UNIVERSITY OF CALIFORNIA, SAN DIEGO

An Efficient Online Feature Extraction Algorithm for Neural Networks

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Pouya Bozorgmehr

Committee in charge:

Professor Garrison W. Cottrell, Chair
Professor Sanjoy Dasgupta
Professor Lawrence K. Saul

2009

The thesis of Pouya Bozorgmehr is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

DEDICATION

To my family and friends who gave their love, support, and time freely.

EPIGRAPH

True knowledge exists in knowing that you know nothing.

—Socrates

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor Gary Cottrell for devoting so much time and teaching me a great deal on neural networks and research.

VITA

2009	Master of Science in Computer Science University of California, San Diego San Diego, CA
2004	Bachelor of Science in Computer Engineering University of California, San Diego San Diego, CA

ABSTRACT OF THE THESIS

An Efficient Online Feature Extraction Algorithm for Neural Networks

by

Pouya Bozorgmehr
Master of Science in Computer Science
University of California San Diego, 2009
Professor Garrison W. Cottrell, Chair

Finding optimal feature sets for classification tasks is still a fundamental and challenging problem in the area of machine learning. The human visual system performs classification tasks effortlessly using its hierarchical features and efficient coding in its visual pathway. It is shown that early in the visual system the information is encoded using distributed coding schemes and later in the visual system the sparse coding is utilized. We propose a biologically motivated method to extract features that encode the information according to a specific activation profile. We show how our model much like the visual system, can learn distributed coding in lower layers and sparse coding in higher layers in an online manner. Online feature extraction is used in biometrics, machine vision, and pattern recognition. Methods that can dynamically extract features and perform online classification are especially important for real-world applications. We introduce online algorithms that are fast and efficient in extracting features for encoding and discriminating the input space. We also show a supervised version of this algorithm that performs feature selection and extraction in alternating steps to achieve a fast convergence and high accuracy.

Chapter 1

Introduction

Feature extraction has been a topic of great interest in the machine learning community. Features are used in computer vision, pattern recognition and other classification tasks, where the input data lives in a very high-dimensional space. Conventional classifiers cannot work properly in high-dimensional spaces, so high-dimensional data has to first be converted into a low-dimensional feature space. This is called dimensionality reduction or feature extraction.

Many feature extraction methods have been proposed that reduce the dimensionality of the input data. *Principal component analysis* (PCA) is an example of dimensionality reduction method, where it extracts features that capture most of the variation in the data-set. In face images PCA results in features known as eigenfaces [21]. Another commonly used method is *Linear Discriminant Analysis* (LDA) that results in features known as fisherfaces [3].

Most feature extraction methods assume access to the whole or big chunks of training data. However, in real-world applications, we do not have access to the whole data set and the input is streaming in smaller chunks and is constantly changing. This requires online feature extraction methods. Online feature extraction has been used in biometrics, machine vision, and pattern recognition. Methods that can dynamically extract features and perform online classification are especially important for real-world applications.

Previous incremental learning methods include *Incremental Principal Component Analysis* (IPCA) [12], *Incremental Discriminant Analysis* (ILDA) [17], and

Incremental Weighted Average Samples (IWAS) [20] have been used to update the feature set in an online manner.

We introduce an incremental algorithm that is fast, efficient, and flexible in extracting features for encoding and discriminating the input space. This work addresses two related problems. The first problem is the feature extraction in an unsupervised setting where there are no labels associated with the input data. The second problem is supervised learning where there are labels associated with the input data. This supervised version of our algorithm uses the class label to influence the feature extraction step. We first introduce our learning algorithm for unsupervised setting and show how it can be easily modified to generate more discriminative features in a supervised setting.

Feature extraction (FE) can be defined as a way to find a transform W into the feature space $\mathscr{Y}$ that would be useful for different tasks in machine learning. Feature extraction or dimensionality reduction are used in classification problems when the dimensionality of class space is very high. Our algorithm extracts features that result in different ways of encoding the input data. It can efficiently extract features that encode the data in a distributed way to tackle the high dimensional data that is coming from the real world, very similar to how the first layers of human visual system handle the high-diminsional inputs from our retina. Feature extraction methods like PCA try to extract features that capture most of the variance in the data. There is also a class of methods called Infomax methods used for feature extraction where the model tries to maximize the information that it encodes. An example of an Infomax method is *Independent Component Analysis* (ICA) where it tries to maximize the encoded information by minimizing the reconstruction error. Reconstruction error is the difference between the real data and the generated data by the model. These methods are called generative models. Features extracted in an attempt to model the data may not necessarily be good for classification. Other methods try to optimize the model of data given the class labels. These are called discriminative models. Our method however attempts to affect the way the information is encoded while minimizing the loss of information. For example, our method can learn features to encode data in a

distributed or sparse code.

It is important to be able to encode information in various forms for different tasks. For example, as mentioned before, in early stages of human visual system, because of high-dimensionality and large variability in the input, the information is encoded in a distributed way. Distributed coding means that input data can be expressed as a combination of many features as oppose to sparse coding where only a few features account for most of the input structure. It is important to encode as much of the relevant information without too much loss in accuracy. This is done by using distributed coding of information. However, in higher layers the accuracy in recognition of more abstract representation becomes important so the information becomes encoded using sparse coding. V1 is the first layer of visual system that takes its input from the retina and the lateral geniculate nucleus (LGN). It is in charge of extracting features like blobs and edges of different orientations. V2 gets its input from V1 but is similar to the V1 layer; however, the information is coded more compactly. In another words, the encoding is sparser in V2 that it is in V1.

In an experiment, we show how different levels of sparsity can be learned in our hierarchical model. Our model consist of a layer of logistic basis functions that represents our V1 feature layer with full connectivity to the input and V2 complex feature layer. Feature values are calculated using a logistic function of linear combinations of weight vectors and data vectors giving values in the range of $[0, 1]$.

$$f_i = \frac{1}{1 + e^{-(W_i X)}} \tag{1.1}$$

This is also called the activation function in the neural network literature. Throughout our experiments we use this activation function for our models.

In higher cortical areas we need to make decisions and perform classification from the extracted features of our sensory input. To improve the classification our brain filters irrelevant information, and only uses discriminative information. In a supervised task, we would like to select features from lower layers that do well to discriminate one class from other classes. This is called feature selection (FS). FS methods are used to do pattern classification with a set of already defined features, and they select the best set that discriminate between the patterns or

classes. FS has a lower bound because of the fixed set of basis functions or features. FS methods cannot produce features not from the original set and this usually result in sub-optimal solutions. On the other hand, FE has few constraints on the transform from input space to feature space giving it more flexibility; however, FE is a significantly more difficult optimization problem. [6]

Neural networks combine FS and FE in one model by extracting features in their hidden layers and performing feature selection in their output layer by using back-propagation algorithms. These models are complex and generally encounter the problem of different convergence rates as FS generally has a faster convergence rate than FE. In order to achieve equivalent convergence rates to FS methods, Carneiro, G. et al in [6] introduces an algorithm that performs joint FS and FE. This method uses linear search in feature space spanned by two of its initial features and chooses the one that reduces the Bayes Error the most. Our classification method also attacks the convergence problem in a similar manner by performing sequential feature selection and extraction. We use probabilistic measures for feature selectivity called softmax. This selectivity measure is propagated to the feature layer and is used to extract features. Selectivity measure used in our learning algorithm helps to overcome the convergence problem in our neural network model.

