\lambda}{a}\left\|W\right\|_1 \tag{4.9}$$

$$W'_{k+1} = \arg\min_{W'} \left\|B - W'W_{k+1}\right\|_F^2 + \frac{\lambda}{a}\left\|W'\right\|_1 \tag{4.10}$$

In both cases, the problem remains the same – that of the least squares minimization with l_1-norm penalty.

Let us take the first problem and work out the solution for it; the solution for the other problem will remain the same. To solve (4.9) we invoke the Majorization approach once again. Therefore (4.9) can be expressed as:

$$\arg\min_{W} \left\|P - W\right\|_F^2 + \frac{\lambda}{\alpha a}\left\|W\right\|_1 \tag{4.11}$$

Where $P = W_k + \frac{1}{\alpha}W'^T_k(B - W_k'W_k)$, $\alpha \geq \max eig(W'^T W')$.

The above function (4.11) is actually decoupled, i.e.

$$\left\|P - W\right\|_F^2 + \frac{\lambda}{\alpha a}\left\|W\right\|_1 = \sum_i (P_i - W_i)^2 + \frac{\lambda}{\alpha a}\left|W_i\right| \tag{4.12}$$

Therefore, (4.12) can be minimized term by term, i.e.

$$\frac{\partial(\left\|P - W\right\|_F^2 + \frac{\lambda}{\alpha a}\left\|W\right\|_1)}{\partial W_i} = 2P_i - 2W_i + \frac{\lambda}{\alpha a}\,signum(W_i) \tag{4.13}$$

Setting the partial derivatives to zero and solve for *W*. That is, the minimizer of (4.11) is obtained by applying the soft-threshold rule to *P* with a threshold $\frac{\lambda}{2a\alpha}$. The soft-threshold rule is the non-linear function defined as:

$$soft(x,\tau)=\begin{cases} x+\tau & x<-\tau \\ 0 & |x|=\tau \\ x-\tau & x>\tau \end{cases} \tag{4.14}$$

Or more compactly

$$W = \mathrm{s}ignum(P)\max(0,|P|-\frac{\lambda}{2a\alpha}) \tag{4.15}$$

This concludes the steps for solving (4.9); the steps for (4.10) are exactly the same. In a compact fashion, the algorithm for solving the *l1*-norm penalty problem is given in Algorithm 1.

Our initialization is deterministic, hence the results are results are repeatable – there is no variation between trials as long as other parameters remain same.

$Initialisation:$

$$H = \arg\min_{H} \left\| X^T - X^T H \right\|_F^2$$

$$H = USV^T$$

$$W_0^{'} = US \; and \; W_0 = V^T$$

$In\,every\,iteration\,'k'$

$$compute\; B^T = H_k^T + (\frac{1}{\alpha}) X\,(X^T - X^T H_k)$$

$$update\; W_{k+1} = \arg\min_{W} \left\| B - W_k^{'} W \right\|_F^2 + \frac{\lambda}{\alpha} \left\| W \right\|_1$$

$$P = W_k + \frac{1}{a} W'^T_k (B - W_k^{'} W_k)$$

$$W_{k+1} = signum(P)\max(0,|\,P\,| - \frac{\lambda}{2a\alpha})$$

$$similarly\; update\; W'_{k+1} = \arg\min_{W^{'}} \left\| B - W^{'} W_{k+1} \right\|_F^2 + \frac{\lambda}{\alpha} \left\| W \right\|_1$$

Algorithm 1: Proposed *l_1-norm*

4.1.3. l_0-*norm* penalty

The l_1-norm penalty is basically a shrinkage function defined by the soft thresholding. It cannot get an exactly sparse solution; it only shrinks the values of unwanted weights. To get a sparse solution in every iteration, one needs to solve the l_0- norm minimization problem. This is an NP-hard problem but has approximate solutions. The more common practice is to solve using a greedy approach based on Orthogonal Matching Pursuit [30] and [31]. However, these are not efficient for solving large scale problems. To solve a k-sparse problem, k iterations are required. A better approach to solve (4.16) is based on Iterative Hard Thresholding [32].

$$\arg\min_{W,W'} \left\|B - W'W\right\|_F^2 + \lambda\left(\left\|W\right\|_0 + \left\|W'\right\|_0\right) \tag{4.16}$$

We solve it via alternating minimization.

$$W_{k+1} = \arg\min_{W} \left\|B - W_k'W\right\|_F^2 + \frac{\lambda}{a}\left\|W\right\|_0 \tag{4.17}$$

$$W'_{k+1} = \arg\min_{W'} \left\|B - W'W_{k+1}\right\|_F^2 + \frac{\lambda}{a}\left\|W'\right\|_0 \tag{4.18}$$

As before, both the problems remain the same. We only derive the algorithm to solve (4.17). To solve it, we invoke the majorization-minimization approach once again. Therefore (4.17) can be expressed as:

$$\arg\min_{W} \left\|P - W\right\|_F^2 + \frac{\lambda}{\alpha a}\left\|W\right\|_0 \tag{4.19}$$

Where $P = W_k + \frac{1}{\alpha}W'^T_k(B - W_k'W_k)$, $\alpha \geq \max eig(W'^T W')$

This is a decoupled problem and can be expressed as:

$$\left\|P - W\right\|_F^2 + \frac{\lambda}{\alpha a}\left\|W\right\|_0 = (P_1 - W_1)^2 + \frac{\lambda}{\alpha a}\left|W_1\right|^0 + \ .... + (P_n - W_n)^2 + \frac{\lambda}{\alpha a}\left|W_n\right|^0 \tag{4.20}$$

We can process (4.20) element-wise.

To derive the minimum, two cases need to be considered: case 1 –W_i = 0and case 2 – $W_i \neq 0$. The element-wise cost is 0 in the first case. For the second case, the minimum is reached when $W_i = P_i$. Comparing the cost in both cases: 0 if $W_i = 0$ and $-(W_i)^2 + \frac{\lambda}{a\alpha}$ *if* $W_i = P_i$.

This suggests the following updates rule:

$$W_i = \begin{cases} P_i & |P_i| > \lambda / 2a\alpha \\ 0 & |P_i| \leq \lambda / 2a\alpha \end{cases} \tag{4.21}$$

This is popularly known as hard thresholding and is represented as:

$$W_{k+1} = HardTh(P, \frac{\lambda}{2a\alpha}) \tag{4.22}$$

This leads to an algorithm somewhat similar to the previous one. It is succinctly represented below.

Algorithm 1 and Algorithm 2 show the training phases for both the methods. The processes are unsupervised. The testing phase is just multiplication of matrices hence very efficient and fast when we have a generalized network and want to get quick results.

Let Z be the Testing data. Then the features space of testing data (Z_1) will be:

$$Z_1 = WZ \tag{4.23}$$

And the recovered testing data (Z_2) will be given by:

$$Z_2 = W'WZ \tag{4.24}$$

For making a deeper network, we have adopted a greedy layer-wise training approach. The latent representation of the outermost layer is used to train the next deeper layer and so on we can train other deeper layers as per the performance at every layer. We employ the same penalty in each layer when building a Deep Network. We generally test the performance of the network at every layer and stop if there is a decrement in the performance or there is no significant improvement in the performance by going deeper into the network.

$$Initialisation:$$

$$H = \arg\min_{H} \left\| X^T - X^T H \right\|_F^2$$

$$H = USV^T$$

$$W_0^{'} = US \; and \; W_0 = V^T$$

$$In \; every \; iteration \; 'k'$$

$$compute \; B^T = H_k^T + (\frac{1}{\alpha}) X (X^T - X^T H_k)$$

$$update \; W_{k+1} = \arg\min_{W} \left\| B - W_k^{'} W \right\|_F^2 + \frac{\lambda}{\alpha} \left\| W \right\|_0$$

$$P = W_k + \frac{1}{a} W_k^{'T} (B - W_k^{'} W_k)$$

$$W_{k+1} = HardTh(P, \frac{\lambda}{2a\alpha})$$

$$similarly \; update \; W^{'}_{k+1} = \arg\min_{W^{'}} \left\| B - W^{'} W_{k+1} \right\|_F^2 + \frac{\lambda}{\alpha} \left\| W \right\|_0$$

Algorithm 2: Proposed l_0-norm

4.2. Experimental Setup and Results

We have already discussed the various datasets used for comparing our proposed methods with state-of-the-art methods. Without elaborating on them further, we will discuss the results and establish the superiority of this proposed method. As discussed earlier we have used linear activation functions in our network instead of non-linear functions as used in most of the studies. We give experimental validation to that first in the following section through Classification task.