Our supervised algorithm has two steps. In the first step, FS, we calculate the selectivity of each feature for the target class. This is done using softmax probabilities.

$$P_i = \frac{e^{x_i}}{\sum_j^{|C|} e^{x_j}} \tag{1.2}$$

This function represents the selectivity of each unit towards class i and can be used to do feature selection. There are other methods of measuring selectivity such as divisive normalization [24] to achieve the same result, which shows to be biologically plausible. Once the selectivty is measured we can change the features so the overall energy that goes to the incorrect classes is minimized and the energy to the correct class is higher than the rest.

This work describes an efficient online learning concept that is used for supervised and unsupervised algorithms. Our algorithm relies on three simple steps:

calculating activations, sorting, and minimizing cross-entropy error of template and activations. We extract different types of features using different templates. Among many interesting features that we extract are edges and blobs in our unsupervised learning experiments and generic faces of individuals in the supervised learning experiments. In FE using unsupervised method, we used a single layer network using a variety of template functions for learning features. These include: linear (fig 5.4), Gaussian(fig 5.9), and exponential(fig 4.1) functions. In these experiments, we show the ability of our model to learn features similar to features found in V1. In supervised learning experiments we achieve near perfect classification, 98.0 percent accuracy on Yale dataset [7] and 97.2 percent on MNIST dataset [15].

Chapter 2

Background Material

This chapter describes the background material and related works needed to make this thesis self-contained.

2.1 Feature Extraction

Most algorithms are controlled by parameters. The learning algorithm tries to find parameters that best fit the training examples. The assumption is that if the training data is a good representation of the data then the performance of the classifier on test examples will be close to the training examples.

There are many learning algorithms and they differ in performance. Some algorithms do well in small training sets but are not practical for large training sets. For example, most Bayesian algorithms are classifiers with high accuracy [16], but are mostly avoided for large data sets due to their computational complexity. Support Vector Machines [22] and Adaboost [10] are among the most practical classification algorithms that are used today.

The performance of learning algorithms strongly depends on the way the data is represented. The most common way to represent data is to use feature values. Feature values are extracted from the input data using basis or feature functions. If the basis functions map the data so that a learning algorithm can classify them better, then these features are called discriminative features. Most algorithms extract their own features, but some use predefined features that are

known to work well on some type of data. For example, in image classification one may use Gabor features or filters that extract edges of different orientations and sizes that are known to be successful in image classification tasks. Neurons that are selective to Gabor like features have been discovered in the human visual pathway, more specifically in V1 layer. In the field of machine learning there has been intensive work on methods that can learn these types of features from natural images. In this section we introduce some of the most commonly used methods to extract features.

Let W be a transformation matrix that transforms the data vector, $\boldsymbol{x}$, into *feature space* $\mathcal{F}$ through feature function $f(\boldsymbol{x})$. The rows of matrix W are the basis vectors spanning a subset $\mathcal{X}_s \subset \mathcal{X}$. Linear feature function of $\boldsymbol{x}$, $\boldsymbol{y} = W\boldsymbol{x}$ is an example of such a function. However, in our work, we consider $f(\boldsymbol{x})$ to be a logistic function of $\boldsymbol{x}$, $\boldsymbol{y} = 1/(1 + exp(-W\boldsymbol{x})$, which is just a linear function squashed by a nonlinear sigmoidal function to range the feature value between 0 and 1.

Other commonly used methods to perform feature extraction are: Discrete cosine transform (DCT), Principal component analysis (PCA) and Independent component analysis (ICA).

DCT expresses a sequence of finitely many data points in terms of a sum of cosine functions oscillating at different frequencies. PCA transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. ICA is a computational method for separating a multivariate signal into additive subcomponents supposing the mutual statistical independence of the non-Gaussian source signals. Special form of ICA called linear ICA describes observation $\boldsymbol{x}$ as a linear combination of sources $\boldsymbol{s}$ and the mixing matrix A with additive noise ϵ:

$$\boldsymbol{x} = A\boldsymbol{s} + \epsilon \tag{2.1}$$

There are also nonlinear ICAs, where the mixing of the sources do not need to be linear and can be achieved with a nonlinear mixing function.

2.2 Feature Selection

Feature selection (FS) consists of selecting the best subset from a set of already available basis functions that is optimal for classification. Since FS cannot produce its own features, the results may be sub-optimal because there could be another set of basis functions that would be optimal for the classification task. Next, we describe some of the well-known measures for feature selection.

2.2.1 Infomax Feature Selection

Given a classification problem that maps a feature vector $\boldsymbol{x} = (x_1, \ldots, x_N)^\top \in \mathcal{X} \subset \mathbb{R}^N$ that is generated from the process $\boldsymbol{X}$ into the class label $i \in \mathcal{L}$ that is sampled from random variable Y. Let $\mathcal{S}$ be a set of feature transforms under consideration, the infomax space is

$$\mathcal{X}^* = \arg\max_{\mathcal{X} \in \mathcal{S}} I(Y; \boldsymbol{X}), \tag{2.2}$$

where

$$I(\boldsymbol{X}; Y) = \sum_i \int_{\mathcal{X}} p_{\boldsymbol{X},Y}(\boldsymbol{x}, i) \log \frac{p_{\boldsymbol{X},Y}(\boldsymbol{x}, i)}{p_{\boldsymbol{X}}(\boldsymbol{x}) p_Y(i)} \tag{2.3}$$

is the mutual information between $\boldsymbol{X}$ and Y. Feature set $\boldsymbol{X}$ usually is a selection of bandpass filters such as a Wavelet, Gabor, or windowed Fourier transform. One of the classes in infomax is the *Marginal infomax* [2]. Marginal infomax approximates the measure of discriminant information from individual features:

$$M(\boldsymbol{X}; Y) = \sum_{k=1} DI(X_k; Y). \tag{2.4}$$

Infomax is closely related to minimization of bayes classification error. [23] Another criterion used in feature selection is minimum Bayes error that we discuss next.

2.2.2 Minimum Bayes Error Features

Let us define the optimal feature space for a classification problem $\mathcal{C}_{\mathcal{X}}$ to be:

$$\mathcal{X}^* = \arg\min_{\mathcal{X} \in \mathcal{S}} J(\mathcal{C}_{\mathcal{X}}), \tag{2.5}$$

where $J(\dot{})$ is a cost, and $\mathcal{S}$ is the set of feature transforms under consideration. One measure of the goodness for a classification problem is the *Bayes error* [8] that measures the lowest possible probability of classification error:

$$L^*_{\mathcal{X}} = 1 - E_x \left[\max_i P_{Y|X}(i|X) \right], \tag{2.6}$$

This cost function is non differentiable because of the max function involved. We can use softmax to approximate the max function.

$$s\{P(h_k|c_i, X), \sigma\} = \frac{e^{\sigma P(h_k|c_i,X)}}{\sum_j^{|C|} e^{\sigma P(h_k|c_j,X)}} \tag{2.7}$$

Figure 2.1 shows both the max and softmax function for different σ values. From this figure you can see $\sigma = 10$ is a very close approximation to the max function. In our learning algorithm we use a similar measure, softmax probability, to determine selectivity towards a class and is described in detail in the Supervised Learning section. In the next section we talk about artificial neural networks that perform both FE and FS within the same model.