4.2.1. Linear vs. Non-linear

Most studies in neural networks employ a non-linear activation function. We proposed linear activation owing to the ease of solution. We will show that at least for the benchmark datasets used in these experiments, the simple linear (Identity) activation function yields better classification accuracy than their non-linear (sigmoid) counterpart.

Linear autoencoder: $\min_{W',W} \left\| X - W'WX \right\|_F^2$; Non-linear Autoencoder: $\min_{W',W} \left\| X - W'\phi(WX) \right\|_F^2$

The linear autoencoder weights are initialized by solving the least squares problem $\min_{Q} \left\| X - QX \right\|_F^2$ and setting W as the top (number of nodes) right singular vectors of Q. For the non-linear autoencoder, we use the Hinton's implementation [33].

The autoencoder architectures remain same otherwise; for MNIST datasets, linear and non-linear versions are three layered architectures. The nodes in the architecture vary as 392 (1st layer)-198 (2nd layer)-92 (3rd layer). In the case of USPS both are two layered with 180 nodes in 1st layer and 100 in the 2nd. In case of Bangla and Devanagari dataset linear architecture is three layered with nodes varied as 512 (1st layer)-256 (2nd layer)-128 (3rd layer) but for non-linear version to get the best results possible we made two layered architectures with 1600 in the 1st layer and 800 nodes in the 2nd layer for Devanagari and 512 in 1st and 256 nodes in the 2nd layer for Bangla. In the case for ISOLET both the architectures are single layered with 400 nodes. The representation from the deepest layer is used for classification.

We have used two classifiers that are K-Nearest Neighbors (KNN) and Sparse Representation Classifier (SRC) [34]. Both the classifiers are non-parametric. Our objective is to see how good the learned features are from various deep learning tools. This is best compared on such non-parametric classifiers. Parametric classifiers like neural networks and support vector machines can be tuned to yield very good results on individual datasets by using separate fine-tuned parameter values. In such a case it is difficult to ascertain if the improvement in the result is owing to better features / representation or owing to the better choice of classification parameters.

The results in Table 4.1 show that the linear autoencoder always yields better results. The improvement is small when the number of training samples is larger (MNIST) but for other more challenging datasets, the linear autoencoder yields improve by a large margin.

Dataset	KNN		SRC	
	Linear	**Non-linear**	**Linear**	**Non-linear**
MNIST	**97.33**	96.11	**98.33**	97.29
basic	**95.25**	94.86	**96.91**	96.43
basic-rot	**84.83**	80.71	**90.04**	84.29
bg-img	**77.16**	70.97	**84.14**	76.94
bg-rand	**86.42**	81.11	**91.03**	85.49
bg-img-rot	**52.21**	44.60	**62.46**	50.96
USPS	**94.92**	94.12	**96.06**	94.37
Devanagari	**91.15**	68.88	**93.70**	56.66
Bangla	**81.78**	52.85	**89.05**	55.27
ISOLET	**88.19**	87.17	**93.07**	91.73

Table 4.1: Classification Accuracy (%) - Linear Vs. Non-linear Results

4.2.2. Classification Performance

We compare our results with the stacked autoencoder (SAE), Tikhonov Regularized Autoencoder (TRAE) and Deep Belief Network (DBN) [35]. The nodes in the architecture vary as 392 (1^{st} layer)-198 (2^{nd} layer)-92 (3^{rd} layer). We report the results for the deepest layer i.e. the 3^{rd} layer here. The SAE and TRAE use linear activation. As before, the representation from the deepest layer is used as features. For our sparsely connected autoencoder λ=0.01 is

used. In each of the tables, the best results are shown in bold. We test with three classifiers – KNN (K=1), SRC and SVM (RBF kernel- 5 fold cross-validation); results are shown in Tables 4.2, 4.3, and 4.4 respectively.

Dataset	**SAE**	**TRAE**	**DBN**	**Proposed *l0-penalty***	**Proposed *l1-penalty***
MNIST	**97.33**	82.83	97.05	97.21	95.91
basic	95.25	79.92	95.37	**95.39**	92.49
basic-rot	84.83	58.56	84.71	**85.14**	81.01
bg-rand	86.42	65.61	86.36	**86.88**	68.87
bg-img	77.16	58.51	77.16	**77.22**	79.84
bg-img-rot	**52.21**	27.10	50.47	51.80	38.91

Table 4.2: Classification Accuracy (%) - KNN (K=1) Results

Dataset	**SAE**	**TRAE**	**DBN**	**Proposed *l0-penalty***	**Proposed *l1-penalty***
MNIST	98.33	87.19	98.26	**98.42**	97.16
basic	96.91	85.03	96.80	**97.03**	95.43
basic-rot	90.04	68.63	89.04	**90.19**	87.76
bg-rand	91.03	72.25	88.49	**91.69**	76.17
bg-img	84.14	65.68	83.72	**85.11**	85.84
bg-img-rot	62.46	34.01	55.89	**62.61**	47.75

Table 4.3: Classification Accuracy (%) - SRC Results

Dataset	SAE	TRAE	DBN	Proposed *l0-penalty*	Proposed *l1-penalty*
MNIST	98.50	88.74	98.53	**98.60**	97.70
basic	96.96	86.61	97.07	**97.12**	95.70
basic-rot	**89.43**	72.54	89.05	89.22	87.07
bg-rand	91.28	75.20	89.59	**91.73**	87.04
bg-img	84.86	68.76	85.46	**85.35**	78.01
bg-img-rot	60.53	40.97	58.25	**60.91**	46.30

Table 4.4: Classification Accuracy (%) - SVM Results

The results show that our proposed method yields better results than SAE and DBN on an all the datasets (except for the larger MNIST with KNN). Under fair comparison (keeping the classifiers to be same) one can say that our method yields better representation than other deep learning techniques like SAE, TRAE, and DBN.

TRAE yields the worst results. This is because the Tikhonov regularization terms lead to small values of all weights – it penalizes valid connections (connections which should have higher values) and overfits because none of the irrelevant connection weights are zeroes. The proposed SparseConnect with l_1-norm yields better results; this is because the l_1-norm penalty is more robust than the Tikhonov regularizer. It shrinks the invalid / unimportant connections and emphasizes important ones. But the shrinkage does not put the connections to zeroes – therefore the problem of overfitting remains. The best results, as expected are from the SparseConnnect with l_0-norm; hard thresholding aggressively drops unimportant connections. The value of λ controls the sparsity of the connections – higher the value, sparser are the connections.

We find from Table 4.5 that results do not decrease monotonically. For a particular value of λ, the classification accuracy reaches the maximum and falls off when the value deviates.

This is what we expect intuitively. There is some sparsity level beyond which, the autoencoder starts to lose information; when the sparsity level is lower (i.e. more connections) the autoencoder overfits and hence shows poorer results. The best results are obtained for $\lambda = 0.01$.

Dataset	λ=0.001		λ=0.005		λ=0.01		λ=0.05		λ=0.1	
	KNN	**SRC**	**KNN**	**SRC**	**KNN**	**SRC**	**KNN**	**SRC**	**KNN**	**SRC**
MNIST	97.06	98.26	96.90	98.25	**97.21**	**98.42**	97.08	98.28	96.70	97.92
basic	95.22	97.05	95.23	96.93	**95.39**	**97.03**	95.12	96.93	95.11	96.87

Table 4.5: Variation of Classification Accuracy (%) with λ

Next, we compare the results with stacked denoising autoencoder (SDAE), deep belief network (DBN) [6], sparse deep neural network (SDNN) [36] and optimal brain damage (OBD) [17]. We repeat the results from l_0-norm sparsely connected autoencoder with SRC (since these are the best results we obtained). SDAE and DBN use a fine tuning with a neural network classifier in the final stage. SDNN is a contemporary sparse deep classifier and OBD is a classical work with a shallow architecture. The results show that our method yields results which are at par with SDAE and DBN and are better than sparse neural networks.

Dataset	**SDAE**	**DBN**	**SDNN**	**OBD**	**Proposed *l0-penalty***
MNIST	98.72	**98.76**	98.57	97.99	98.42
basic	**97.16**	96.89	96.69	95.41	97.03
basic-rot	**90.47**	89.70	89.58	87.89	90.19
bg-rand	89.70	**93.27**	90.21	88.27	91.69
bg-img	83.32	83.69	82.96	80.65	**85.11**
bg-img-rot	56.24	52.61	51.63	50.19	**62.61**

Table 4.6: Classification Accuracy (%) - Comparative Results

The proposed methods are implemented in MATLAB. We have compared the training times of the different autoencoders. The configuration of the system is as follows.

RAM- 24 GB
OS- Red Hat Enterprise Linux Server release 7.0 (Maipo)
CPU - Intel(R) Xeon(R) CPU E5-2430 0 @ 2.20GHz , there are two cpus of 6 cores each
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 15360K

We only report results for the MNIST and the MNIST basic since the others have the same size as MNIST basic and takes approximately the same time to train.

We find that the proposed autoencoders are considerably faster, especially for larger datasets. The l_1-norm autoencoder is more than an order of magnitude faster compared to the stacked autoencoder and TRAE. This is mainly because it converges faster.

Dataset	**MNIST**	**basic**
SAE (linear)	111	6
SAE(non-linear)	251	53
TRAE	98	24
DBN	78	21
Proposed *l0-penalty*	50	6
Proposed *l1-penalty*	**4**	**1**

Table 4.7: Training time in minutes

4.2.3. Denoising Performance

Autoencoders have been used previously for image denoising. In [12] it was shown that autoencoder with sparse features leads to good denoising results. They showed results for Gaussian and impulse denoising. It is not optimal to remove impulse noise with autoencoders; this is because impulse noise is sparse, hence the l_2-norm data fidelity term is not appropriate. It is well known in image processing [37] and [38] that to remove impulse noise, one needs to employ a l_p-norm data fidelity term where 'p' lies between 0 and 1. Since we are formulating an autoencoder with l_2-norm data fidelity, we can optimally remove Gaussian noise only; this is the problem we address in this work.

For comparison, we use the standard metrics for image quality assessment - PSNR (Peak Signal to Noise Ratio) and SSIM (Structural Similarity Index) [39]. We compare our approach (SparseConnect) with the sparse autoencoder [12] and stacked denoising autoencoder (SDAE). We use single layer autoencoders for image denoising. The number of nodes in the hidden layer is kept to be 512 and only one layer is used. The value of λ is 0.001.

The Denoising dataset used is also explained in chapter 3. The noisy images served as the input to the autoencoders and the clean images were the output. For testing, the noisy test images were as inputs and the image obtained at the output was compared with the clean image to test the denoising performance.

The results are shown in the following table. The PSNR and the SSIM values shown here are average over 10,000 test images.

We can see from Table 4.8 the improvement is significant. Usually, in image denoising literature a PSNR improvement of 0.5 dB to 1 dB is considered to be good. In this case, the improvement is near about 3dB compared to the sparse denoising autoencoder. Also, the improvement in SSIM is around 0.1 – this is a huge improvement. For visual evaluation, some sample images are shown in Fig 4.4.

The denoising results may not be at par with the state-of-the-art like BM3D [40] or KSVD but are better than competing autoencoder based techniques. The proposed SparseConnect autoencoder gets the best denoising results, balancing noise and sharpness.

Noise Variance, PSNR, and SSIM for Noisy Image	**SDAE**	**Sparse SDAE**	**Proposed *l_1-penalty***	**Proposed *l_0-penalty***
Variance=0.01 PSNR=19.96 SSIM=0.5691	PSNR=21.96 SSIM=0.6315	PSNR=22.94 SSIM=0.6803	PSNR=23.90 SSIM=0.7238	**PSNR=26.03 SSIM=0.8052**
Variance=0.04 PSNR= 14.01 SSIM=0.3256	PSNR=21.67 SSIM=0.6167	PSNR=22.63 SSIM=0.6667	PSNR=23.53 SSIM=0.7027	**PSNR=25.68 SSIM=0.7928**
Variance=0.09 PSNR= 10.49 SSIM=0.2011	PSNR=21.31 SSIM=0.5973	PSNR=22.25 SSIM=0.6478	PSNR=23.01 SSIM=0.6725	**PSNR=25.27 SSIM=0.7779**

Table 4.8: Denoising Results

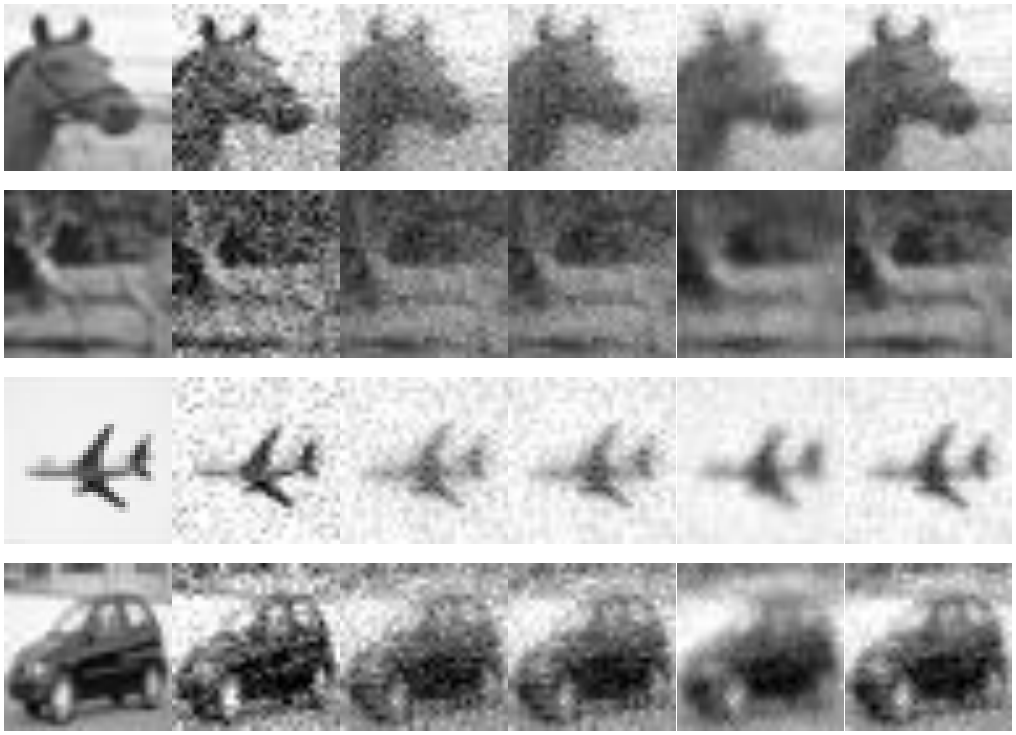


Figure 4.4: Left to Right – Original test image, noisy image, SDAE, Sparse Autoencoder, Proposed *l_1-norm* and Proposed *l_0-norm.*

CHAPTER 5

Proposed Formulation -II

5. Rank Deficiency in Weights of Autoencoder

5.1. Proposed Formulation

This method also aims at modeling the redundancy in the network of the Autoencoders just like the last method. Instead of making the connections sparse we try to explore the linear dependency in the connections. We will progress similarly as we did in the last formulation. We propose to learn the encoder and decoder weights such that they are of low-rank. We also use linear activation function in the hidden layers. In [26], it was shown that for a single layer autoencoder, even with a non-linear activation function, the network mostly operates in the linear region and which has been established with results in section 4.2.1. Following this work, we propose to use linear activation for our autoencoder. Since we learn deeper architectures in a greedy fashion, the conclusions of [26] are therefore extendable to stacked autoencoders. Linear activation functions simplify the derivation of the algorithm and speeds of implementation.