2.3 Artificial Neural Networks

Artificial neural networks (ANNs) are used in many real-world applications and are the subject to a wide range of research. They are computational models that try to resemble biological neural networks by simulating the known structural and functional aspects of our neural system. It is important to mention that these models are overly simplified versions of the actual interaction that happens in a real neural system. These models consist of interconnections between groups of artificial neurons using mathematical computations know as connectionist approach. Most artificial networks dynamically change their parameters based on the observed information seen by the networks and propagated through their internal connections during learning. Neural networks are used in statistical data modeling because of their ability to model complex relationships between inputs and outputs.

A neural network is a network of simple elements called nodes or neurons that perform computation on their input, connections from other nodes and compute an output value. This is inspired by the observation of neurons in the central nervous system that connect through axons, dendrites and synapses and create membrane potential that can affect other connected neurons. Neural networks can be adaptive and change the strength also known as weights of the connections in the network to produce a desired signal.

Artificial neural networks and biological neural networks both act like functions that are performed collectively and in parallel by the units. Artificial neural network models are employed in statistics, cognitive psychology, artificial intelligence, theoretical neuroscience as well as computational neuroscience. Research in artificial neural networks has been directed to more practical approaches based on statistics and signal processing, rather than biological motivation. In these approaches, neural networks are used as components in larger systems that combine both adaptive and non-adaptive elements.

Artificial neural networks models are essentially mathematical models defining a function $f : X \to Y$. Function $f(x)$ is defined as a combination of other functions $g_i(x)$, which they themselves can further be made up of other functions. This is represented as a network of functions with collective inputs to other functions. These functions can be combined in variety of ways; however, most of the time, they are combined in a nonlinear weighted sum, $f(x) = K\left(\sum_i w_i g_i(x)\right)$, where K is usually a nonlinear function also known as *activation function*. Popular choices for activation functions include logistic, hyperbolic tangent and softmax function.

The relationship between the functions within nodes are usually seen in functional or probabilistic ways. In a functional view, the inputs are looked at as changing the dimensionality of their input and are utilized when we are trying to solve an optimization problem. In a probabilistic view, $F = f(G)$ is a random variable that is dependent on its input random variable G and so on. This view is used when the network is treated as a graphical model. Probabilistic and functional views are equivalent. Networks that are only dependent on the input are called

feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called *recurrent neural networks*. In recurrent networks, not only is f dependent on its input from lower layers but it is also dependent on itself.

2.3.1 Learning in Artificial Neural Networks

Learning in neural networks consist of a task and a set of functions F, where given a set of "observations" to find $f^* \in F$ which solves the task in an optimal way. This is translated into an optimization problem where a cost function is defined $C : F \to \mathbb{R}$ such that, for the optimal solution f^*, $C(f^*) \leq C(f)\ \forall f \in F$.

The cost function C, also known as loss function is used in learning algorithms and a way to measure the progress in learning. Most learning algorithms search through the solution space to find a function that has the smallest possible cost.

The cost function of the observations can learn a model of the data. A classic example of a cost function on observation data is the mean squared error between the model output and the observation of the output, $C = E\left[(f(x) - y)^2\right]$, for data pairs (x, y) drawn from the data distribution. In practice we have finite samples from the data distribution. In this case the cost function becomes $\hat{C} = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$ where N is the number of samples. When N is too large or we only have access to samples in smaller batches or one at a time, the cost is partially minimized at each step, this is called *online learning*.

Design of a cost functions can be arbitrary; however, it ultimately depends on the learning task. Sometimes it is important to be able to find a cost function that is minimized to the global value and so the cost function is desired to be convex, or one may desire to have a cost function that its derivative is easily computed to use gradient decent algorithms to find the solution. The choice of the model can make a certain cost functions more appealing. For example, in probabilistic models its usually the case that the posterior probability of the model is used as an inverse cost. However learning task is the main driving force behind the choice of cost function.

In supervised learning task, we are given a pairs $(x, y), x \in X, y \in Y$ where

x is usually the input data like image, text or sound and y is the label associated with that input data. This type of learning aims at finding a function $f : X \rightarrow Y$ that can classify the sample correctly. Inference in this model is simply done by applying this mapping f from the input x to output y. In this group cost functions are related to the difference between the mapping by the model and the actual mapping from the data. Mean-squared error, perviously mentioned, is an example of such method that tries to minimize the average squared error between the output of our model, $f(x)$, and the target value y over all the training examples. This cost function is used in Multilayer perceptron to calculate the backtropagation error. Supervised learning tasks include pattern recognition, and regression.

In unsupervised learning, we are only given some data x with no label. The cost function in this case is a function of data x and the networks output f to be minimized. The cost function depends on what we are trying to model. For example, the task can be to output the reconstruction of data where $f(x)$ output a vector of the size x through some non-linearity. This forces the network to model x and minimize the difference between the input and the networks output creating a generative model. General unsupervised tasks include clustering, dimensionality reduction, estimation of distributions, and data compression.

Neural networks are also used as part of systems for reinforcement learning. In reinforcement learning data are usually not given, but generated by agent. Agents generate these data from their interactions with the environment. At each time step agent can perform an action and generate an observation from the environment and a cost is also generated for the that action. The goal is to discover a policy for selecting actions that minimizes some measure of a long-term cost (i.e the expected cumulative cost). Tasks that fall within the paradigm of reinforcement learning are control problems, games and other sequential decision making tasks.

2.3.2 Hopfield Network

The Hopfield network is a recurrent neural network in which all connections are symmetric. Invented by John Hopfield in 1982 [14], this network guarantees

that its dynamics will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable (or associative) memory, resistant to connection alteration.

A stochastic neural network differs from a typical neural network because it introduces random variations into the network. In a probabilistic view of neural networks, such random variations can be viewed as a form of statistical sampling, such as Monte Carlo sampling.

The Boltzmann machine can be thought of as a noisy Hopfield network. Invented by Geoff Hinton and Terry Sejnowski in 1985 [1], the Boltzmann machine is important because it is one of the first neural networks to demonstrate learning of latent variables (hidden units). Boltzmann machine learning is at first slow to simulate, but the contrastive divergence algorithm of Geoff Hinton allows models such as Boltzmann machines and products of experts to train much faster. In the next chapter we describe our model which is utilizing connections like Boltzmann machines and Hopfield networks.