In the simplest scenario, we can achieve low-rank weights using existing tools. We can train an autoencoder by solving basic formulation: $\min_{W',W} \|X - W'WX\|_F^2$ and then have a low-rank approximation of the weights by keeping the high valued singular values and the corresponding singular vectors of W and W'. This would be the low-rank equivalent of Optimal Brain Damage [17]. The low rank weights are more practical than sparse weights because no weights will be dropped and correlation within the data is also captured effectively. But such a method would be very rudimentary. A more elegant solution would be introducing low-rank penalty into the autoencoder formulation:

$$\arg\min_{W,W'} \|X - W'WX\|_F^2 + \lambda\left(\|W\|_* + \|W'\|_*\right) \tag{5.1}$$

Here $\|.\|_*$ denotes the nuclear norm and is defined as the sum of singular values. Nuclear norm is the closest convex envelope of matrix rank. We solve this nuclear norm formulation by applying Majorization-Minimization Technique followed by Singular Value Shrinkage [41]. We will skip the derivation and discussion on Majorization-Minimization Technique as it has been discussed in the last chapter under section 4.2.1.

We directly derive a solution for nuclear norm formulation. We first apply Majorization-Minimization Technique which just affects the base cost function part and thus we can write (5.1) as:

$$\arg\min_{W,W'} \|B - W'W\|_F^2 + \lambda\left(\|W\|_* + \|W'\|_*\right) \tag{5.2}$$

This is again a bilinear problem and we propose to solve it alternately, i.e. we fix *W'* and solve *W* and then solve *W'* assuming *W* is fixed. These two steps are done in every iteration.

$$W_{k+1} = \arg\min_{W} \|B - W_k'W\|_F^2 + \frac{\lambda}{a}\|W\|_* \tag{5.3}$$

$$W'_{k+1} = \arg\min_{W'} \|B - W'W_{k+1}\|_F^2 + \frac{\lambda}{a}\|W'\|_* \tag{5.4}$$

In both cases, the problem remains the same – that of the least squares minimization with low rank -norm penalty.

To solve both (5.3) and (5.4) using Singular Value Shrinkage Algorithm.
Since SVS is not a standard technique like CG, we briefly describe it by solving (5.5) in Algorithm 3.

$$\arg\min_{X} \|y - Ax\|_F^2 + \lambda\|X\|_* \tag{5.5}$$

Where x=vec(X).

We have already described the training and testing phases but for convenience, we describe it again. Algorithm 4 shows the training phase for the proposed method. The process is unsupervised. The testing phase is just multiplication of matrices hence very efficient and fast when we have a generalized network and want to get quick results.

Let Z be the Testing data. Then the features space of testing data (Z_1) will be:

$$Z_1 = WZ \tag{5.6}$$

And the recovered testing data (Z_2) will be given by:

$$Z_2 = W'WZ \tag{5.7}$$

For making a deeper network, we have adopted a greedy layer-wise training approach. The latent representation of the outermost layer is used to train the next deeper layer and so on we can train other deeper layers as per the performance at every layer. We employ the same penalty in each layer when building a Deep Network. We generally test the performance of the network at every layer and stop if there is a decrement in the performance or there is no significant improvement in the performance by going deeper into the network.

$Init: x_0 = \arg\min_x \|y - Ax\|_F^2$

Iterate till convergence

Landweber: $b = x_{k-1} + \frac{1}{\alpha} A^T (y - Ax_{k-1}), \alpha = \text{maxEigen}(\text{A}^{\text{T}}\,\text{A})$

Format: $B = mat(b)$

Compute SVD: $USV^T = B$

Shrink: $\Sigma = signum(S)\max\left(0, |S| - \frac{\lambda}{2\alpha}\right)$

Update: $X_k = U\Sigma V^T$

Algorithm 3: SVS Algorithm

The whole process is compactly shown in Algorithm 4.

$$Initialisation:$$

$$H = \arg\min_{H} \left\| X^T - X^T H \right\|_F^2$$

$$H = USV^T$$

$$W_0^{'} = US \; and \; W_0 = V^T$$

$$In \; every \; iteration \; 'k'$$

$$compute \; B^T = H_k^T + (\frac{1}{\alpha}) X (X^T - X^T H_k)$$

$$update \; W_{k+1} = \arg\min_{W} \left\| B - W_k^{'} W \right\|_F^2 + \frac{\lambda}{\alpha} \left\| W \right\|_*$$

$$Apply \; SVS(A\lg.3)$$

$$similarly \; update \; W^{'}_{k+1} = \arg\min_{W^{'}} \left\| B - W^{'} W_{k+1} \right\|_F^2 + \frac{\lambda}{\alpha} \left\| W \right\|_*$$

$$Apply \; SVS(A\lg.3)$$

Algorithm 4: Proposed *low-rank Penalty on weights*

5.2. Experimental Setup and Results

We have already discussed the various datasets used for comparing our proposed methods with state-of-the-art methods. Without elaborating on them further, we will discuss the results and establish the superiority of this proposed method. The format of the results is same as for the Sparsely Connected Autoencoders. We have compared with SAE, TRAE, and DBN [35]. The nodes in the architecture vary as 392 (1st layer)-198 (2nd layer)-92 (3rd layer). We test with three classifiers – KNN (K=1), SRC and SVM (RBF kernel – 5 fold cross-validation); shown in Tables 5.1, 5.2 and 5.3 respectively. We report the results for the deepest layer i.e. is the 3rd layer here. As before, the representation from the deepest layer is used as features. In each of the tables, the best results are shown in bold.

5.2.1. Classification Performance

Dataset	SAE	TRAE	DBN	Proposed *low rank-penalty on weights*
MNIST	**97.33**	82.83	97.05	97.12
basic	95.25	79.92	**95.37**	95.14
basic-rot	84.83	58.56	84.71	**84.91**
bg-rand	86.42	65.61	86.36	**86.77**
bg-img	77.16	58.51	77.16	**77.39**
bg-img-rot	52.21	27.10	50.47	**52.40**

Table 5.1: Classification Accuracy (%) - KNN (K=1) Results

The results in Table 5.1, 5.2 and 5.3 shows that our proposed method yields better results than SAE and DBN on an average; in each of the tables, our method yields the best results on four or more datasets. Others perform better only for the simplest datasets (MNIST and basic). The results from the simple SRC yields by far the best results.