2.3.3 Boltzmann Machine

The Boltzmann Machine(BM) is a probabilistic model that is designed to resemble neural representations. This model defines probability distribution over $\boldsymbol{X} \in \{0,1\}^N$ as:

$$P(\boldsymbol{X}) = \frac{e^{(\frac{1}{2}\boldsymbol{X}^\top W \boldsymbol{X} + \boldsymbol{b}^\top \boldsymbol{X})}}{Z(W, \boldsymbol{b})}, \tag{2.8}$$

where W and $\boldsymbol{b}$ are the parameters of the Boltzmann Machine. W is the weights of the undirected connections between the elements of $\boldsymbol{X}$ with no connection to itself, $W_{ii} = 0$, $W_{ij} = W_{ji}$, $\boldsymbol{b}$ is the bias vector and $Z(W, \boldsymbol{b})$ is the partition function to make $P(\boldsymbol{X})$ a probability distribution. $\boldsymbol{X}$ usually partitioned into two sets visible V and hidden H. To perform Gibbs sampling from $P(\boldsymbol{X})$ we can repeatedly set $X^{(i)}$ to 1 with probability $(1 + \exp(-(WX)^{(i)} - b^{(i)}))^{-1}$. To perform Gibbs sampling from $P(H|V)$ we fix the values of V and sample H using the same probability function. Another useful property of a Boltzmann Machine

is the simple form the derivatives of its average log likelihood:

$$\frac{\partial \left\langle \log P(V) \right\rangle_{\widetilde{P}(V)}}{\partial W_{ij}} = \left\langle X^{(i)} X^{(j)} \right\rangle_{P(H|V)\widetilde{P}(V)} - \left\langle X^{(i)} X^{(j)} \right\rangle_{P(V,H)} \tag{2.9}$$

where $\widetilde{P}(V)$ is the distribution over the training data. This learning rule has two parts, the positive component which is the correlation of data and the hidden units and a negative component which is the state of the model when it is let to run freely. This negative component is where the model likes to settle in. This gradient will be zero when the model settles in the data distribution. A simplified version of a Boltzmann machine's called a restricted Boltzmann machine (RBM) removes the lateral connections from both hidden and visible units. RBSs are the building blocks of our model and they are described in the next chapter.

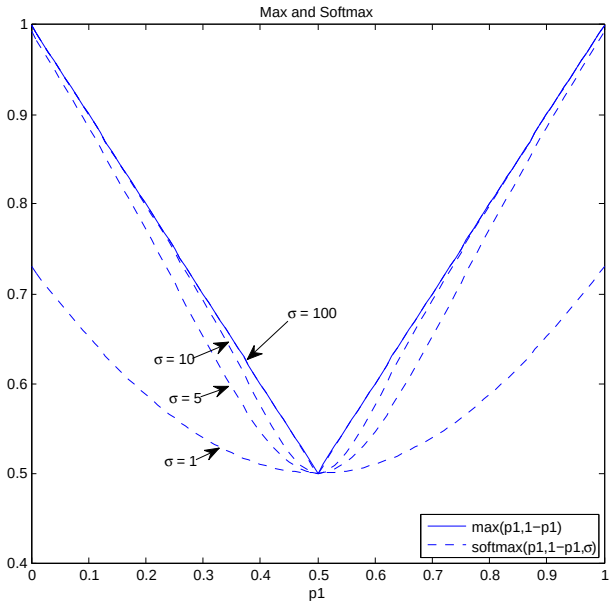

Figure 2.1: Max and Softmax Comparison

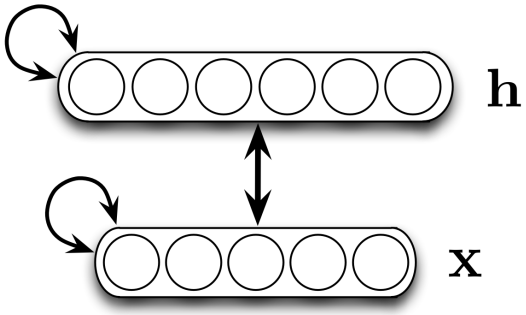

Figure 2.2: Boltzmann Machine

Chapter 3

The Model

Restricted Boltzmann Machines are the building blocks of our models that we use in our experiments.

3.1 Restricted Boltzmann Machine

Calculating the variables of a fully connected model such as Boltzmann machine is computationally expensive. However using a simplified version of this model we can easily calculate the values of hidden nodes of the model given the observations. This simplified model is called the Restricted Boltzmann Machine (RBM). In this model there are no lateral connections within each subsets V and H, i.e, $W_{ij} = 0 \quad \forall \{i, j\} \in V \; and \; \{i, j\} \in H$. With enough hidden units RBMs are universal approximators of the distribution of binary input [9]. In our model, we only perform deterministic evaluation of each node, instead of using the probabilities and stochastically sampling the nodes. We use probability measures in RBMS as feature values, or strength in belief that a certain feature is stimulated. Our version of RBM still uses the undirected connections and can be considered a generative model.

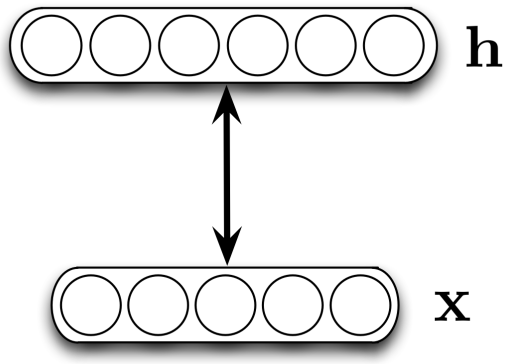

Figure 3.1: Restricted Boltzmann Machine

3.2 Stacked RBMs

The representational power of the hidden variables of an RBM is limited because they are simply linear functions that are squashed in a nonlinear sigmoid function. A more powerful representation is the stacked RBM (SRBM) where layers of RBMs are stacked on top of each other, and the hidden variables of one layer serves as visible units of the next. This setup creates an SRBM model which is a type of deep belief network (DBN). In this section we describe SRBMs, the model that is used in this thesis.

Given the data distribution $\tilde{P}(V)$ the RBM defines the joint distribution of visible V and hidden H^1 variables, $P(V, H^1)$ of the first layer. The hidden variables in the last RBM become the visible variables in the next RBM. This RBM define the joint distribution, $Q(H^1, H^2)$. $\tilde{Q}(H^1)$ is the posterior distribution of the first RBM:

$$\tilde{Q}(H^1) = \sum_V P(H^1|V)\tilde{P}(V). \tag{3.1}$$

Then our model becomes

$$Model(V, H^1, H^2) = Q(H^1, H^2)P(V|H^1) \tag{3.2}$$

given that the $Q(H^1)$ models the posterior distribution better than $P(H^1)$. This layering can continued indefinitely.

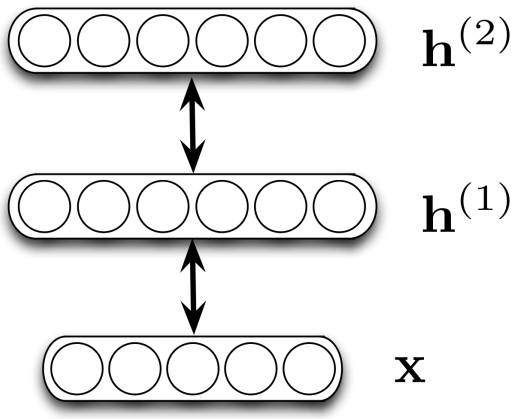

Figure 3.2: Stacked Restricted Boltzmann Machines

To do classification using stacked RBMs one can use the activation of top hidden layer to separate each class. The model used in the supervised classification experiments is shown in figure 3.3. Labels are represented in the output layer where each node belongs to a class. Inference in the network is done using a deterministic up-pass by using the probabilities of activations from the input layer to the output layer. The output unit with the highest total input is the winner.