Dataset	SAE	TRAE	DBN	Proposed *low rank-penalty on weights*
MNIST	**98.33**	87.19	98.26	98.20
basic	96.91	85.03	96.80	**96.97**
basic-rot	90.04	68.63	89.04	**90.23**
bg-rand	91.03	72.25	88.49	**91.61**
bg-img	84.14	65.68	83.72	**84.67**
bg-img-rot	62.46	34.01	55.89	**63.27**

Table 5.2: Classification Accuracy (%) - SRC Results

Dataset	SAE	TRAE	DBN	Proposed *low rank-penalty on weights*
MNIST	98.50	88.74	**98.53**	98.51
basic	96.96	86.61	97.07	**97.13**
basic-rot	89.43	72.54	89.05	**89.55**
bg-rand	91.28	75.20	89.59	**91.52**
bg-img	84.86	68.76	**85.46**	85.37
bg-img-rot	60.53	40.97	58.25	**61.70**

Table 5.3: Classification Accuracy (%) - SVM Results

In Table 5.4 we compare the results with stacked denoising autoencoder (SDAE), deep belief network (DBN) from [6], sparse deep neural network (SDNN) [36] and optimal brain damage (OBD) [17]. These are sophisticated techniques with pre-training and fine-tuning with neural networks. We do not think it is fair to compare our simple classification algorithms with such sophisticated techniques. But we find that even with a simple SRC; our results are comparable with these techniques.

Dataset	SDAE	DBN	SDNN	OBD	Proposed *low rank-penalty on weights*
MNIST	98.72	**98.76**	98.57	97.99	98.20
basic	**97.16**	96.89	96.69	95.41	96.97
basic-rot	**90.47**	89.70	89.58	87.89	90.23
bg-rand	89.70	**93.27**	90.21	88.27	91.61
bg-img	83.32	83.69	82.96	80.65	**84.67**
bg-img-rot	56.24	52.61	51.63	50.19	**63.27**

Table 5.4: Classification Accuracy (%) - Comparative Results

Next, in Table 5.5 we show the training times of different algorithms. The results show that our proposed method is the fastest of all; even though we need to compute SVD in every iteration. This can be primarily attributed to the early convergence of our proposed method.

The details about the configuration of the system used for simulations are mentioned in the last chapter.

Dataset	**MNIST**	**basic**
SAE (linear)	111	6
SAE(non-linear)	251	53
TRAE	98	24
DBN	78	21
Proposed *low rank-penalty on weights*	**24**	**5**

Table 5.5: Training Times in minutes

5.2.2. Denoising Performance

For comparison, we use the standard metrics for image quality assessment - PSNR (Peak Signal to Noise Ratio) and SSIM (Structural Similarity Index) [39]. We compare our approach Low Rank with the sparse autoencoder [11] and stacked denoising autoencoder (SDAE). We use single layer autoencoders for image denoising. The number of nodes in the hidden layer is kept to be 512. The value of λ is 0.001.

The Denoising dataset used is also explained in chapter 3. The noisy images served as the input to the autoencoders and the clean images were the output. For testing, the noisy test images were as inputs and the image obtained at the output was compared with the clean image to test the denoising performance.

The results are shown in the following table. The PSNR and the SSIM values shown here are average over 10,000 test images.

Noise Variance, PSNR and SSIM for Noisy Image	**SDAE**	**Sparse SDAE**	**Proposed *low* rank-*penalty on weights***
Variance=0.01 PSNR=19.96 SSIM=0.5691	PSNR=21.96 SSIM=0.6315	PSNR=22.94 SSIM=0.6803	**PSNR=23.94 SSIM=0.7300**
Variance=0.04 PSNR= 14.01 SSIM=0.3256	PSNR=21.67 SSIM=0.6167	PSNR=22.63 SSIM=0.6667	**PSNR=23.69 SSIM=0.7184**
Variance=0.09 PSNR= 10.49 SSIM=0.2011	PSNR=21.31 SSIM=0.5973	PSNR=22.25 SSIM=0.6478	**PSNR=23.31 SSIM=0.7000**

Table 5.6: Denoising Results

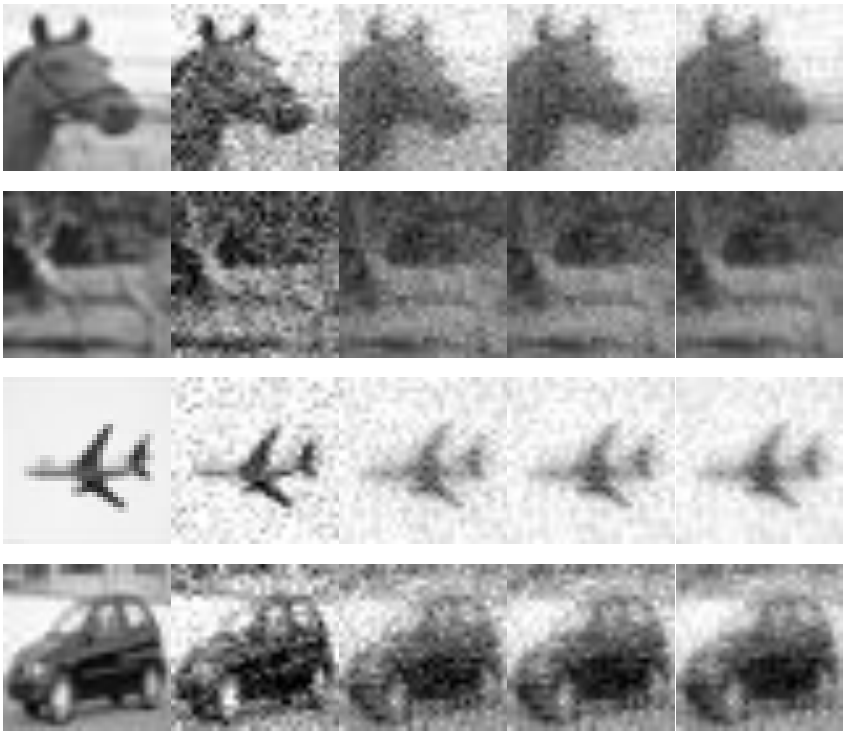


Figure 5.1: Left to Right – Original test image, noisy image, SDAE, Sparse Autoencoder, Proposed low-rank penalty on weights

Results show an improvement of around 2dB in the PSNR which is quite significant. We can also see a huge improvement in terms of SSIM. The results are more pronounced in the case of more noisy images. For visual evaluation, some sample images are shown in Fig 5.1.

PART-II

MODELING REDUNDANCY IN THE REPRESENTATION

CHAPTER 6

Proposed Formulation -III

6. Rank Deficiency within Classes

6.1. Proposed Formulation

In this part of the thesis, we try to learn features such that they are linearly dependent within the same class motivated from the belief that the samples belonging to the same class will have similar signatures and will exhibit same characteristics in the features. This gives rise to the fact that the features of the samples belonging to the same class are linearly dependent. Hence, features from the same class when stacked as columns of a matrix will lead to a low-rank matrix. This can be easily introduced into the classical optimization framework of an autoencoder by just adding a regularization term to promote low-rank constraint on the feature matrix. However optimizing the rank problem is NP-hard with double exponential complexity. Signal Processing literature showed that one can substitute the rank by its nearest convex surrogate - the nuclear norm [42], [43]. Following these studies, we propose to add a nuclear norm (on features) penalty on the traditional autoencoder optimization problem.