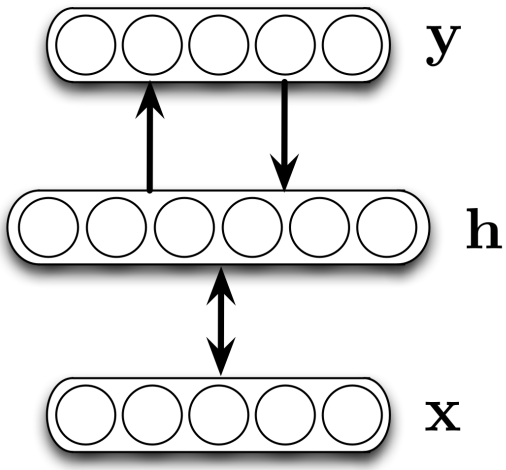

Figure 3.3: Stacked Restricted Boltzmann Machines for Classification

Chapter 4

Learning

In this chapter we describe our learning algorithm and the biological and computational motivations behind it. We first describe our biological motivation behind our learning algorithm and show how we approximate and simplify the complex interactions in the neural circuits to make our algorithm run efficiently in real time.

4.1 Motivation

In this section we look at the studies done in human visual pathway and the way the information is encoded through out our visual system, also known as *Population Coding* [11]. If each neuron had to represent a unique configuration of its *receptive field* (RF), then representing all stimulus features of almost any environment in real life such as natural scenes requires an unimaginable number of neurons. This number grows exponentially as the number of input dimensions increases. This is also called the "curse of dimensionality" [4]. From studies of single neurons representing multiple stimulus features in V1 by Grunewald, A. and Skoumbourdis, E.K. [13] it is hypothesized that the lower layers in visual pathway use distributed coding to represent the visual input. They show that lower visual layers such as V1 use combinations of neurons of only a few direction to represent all the possible directions.

In distributed coding most neurons are active at different levels. It is the-

orized that distributed coding has the advantage of resisting the curse of dimensionality under certain assumption [19]. The disadvantage of a distributed code is its high energy consumption.

It is known that in higher cortical areas the representation is sparse [26]. This sparsity in neural activities also helps to reduce the over all energy consumption of the brain as well as improving classification and decision making task. The transition from distributed code in the lower layers to sparse code in higher layers is still under studies. We propose a method that results in distributed coding in lower layers and results in sparse coding in higher layers. Looking into V1 complex cells we can see a topological arrangement of neurons that are excited towards edges with similar angles. We hypothesize that this is due to the lateral connections in each layer. We hypothesize that these lateral connections control the amount of the distribution of activities in each layer. For example, neurons can propagate the signal laterally and spread the activations to more neurons, increasing the number of active neurons therefore a more distributed code can be formed. This behavior could explain why similar features are spatially close to each other. Neurons also could inhibit the other neurons increasing the sparsity of the layers and also increase their selectivity towards a specific stimuli, this is also known as inhibitory competition in neural circuits.

However, using lateral connection in computational models specially in the hidden layers introduces a large amount of computational complexity. We propose an approximation to what lateral connection achieve in determining the coding scheme by assuming a profile of activations for each layer imposed by lateral connections and try and learn the profiles for a given training set. This learning procedure does not simulate any of the lateral interactions between neurons. In real neural networks the lateral connections are known to excite or suppress the neighboring neurons, changing the dependencies of each unit. The spread of activations can be approximated by matching activation profile templates. For example, a decaying exponential function.

$$f(x) = e^{-\lambda x} \tag{4.1}$$

where $\lambda \in (0, \infty)$ controls the rate of decay. To achieve higher distribution we can use a small value close to 0 and for sparse distribution we can use larger values. Figure 4.1 shows plots of this function for different values of λ. From template

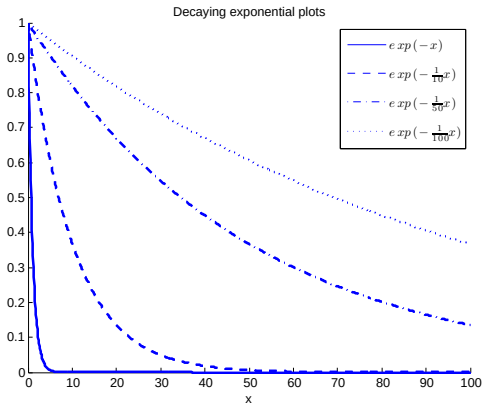

Figure 4.1: Decaying exponential function used to adjust the level of sparsity in the hidden layers

functions, we create template sets. We also refer to these sets as templates. These templates are sets of values evaluated from the template functions for each index. For our algorithm, they have the property of being in decreasing order and have values between 0 and 1.

In the following sections we describe the learning algorithm first used to

extract features in an unsupervised setting using a distributed coding activation profile template for the first layer and sparse coding for the second layer. We use our algorithm to extract features using different types of template with different shapes and levels of sparsity.

4.2 Learning Algorithm

The learning algorithm can be divided into 3 distinct steps:

1. Calculating activation of hidden nodes for a given the input and reconstruction of input from those activations (for unsupervised version)

2. Sorting of templates based on activations(unsupervised) or selectivity measures(supervised)

3. Modify the model parameters so that Cross-Entropy error between sorted activations and template and between data input and its reconstruction are minimized

Each step is explained in the following sections.

4.2.1 Feature Function

Feature values are calculated via a sigmoidal feature function:

$$f(\boldsymbol{x}) = \frac{1}{1 + e^{\boldsymbol{w}^\top \boldsymbol{x}}} \tag{4.2}$$

where $\boldsymbol{x} \in [0, 1]^d$ is the input vector and $\boldsymbol{w}$ are the parameters of the model. $\boldsymbol{w}$ is the weight vector connecting the input vector to the hidden feature node. A sigmoidal or logistic function is just a linear function that squashes the feature value so the output ranges between 0 and 1.

To calculate the reconstructions from activation we use the same sigmoidal activation function only this time we use the hidden activations and the same set of connections in reverse:

$$\hat{\boldsymbol{x}} = \frac{1}{1 + e^{\boldsymbol{w} f(\boldsymbol{x})}} \tag{4.3}$$

where $\hat{\boldsymbol{x}}$ is the reconstruction vector.

Once the activations and reconstructions are calculated, we need to sort the template's values based on the hidden activations ordering for unsupervised learning and softmax probabilities for supervised learning.

4.2.2 Sorting

Sorting is done in order to match the template descending nature to the activations. Activations are sorted in descending order so the highly active features are corresponded to the high values of the template and features with small values are corresponding to the low template values. The ultimate goal of this learning algorithm is to create activation profiles that match the ones in the template and ultimately provide similar distribution as the template. To learn these activation profiles we reduce the cross entropy of the sorted activations and the desired template. Next we explain minimization step using cross entropy loss function and its gradient decent learning rule.

4.2.3 Cross-entropy minimization

Cross entropy between two probability distributions measures the average number of bits of information, required to identify an event from a set of possibilities, given probability distribution q, rather than the "true" distribution p.

The cross entropy for two distributions p and q over the same probability space is thus defined as follows:

$$\mathrm{H}(p, q) = \mathrm{E}_p[-\log q] = \mathrm{H}(p) + D_{\mathrm{KL}}(p\|q) \tag{4.4}$$

, where $H(p)$ is the information entropy of p, and $D_{\mathrm{KL}}(p||q)$ is the relative entropy or Kullback-Leibler divergence of q from p. [18]

In the case of discrete random variable:

$$\mathrm{H}(p, q) = -\sum_{x} p(x)\ \log q(x) \tag{4.5}$$

and for continuous random variable distributions is:

$$-\int_{X} p(x)\ \log q(x)\, dx. \tag{4.6}$$

Cross-entropy error measure has been used instead of mean squared error in many optimization problems. It is usually the cases when probabilities are involved. Cross-entropy can be used as a cost function when the outputs of a network represent independent hypotheses, for example each node represents a concept that is independent of others and the activation values can be thought of as probabilities or confidence of the network in that concept is true. The cross-entropy error then indicates the difference in what the network believes and what the teacher say it should believe in. We use this error measure to learn the activation templates by minimizing the cross entropy error using gradient decent and the derivative or this error function:

$$\frac{\partial C}{\partial w_{jk}} = \eta \cdot (h_k - t_k) \cdot x_j \tag{4.7}$$

where w_{jk} is the weight that connects input node x_j to output node h_k and teacher signal t_k and η is the learning rate. Computing this derivative is relatively easy and is desirable in online learning methods. In the next section, we show how cross entropy is used to extract features from data in unsupervised setting.

4.3 Unsupervised Learning Algorithm

In unsupervised setting, we only have access to the data and not the labels. Consider a set of training data $\{\boldsymbol{x}_l\}_{l=1}^{N}$ drawn from a continuous-valued random variable X such that $\boldsymbol{x}_l \in [0,1]^{n\times 1}$. In this setting, we are interested in finding a feature transformation $f : \mathscr{X} \subset \mathbb{R}^{n\times 1} \rightarrow \mathscr{Y} \subset \mathbb{R}^{m\times 1}$. Here we consider $f(\boldsymbol{x})$ as a logistic function:

$$f(\boldsymbol{x}_l) = \frac{1}{1 + e^{-(W\boldsymbol{x}_l)}} \tag{4.8}$$

where $W \in \mathbb{R}^{n\times m}$ is the weight matrix, connecting the data layer to the feature layer and its m rows are made up of *feature vectors.*We now explain the learning algorithm in which one can learn these encodings.

Most unsupervised learning method try to learn features that can reduce reconstruction error or reduce the dimensionality of data while preserving certain structures of the data. In our algorithm, we minimize a combination of two cost functions:

$$C = C_{rec} + C_{temp} \tag{4.9}$$

Where C_{rec} is the cross-entropy error between the observed data and its reconstruction using the bidirectional connections.

$$C_{rec} = -\sum_i x_i \log \hat{x}_i \tag{4.10}$$

The second part of cost function acts as a regularization factor, to force the activation to follow the template. C_{temp} is the cross-entropy error between the activation and the template:

$$C_{temp} = -\sum_i h_i \log t_i. \tag{4.11}$$

where h_i is the activation value of feature i, calculated using the feature function given the input and t_i is the template value assigned to feature i. Using gradient-decent algorithm and the derivative of these cost functions we can update the model parmaters. Weight update rule is just the combination of the gradient of each cost function:

$$\frac{\partial C}{\partial w_{jk}} = \frac{\partial C_{rec}}{\partial w_{jk}} + \frac{\partial C_{temp}}{\partial w_{jk}} \tag{4.12}$$

where

$$\frac{\partial C_{rec}}{\partial w_{jk}} = (x_j - \hat{x}_j) \cdot h_k \tag{4.13}$$

and

$$\frac{\partial C_{temp}}{\partial w_{jk}} = (h_k - t_k) \cdot x_j \tag{4.14}$$

where x_j is j^{th} input node, h_k is k^{th} hidden node, t_k is template value paired in sorting step for k^{th} hidden node, $\hat{x}_j$ is the reconstruction of node x_j and w_jk is bidirectional weight connecting x_j to h_k. This results in the total update gradient for weights:

$$\frac{\partial C}{\partial w_{jk}} = \sigma \cdot ((x_j - \hat{x}_j) \cdot h_k + (h_k - t_k) \cdot x_j) \tag{4.15}$$

where σ is the learning rate. The update rule forces the activations to follow the template while retaining the ability of the network of encoding and reconstructing

the input data. The following algorithm is a special case where each layer uses same type of template with different value for the parameter that changes the sparsity of each template. Here we would like to learn these feature vectors that result in distributed code in the lower layers and sparse code in higher layers.

foreach *training data,* x^n **do**
Let template be a monotonically decreasing function,
$template(i) \in [0\ 1] \quad \forall i = \{0, 1, ...N-1\}$ where N is the number of hidden units;
Let hidden activations be $h(x^n) = (1 + \exp(-W.x^n)^{-1}$;
Rearrange template so index i of $template(i)$ be the index of sort feature values: based on $h(x^n)$ where $h_1 \geq h_2 \geq \cdots \geq h_N$;
Let $\hat{x} = logit(h \cdot W^\top)$ be the reconstruction of x;
Update the weights with gradient decent:;
$\Delta w_{jk} = \sigma \cdot ((x_j - \hat{x}_j) \cdot h_k + (h_k - template(k)) \cdot x_j)$ where σ is the learning rate;
end

This is an online learning algorithm that learns activation profiles in an unsupervised setting by minimizing the cross-entropy between the activations and template as well as minimizing the reconstructions error. The templates used here are monotonically decreasing functions. An example of such function is a decaying exponential. To demonstrate a biologically motivated visual system we use decaying exponential functions with decaying rate proportional to the number of hidden units and the layer, $\lambda = 2\frac{l}{N}$, where l is the layer in which the template is used and N is the number of hidden units in that layer.

Our algorithm is not restricted to any particular template function, and depending on the template used the features and the activations can change. In figure 5.5 to 5.17 we show features extracted from natural image patches using different templates with variety of shapes and different amount of sparsity. In figure 5.18 to 5.21 we show reconstructed image patches and the top 16 features associated with the reconstruction. Extracted features, templates and reconstruction results

are discussed in the *Experiments* chapter.

In the next section we show how this unsupervised algorithm can be modified to work in a supervised setting for classification tasks using feed back signal from class layer and the same update rule to improve the classification results while keep the same activation profile.

4.4 Supervised Learning Algorithm

Our supervised algorithm has two main steps: (1) feature selection and (2) feature extraction. Feature selection is the step where we compute the selectivity of each hidden units using a top-down pass from the output layer and calculating softmax probabilities, where the softmax for class c_i and hidden unit h_k is:

$$s\{P(h_k|c_i)\} = \frac{e^{E(h_k|c_i)}}{\sum_j^{|C|} e^{E(h_k|c_j)}} \tag{4.16}$$

where $E(h_k|c_i) = c_i \cdot W_{ik}$ is total energy to the hidden node k given the class c_i and W_{ik} is the set of weights connecting the output unit i to node k. Softmax probability is calculated by dividing the exponential of the energy by the sum of the exponential of energies from all classes. This energy is calculated using the bidirectional connections from output layer to the hidden layer. This feedback measures the selectivity of each hidden units towards the target class c_i.