In [26] it was shown that most of the times autoencoders operate in the linear region, even if the activation function is non-linear and the utility of using linear activation over non-linear activation has been proved with results in chapter 4. Following this, we use a linear activation function (Identity) which radically simplifies the implementation. We have discussed a lot of variations to the basic formulation. One variation that exists in literature and works on feature level is discussed in CAE [13]. CAE encourage robustness of the representation of the data by penalizing the feature representation with Frobenius norm. CAE for linear activation functions can be formulated as:

$$\arg\min_{W,W'} \| X - W'WX \|_F^2 + \lambda \|WX\|_F^2 \tag{6.1}$$

Our proposed formulation for a single layer autoencoder is yet another regularized version of basic Autoencoder formulation:

$$\arg\min_{W',W} \sum_i \left\| X_i - W'WX_i \right\|_F^2 + \lambda \left\| WX_i \right\|_* \tag{6.2}$$

where W is the encoding weight matrix and W' is the decoding weight matrix. The penalty $\sum_i \|WX\|_*$ implies that the feature matrix for the i^{th} class is low-rank and hence will have a small nuclear norm. For sake of convenience, we can write (6.2) as

$$\arg\min_{W',W} \sum_i \left\| X_i^T - X_i^T W^T W'^T \right\|_F^2 + \lambda \left\| X_i^T W^T \right\|_* \tag{6.3}$$

In (6.2) we substitute $Z_i = X_i^T W^T$; this leads to:

$$\arg\min_{W,W'} \sum_i \left\| X_i^T - Z_i W'^T \right\|_F^2 + \lambda \left\| Z_i \right\|_* s.t.\ Z_i = X_i^T W^T \tag{6.4}$$

Forming the Lagrangian for (6.3) and solving it, is not efficient. The Lagrangian enforces equality at every iteration; this is overkill. One only needs the equality at convergence. In recent times, this is achieved by formulating the Augmented Lagrangian instead:

$$\arg\min_{W,W',Z_i} \sum_i \left\| X_i^T - Z_i W'^T \right\|_F^2 + \lambda \left\| Z_i \right\|_* + \mu \left\| X_i W^T - Z_i \right\|_F^2 \tag{6.5}$$

In this formulation, one starts with a small value of μ, solves (6.5) and then increases the value of μ progressively and solving (6.5) for each of its values. This is not efficient in general. The heating of μ is heuristic and solving an iterative problem for every value of μ is time-consuming. In recent times, the Split Bregman approach [44] offers an elegant solution to this issue. One only needs to introduce a Bregman relaxation variable in (6.5) as:

$$\arg\min_{W_1,W_2,Z_i} \sum_i \left\| X_i^T - Z_i W'^T \right\|_F^2 + \lambda \left\| Z_i \right\|_* + \mu \left\| X_i^T W^T - Z_i - B_i \right\|_F^2 \tag{6.6}$$

where B is the Bregman relaxation variable. The relaxation is progressively updated in every iteration by a gradient descent step. Hence one does not need to tamper with the value of μ; one can start with a reasonable value of μ and the rest (inequality at the initial stage and equality during convergence) will be handled automatically by the relaxation variable.

The final formulation (6.6) can be solved using alternating minimization. This decomposes (6.6) into the following sub-problems.

$$P_1 : \arg\min_{W'} \sum_i \left\| X_i^T - Z_i W'^T \right\|_F^2 \tag{6.7}$$

$$P_2 : \arg\min_{W} \sum_i \left\| X_i^T W^T - Z_i - B_i \right\|_F^2 \tag{6.8}$$

$$P_3 : \arg\min_{Z_i} \sum_i \left\| X_i^T - Z_i W'^T \right\|_F^2 + +\lambda \left\| Z_i \right\|_* + \mu \left\| X_i^T W^T - Z_i - B_i \right\|_F^2 \tag{6.9}$$

Both P_1 and P2 can be recast as,

$$\arg\min_{W'} \left\| X^T - Z W'^T \right\|_F^2 \tag{6.10}$$

$$\arg\min_{W} \left\| X^T W^T - (Z + B) \right\|_F^2 \tag{6.11}$$

Where $Z=[Z_1|\ldots|Z_c]$ and $B=[B_1|\ldots|B_c]$ (assuming c classes).

Both (6.10) and (6.11) is a simple least squares problem with a closed form solution. However, in practice, it is usually solved using Conjugate Gradient (CG).

Solving P_3 is not straightforward as the previous ones. It can be recast as follows for each class:

$$\arg\min_{Z_i} \left\| \begin{pmatrix} X_i^T \\ \sqrt{\mu}(X_i^T W^T - B_i) \end{pmatrix} - Z_i \begin{pmatrix} W'^T \\ \sqrt{\mu} I \end{pmatrix} \right\|_F^2 + \lambda \left\| Z_i \right\|_* \tag{6.12}$$

This is a nuclear norm minimization problem. It can be efficiently solved using singular value shrinkage (SVS) which is described in Algorithm 3.

The final step is to update the Bregman relaxation variable. This is a simple gradient descent step.

$$B \leftarrow B - (X^T W^T - Z) \tag{6.13}$$

The problem we are solving is non-convex. Hence, it does not have any global minima. We continue the iterations till some local minima are reached, i.e. when the difference between the objective functions in successive iterations is smaller than some predefined tolerance.

The problem is also non-smooth. Hence it cannot be solved using simple back propagation (BP) type techniques. It needs to be solved using sub-gradient methods (shrinkage in our technique).

We have already described the training and testing phases but for convenience, we describe it once again. Algorithm 5 shows the training phase for the proposed method. This process is a supervised version of the Autoencoders. Since we find rank deficiency within the classes we have to provide labels to the network during the training phase. The testing phase is just multiplication of matrices hence very efficient and fast when we have a generalized network and want to get quick results.

Let Z be the Testing data. Then the features space of testing data (Z_1) will be:

$$Z_1 = WZ \tag{6.14}$$

And the recovered testing data (Z_2) will be given by:

$$Z_2 = W'WZ \tag{6.15}$$

For making a deeper network, we have adopted a greedy layer-wise training approach. The latent representation of the outermost layer is used to train the next deeper layer and so on we can train other deeper layers as per the performance at every layer. We employ the same penalty in each layer when building a Deep Network. We generally test the performance of the network at every layer and stop if there is a decrement in the performance or there is no significant improvement in the performance by going deeper into the network.

The whole training procedure for is compactly described in Algorithm 5.

6.2. Experimental Setup and Results

6.2.1. Classification Performance

We use the proposed nuclear norm regularized autoencoder for extracting features at each layer (ours is a greedy layer by layer training approach). As mentioned above the architecture remains same for every dataset and the representation from the deepest layer is used as features. We compare our results with the stacked autoencoder (SAE), Contractive Autoencoder (CAE) and Deep Belief Network (DBN). We have used the toolbox from [35] whose parameters were already tuned for the MNIST and ISOLET databases. The SAE and

Initialisation:

$$H = \arg\min_{H} \left\| X^T - X^T H \right\|_F^2$$

$$H = USV^T$$

$$W_0^{'} = US \; and \; W_0 = V^T$$

$$B_0 = U(0,1)$$

$$Z_0^T = X^T W_{(0)}^T$$

outer loop $'k'$

$$W_{(k)}^{'T} = \arg\min_{W'} \| X^T - Z_{k-1} W^{'T} \|_F^2$$

$$W_{(k)}^T = \arg\min_{W} \| X^T W^T - (Z_{k-1} + B_{k-1}) \|_F^2$$

inner loop $'i'$ *for classes*

$$Z_{i(k)}^T = \arg\min_{Z_i} \left\| \begin{pmatrix} X_i^T \\ \sqrt{\mu}(X_i^T W^T - B_i) \end{pmatrix} - Z_i \begin{pmatrix} W^{'T} \\ \sqrt{\mu} I \end{pmatrix} \right\|_F^2 + \lambda \left\| Z_i \right\|_*$$

Apply $SVS(A\text{lg}.3)$

end

$$B_k \leftarrow B_{k-1} - (X^T W^T - Z_k)$$

end

Algorithm 5: Proposed *low-rank Penalty on features*

CAE use linear activation. The nodes in the architecture vary as 392 (1st layer)-198 (2nd layer)-92 (3rd layer) for MNIST datasets , 180(1st layer)-100(2nd layer) for USPS, 512 (1st layer)-256 (2nd layer)-128 (3rd layer) for Bangla and Devanagari, and single layer architecture with 400 nodes for ISOLET . We report the results for the deepest layer. We test with three classifiers – KNN (K=1), SRC and SVM (RBF kernel – 5 fold cross-validation); results are shown in Tables 6.1, 6.2 and 6.3 respectively. We have also tabulated the results of Stacked Denoising Autoencoders (SDAE) and DBN from [6] for MNIST datasets. The SDAE and DBN were fine-tuned with neural network classifier. We do not think it is fair to compare such fine-tuned systems with simple classifiers; we show the results nevertheless in Table 6.4. We find that in several instances our proposed formulation yields better results than these benchmarks.