The second step is to do a bottom-up pass from the input layer to calculate the probabilities of each features and change the weights so that the feature values match the selectivity ordering calculated in the first step and the activation profile of that layer. This is done through sorting the activation profile template according to the selectivity measure calculated in step one and using the sorted template as the target distribution for cross entropy minimization.

The next step in our algorithm is to learn the weights for the selection or output layer. This is done by doing another bottom-up pass and adjust the weights so that the softmax probabilities move toward the new activations by increasing the weights between the class units and the hidden units according to the feature value of each hidden units. This is a similar update rule as feature layer except

that the activation profile for the output layer is the same as the output label. In our implementation we used a sequential feature selection and extraction in an online setting which lead to the following algorithm.

foreach *training data,* x^n **do**

foreach *layer* l **do**

Generate desired template value given the number of available features, N: Let template be a monotonically decreasing function, $template(i, l) \in [0\ 1] \quad \forall i = \{0, 1, ...N-1\}$ where N is the number of hidden units;

Let hidden activations be $h(x^n) = logit(W.x^n)$ **if** $l + 1$ *is classification layer* **then**

Let $s_k(c^n) = \frac{e^{energy(h_k|c_i)}}{\sum_j^{|C|} e^{energy(h_k|c_j)}}$ be the softmax selectivity measure for feature k;

Rearrange template so index i of $template(i)$ be the index of sort feature values: based on $s(c^n)$ where $s_1 \geq s_2 \geq \cdots \geq s_N$;

$\Delta w_{jk} = \sigma \cdot ((h_k - template(k)) \cdot x_j)$ where σ is the learning rate;

end

else

if l *is classification layer* **then**

Let *template* be the output label c^n

end

else

Rearrange template so index i of $template(i)$ be the index of sort feature values: based on $h(x^n)$ where $h_1 \geq h_2 \geq \cdots \geq h_N$;

end

Let $\hat{x} = logit(h \cdot W^\top)$ be the reconstruction of x;

Update the weights with gradient decent:;

$\Delta w_{jk} = \sigma \cdot ((x_j - \hat{x}_j) \cdot h_k + (h_k - template(k)) \cdot x_j)$ where σ is the learning rate;

Let $x^n = logit(W.x^n)$;

end

end

end

Chapter 5

Experiments

In this chapter, we show the result of applying our algorithm on a few popular datasets and compare our performance with other learning algorithms.

5.1 Datasets

The MNIST [15] database contains 28×28 pixel grayscale images of handwritten digits. There are 60,000 training images and 10,000 test images. For our experiments the label of each digit is converted into binary representation of one-hot code where a single unit is set to 1 and all the others are set to 0. The task here is to identify the digits associated with the images.

The Yale face set [7] consist of images of 15 individuals, with 11 grayscale images 320×243 with different lighting (e.g. center, right or left) and artifacts such as glasses. We have resized these images to 60×80 to reduce the computation time for our experiments. The task here is to identify the individual in each image.

5.2 Learning Details

First set of experiments is done in unsupervised setting using natural image patches to measure the ability of the algorithm to learn the activation templates at each level. To demonstrate this we use plots of average activations in a sorted order plotted over the template used in learning that layer. The network has two layer that we refer to as V1 and V2 for first layer and second layer respectively. Figure 5.1 shows average sorted activations of each layer of a network before training. Dots in this figure and subsequent figures show the average sorted activations and line shows the templates used in learning that layer. After 100 epochs of training the networks starts to learn the templates, this is shown in figure 5.2, and after 500 epochs the network's activations are very close to the desired templates shown in figure 5.3.