Dataset	**SAE**	**CAE**	**DBN**	**Proposed *low* rank-*penalty on features***
MNIST	97.33	97.17	97.05	**97.34**
basic	95.25	95.32	95.37	**95.68**
basic-rot	84.83	85.03	84.71	**86.32**
bg-rand	86.42	87.31	86.36	**88.28**
bg-img	77.16	77.35	77.16	**79.27**
bg-img-rot	52.21	52.24	50.47	**55.60**
USPS	94.92	95.32	95.01	**95.42**
Devanagari	91.15	94.97	94.39	**96.20**
Bangla	81.78	87.18	**89.00**	87.68
ISOLET	88.19	88.90	72.16	**90.19**

Table 6.1; Classification Accuracy (%) - KNN (K=1) Results

Dataset	**SAE**	**CAE**	**DBN**	**Proposed *low* rank-*penalty on features***
MNIST	98.33	98.37	98.26	**98.41**
basic	96.91	97.16	96.80	**97.18**
basic-rot	90.04	90.32	89.04	**91.05**
bg-rand	91.03	**92.40**	88.49	92.14
bg-img	84.14	85.32	83.72	**85.59**
bg-img-rot	62.46	63.51	55.89	**65.58**
USPS	96.06	**96.31**	95.71	96.11
Devanagari	93.70	96.78	96.41	**97.24**
Bangla	89.05	93.16	**93.48**	93.05
ISOLET	93.07	**93.84**	64.27	93.65

Table 6.2: Classification Accuracy (%) - SRC Results

Dataset	SAE	CAE	DBN	Proposed *low* rank-*penalty on features*
MNIST	98.50	95.40	98.53	**98.61**
basic	96.96	93.03	97.07	**97.25**
basic-rot	89.43	62.24	89.05	**90.23**
bg-rand	91.28	85.34	89.59	**92.31**
bg-img	84.86	81.27	85.46	**86.16**
bg-img-rot	60.53	43.19	58.25	**63.41**
USPS	95.41	95.07	95.62	**95.96**
Devanagari	95.43	93.33	96.81	**97.58**
Bangla	89.83	88.98	**95.18**	93.50
ISOLET	**96.66**	96.41	76.84	96.60

Table 6.3: Classification Accuracy (%) - SVM Results

Dataset	SDAE	DBN	Proposed *low* rank-*penalty on features*
MNIST	98.72	**98.76**	98.41
basic	97.16	96.89	**97.18**
basic-rot	90.47	89.70	**91.05**
bg-rand	89.70	**93.27**	92.14
bg-img	83.32	83.69	**85.59**
bg-img-rot	56.24	52.61	**65.58**

Table 6.4: Classification Accuracy (%) - Comparative Results

For all the classifiers, we see for almost every instance our method yields the best results. In the case of SRC, CAE yields a slightly better result for a few instances. Our proposed method yields even better results than the fine-tuned techniques SDAE and DBN with neural network classification; ours yields only slightly worse results for the larger MNIST dataset, but for the more challenging datasets we consistently perform better(apart for bg-rand). As for the complexity of the algorithm it depends on the number of classes present in the data.

6.3. Comparison of Proposed Methods

In this section we compare the various results from all the methods proposed in the thesis. We see that low-rank penalty on representation gives the best results for classification which was expected since we provided extra label information to the training algorithm. The results are more pronounced for difficult datasets. We get best results on Gaussian Denoising for l_0-penalty formulation.

Dataset	**Proposed *l1-penalty***	**Proposed *l0-penalty***	**Proposed *low rank-penalty on weights***	**Proposed *low* rank-*penalty on features***
MNIST	95.91	97.21	97.12	**97.34**
basic	92.49	95.39	95.14	**95.68**
basic-rot	81.01	85.14	84.91	**86.32**
bg-rand	68.87	86.88	86.77	**88.28**
bg-img	79.84	77.22	77.39	**79.27**
bg-img-rot	38.91	51.80	52.40	**55.60**

Table 6.5: Classification Accuracy (%) - KNN Results

Dataset	**Proposed *l1-penalty***	**Proposed *l0-penalty***	**Proposed *low rank-penalty on weights***	**Proposed *low* rank-*penalty on features***
MNIST	97.16	**98.42**	98.20	98.41
basic	95.43	97.03	96.97	**97.18**
basic-rot	87.76	90.19	90.23	**91.05**
bg-rand	76.17	91.69	91.61	**92.14**
bg-img	85.84	85.11	84.67	**85.59**
bg-img-rot	47.75	62.61	63.27	**65.58**

Table 6.6: Classification Accuracy (%) - SRC Results

Dataset	**Proposed** ***l1-penalty***	**Proposed** ***l0-penalty***	**Proposed** ***low rank-penalty on weights***	**Proposed** ***low* rank-*penalty on features***
MNIST	97.70	98.60	98.51	**98.61**
basic	95.70	97.12	97.13	**97.25**
basic-rot	87.07	89.22	89.55	**90.23**
bg-rand	87.04	91.73	91.52	**92.31**
bg-img	78.01	85.35	85.37	**86.16**
bg-img-rot	46.30	60.91	61.70	**63.41**

Table 6.7: Classification Accuracy (%) - SVM Results

Noise Variance, PSNR, and SSIM for Noisy Image	**Proposed** l_1***-penalty***	**Proposed** l_0***-penalty***	**Proposed** ***low* rank-*penalty on weights***
Variance=0.01 PSNR=19.96 SSIM=0.5691	PSNR=23.90 SSIM=0.7238	**PSNR=26.03 SSIM=0.8052**	PSNR=23.94 SSIM=0.7300
Variance=0.04 PSNR= 14.01 SSIM=0.3256	PSNR=23.53 SSIM=0.7027	**PSNR=25.68 SSIM=0.7928**	PSNR=23.69 SSIM=0.7184
Variance=0.09 PSNR= 10.49 SSIM=0.2011	PSNR=23.01 SSIM=0.6725	**PSNR=25.27 SSIM=0.7779**	PSNR=23.31 SSIM=0.7000

Table 6.8: Denoising Results

Conclusion

With this work, we propose different variations of the Autoencoder network by imposing different regularization penalties on the basic Autoencoder cost function. By doing this we can regulate the learning capacity of the network and hence we improve the performance of the network. In the first part, we worked on the connections of the Autoencoder network. We employed sparsity and rank deficiency on the weights of the network. In the second part we concentrated on finding dependency in the features of the samples and this we modeled by employing rank deficiency within classes. Although there are several studies on sparsely connected neural networks and variations of Autoencoders, there is no prior study on sparsely connected autoencoder and low rank weights for Autoencoders. This is the first work in that respect. Also, this is the first work exploring rank deficiency in the representation through Autoencoders.

Experiments were carried out for two benchmark autoencoder tasks – classification and denoising for different datasets. For classification, we employ stacked layer architecture. The representation from the deepest layer is used for classification. We observed going deeper into the network does help classification task. The comparison is made with the SAE, TRAE, CAE, DBN, SDAE, Sparse Deep Neural Network (SDNN) and optimal brain damage (OBD) throughout the thesis. Under fair comparison (when non-parametric-KNN andSRC classifiers and SVM) our methods outperform others. Even with fine-tuned neural network architectures, our proposed approach yields better results than SDNN and OBD and perform at par with densely connected networks like SDAE and DBN. For denoising, we compared against the denoising autoencoder and the sparse autoencoder. Even for this task, our proposed approaches yield considerably better results.

The performance of our network on naïve classifiers such as KNN, SRC, and SVM can be attributed to the defined feature set obtained from our networks. Through this work, we were able to design machine learning networks which can learn salient characteristics of the data efficiently.

Bibliography

[1] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1-127.

[2] Yuan, Y., Mou, L., & Lu, X. (2015). Scene recognition by manifold regularized deep learning architecture. *IEEE transactions on neural networks and learning systems*, *26*(10), 2222-2233.

[3] Goh, H., Thome, N., Cord, M., & Lim, J. H. (2014). Learning deep hierarchical visual feature coding. *IEEE transactions on neural networks and learning systems*, *25*(12), 2212-2225.

[4] Chen, K., & Salman, A. (2011). Learning speaker-specific characteristics with a deep neural architecture. *IEEE Transactions on Neural Networks*, *22*(11), 1744-1756.

[5] Chen, D., & Mak, B. K. W. (2015). Multitask learning of deep neural networks for low-resource speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(7), 1172-1183.

[6] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, *11*(Dec), 3371-3408.

[7] Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, *19*, 153.

[8] Jarrett, K., Kavukcuoglu, K., & Lecun, Y. (2009, September). What is the best multi-stage architecture for object recognition?. In *2009 IEEE 12th International Conference on Computer Vision* (pp. 2146-2153). IEEE.

[9] Bianchini, M., & Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, *25*(8), 1553-1565.

[10] Ng, A. (2011). Sparse autoencoder. *CS294A Lecture notes*, *72*, 1-19.

[11] Makhzani, A., & Frey, B. (2013). k-Sparse Autoencoders. *arXiv preprint arXiv:1312.5663*.

[12] Cho, K. (2013, June). Simple sparsification improves sparse denoising autoencoders in denoising highly noisy images. In *International Conference on Machine Learning*.

[13] Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 833-840).

[14] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.

[15] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., & Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (pp. 1058-1066).

[16] Frazão, X., & Alexandre, L. A. (2014, October). DropAll: Generalization of Two Convolutional Neural Network Regularization Methods. In *International Conference Image Analysis and Recognition* (pp. 282-289). Springer International Publishing.

[17] LeCun, Y., Denker, J. S., Solla, S. A., Howard, R. E., & Jackel, L. D. (1989, November). Optimal brain damage. In *NIPs* (Vol. 2, pp. 598-605).

[18] Buades, A., Coll, B., & Morel, J. M. (2005, June). A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 2, pp. 60-65). IEEE.

[19] Awate, S. P., & Whitaker, R. T. (2006). Unsupervised, information-theoretic, adaptive image filtering for image restoration. *IEEE Transactions on pattern analysis and machine intelligence*, *28*(3), 364-376.

[20] Tang, K., Liu, R., Su, Z., & Zhang, J. (2014). Structure-constrained low-rank representation. Neural Networks and Learning Systems, IEEE Transactions on,25(12), 2167-2179.

[21] Zhou, P., Lin, Z., & Zhang, C. (2016). Integrated Low-Rank-Based Discriminative Feature Learning for Recognition. IEEE transactions on neural networks and learning systems, 27(5), 1080-1093.

[22] Chen, M., Weinberger, K. Q., Sha, F., & Bengio, Y. (2014). Marginalized Denoising Auto-encoders for Nonlinear Representations. In *ICML* (pp. 1476-1484).

[23] Engan, K., Aase, S. O., & Husoy, J. H. (1999). Method of optimal directions for frame design. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on* (Vol. 5, pp. 2443-2446). IEEE.

[24] Aharon, M., Elad, M., & Bruckstein, A. (2006). K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on signal processing*, *54*(11), 4311-4322.

[25] Rubinstein, R., Peleg, T., & Elad, M. (2013). Analysis K-SVD: a dictionary-learning algorithm for the analysis sparse model. *IEEE Transactions on Signal Processing*, *61*(3), 661-677.

[26] Abbas, H. M. (2004). Analysis and pruning of nonlinear auto-association networks. *IEE Proceedings-Vision, Image and Signal Processing*, *151*(1), 44-50.

[27] Thom, M., & Palm, G. (2013). Sparse activity and sparse connectivity in supervised learning. *Journal of Machine Learning Research*, *14*(Apr), 1091-1143.

[28] Gripon, V., & Berrou, C. (2011). Sparse neural networks with large learning diversity. *IEEE transactions on neural networks*, *22*(7), 1087-1096.

[29] Sparse Signal Restoration: cnx.org/content/m32168/latest/

[30] Pati, Y. C., Rezaiifar, R., & Krishnaprasad, P. S. (1993, November). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on* (pp. 40-44). IEEE.

[31] Tropp, J. A., & Gilbert, A. C. (2007). Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on information theory*, *53*(12), 4655-4666.

[32] Blumensath, T., & Davies, M. E. (2008). Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, *14*(5-6), 629-654.

[33] www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html

[34] Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., & Ma, Y. (2009). Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, *31*(2), 210-227.

[35] http://ceit.aut.ac.ir/~keyvanrad/DeeBNet%20Toolbox.html

[36] http://www.mathworks.in/matlabcentral/fileexchange/42853-deep-neural-network

[37] Rodríguez, P., & Wohlberg, B. (2009). Efficient minimization method for a generalized total variation functional. *IEEE Transactions on Image Processing*, *18*(2), 322-332.

[38] Wang, S., Liu, Q., Xia, Y., Dong, P., Luo, J., Huang, Q., & Feng, D. D. (2013). Dictionary learning based impulse noise removal via L1–L1 minimization.*Signal Processing*, *93*(9), 2696-2708.

[39] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, *13*(4), 600-612.

[40] Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. (2009, April). BM3D image denoising with shape-adaptive principal component analysis. In *SPARS'09-Signal Processing with Adaptive Sparse Structured Representations*.

[41] Majumdar, A., & Ward, R. K. (2011). Some empirical advances in matrix completion. *Signal Processing*, *91*(5), 1334-1338.

[42] Recht, B., Xu, W., & Hassibi, B. (2011). Null space conditions and thresholds for rank minimization. *Mathematical programming*, *127*(1), 175-202.

[43] Recht, B., Fazel, M., & Parrilo, P. A. (2010). Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, *52*(3), 471-501.

[44] Yin, W., Osher, S., Goldfarb, D., & Darbon, J. (2008). Bregman iterative algorithms for \ell_1-minimization with applications to compressed sensing.*SIAM Journal on Imaging sciences*, *1*(1), 143-168.

Publications

Accepted

[1] Gupta, K., & Majumdar, A. (2016). Sparsely Connected Autoencoders. In *International Joint Conference on Neural Network (IJCNN'16).*

[2] Mehta, J., Gupta, K., Gogna, A.,& Majumdar, A. (2016). Stacked Robust Autoencoders for Classification. In *International Conference on Neural Information Processing (ICONIP'16).*

[3] Gupta, K., Raj, A., & Majumdar, A. (2016). Analysis and Synthesis Prior Greedy Algorithms for Non-linear Sparse Recovery. In *Proceedings of Data Compression Conference (DCC'16).*

[4] Gupta, K., & Majumdar, A. (2016). Greedy Algorithms for Non-linear Sparse Recovery. In *Machine Intelligence and Signal Processing* (pp. 99-108). Springer India.

Submitted

[1] Gupta, K., & Majumdar, A. Low Rank Weights for Autoencoder. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'17).*

[2] *Gupta, K., Mehta, J., Gogna, A., & Majumdar, A. Enhancing Latent Factor Model with Neighborhood Information for Recommender System Design. *IEEE Transactions on Knowledge and Data Engineering.*

[3] Gupta, K., & Majumdar, A. Nuclear Norm Regularized Stacked Autoencoder for Classification. In *International Joint Conference on Neural Network (IJCNN'17).*