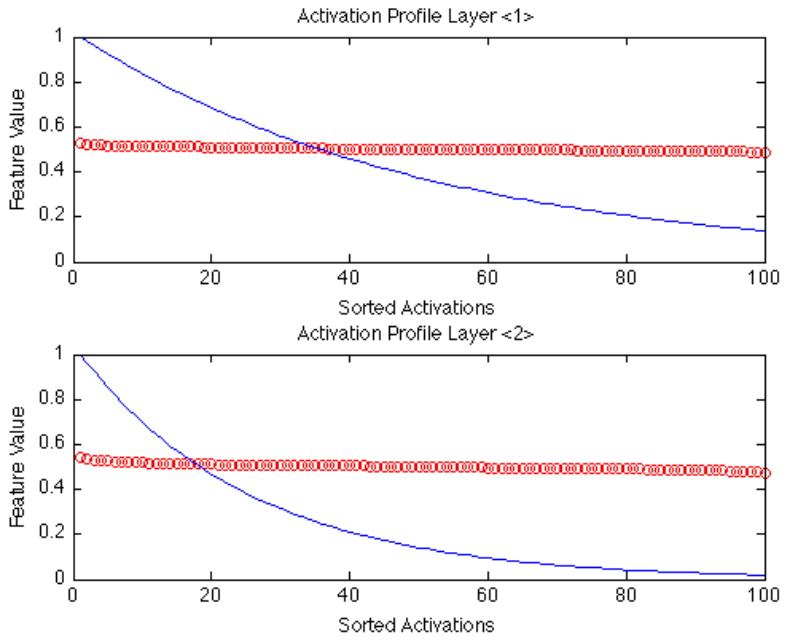

Figure 5.1: Activation profile for V1 and V2 layers before training

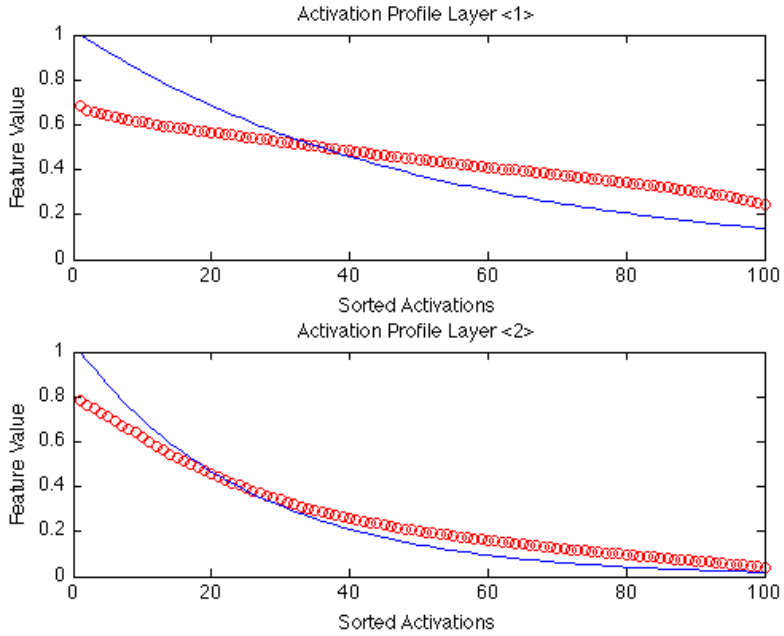

Figure 5.2: Activation profile for V1 and V2 layers at epoch 100

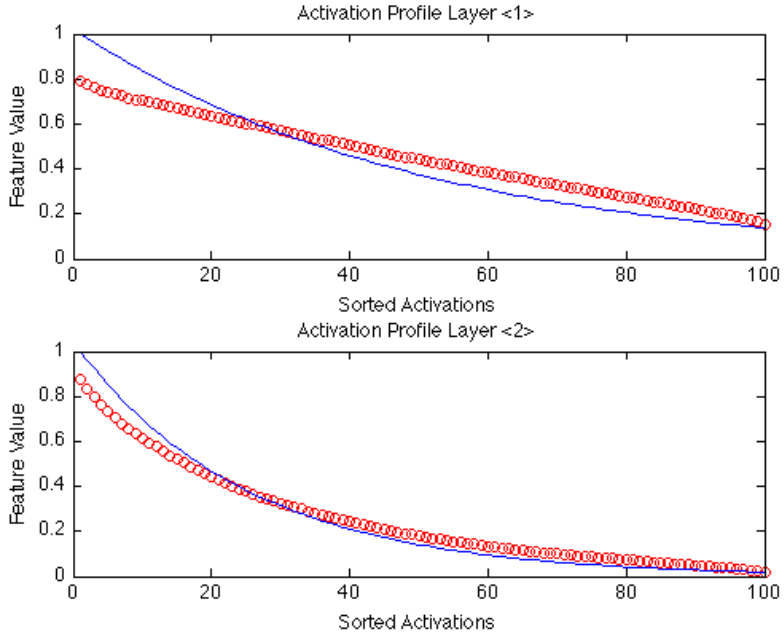

Figure 5.3: Activation profile for V1 and V2 layers at epoch 500

Second set of experiments done using unsupervised learning algorithm is to learn different templates using different functions as our template function. We use Gaussian, exponential, and linear templates, with a tuning parameter that changed the "sparsity" of the templates. We experimented with 4 different levels of sparsity. Both the template and the extracted features are shown in the next section.

First template function we experiment with is a linear function:

$$T_{linear}(x, \lambda) = \begin{cases} 1 - \lambda x & T(x, \lambda) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

where $\lambda > 0$ is the rate of decay and x is the index of the hidden node starting from 0. Higher λ value create sparser templates while lower values create wider and more distributed templates. Plots of features extracted from natural image

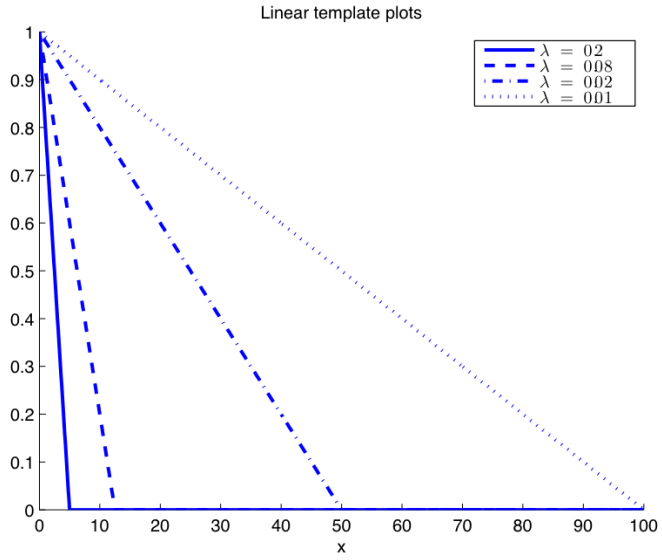

Figure 5.4: Linear template function plots for different λ values

patches using linear template function with different λ values are shown in figures 5.5 to 5.8.

The second function used in our experiments is the Gaussian distribution

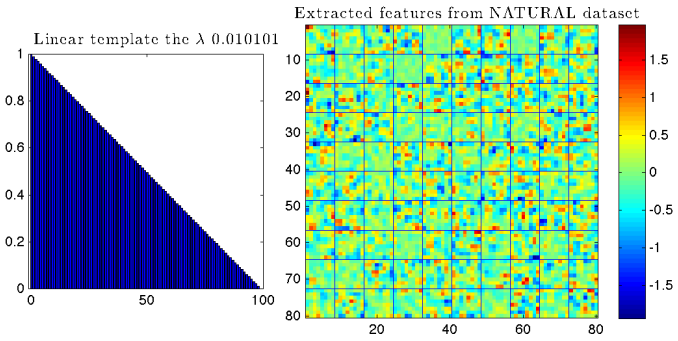

Figure 5.5: Features extracted from natural image patches using linear template with $\lambda \approx 0.01$.

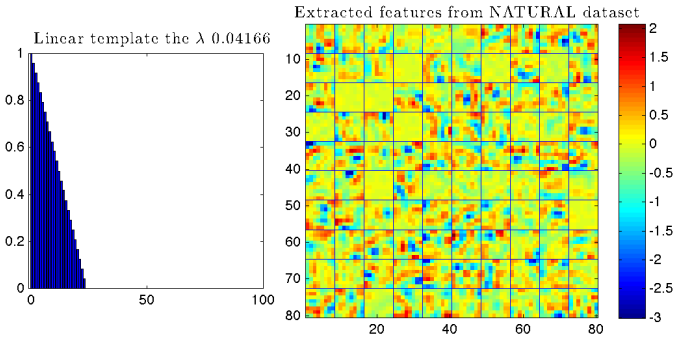

Figure 5.6: Features extracted from natural image patches using linear template with $\lambda \approx 0.04$.

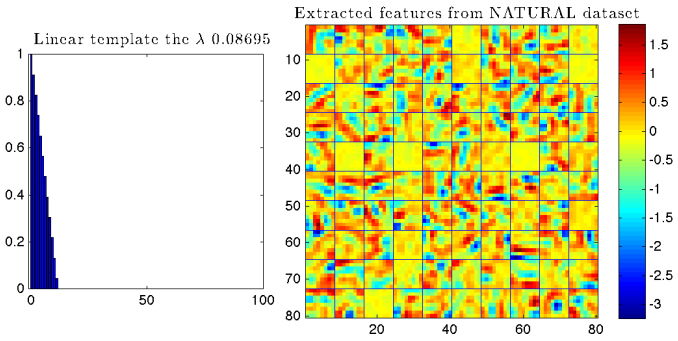

Figure 5.7: Features extracted from natural image patches using linear template with $\lambda \approx 0.087$.

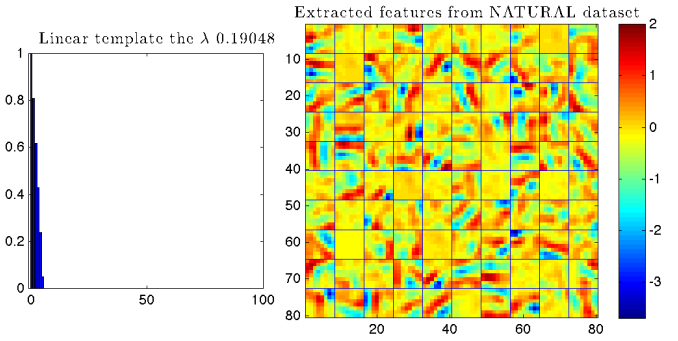

Figure 5.8: Features extracted from natural image patches using linear template with $\lambda \approx 0.19$.

function without the normalization factor and zero mean:

$$T_{gauss}(x, \lambda) = e^{-x^2/(2\lambda^2)} \tag{5.2}$$

where $\lambda > 0$ is the variance of the gaussian distribution and controls the sparsity of this function. Plots of features extracted from natural image patches using

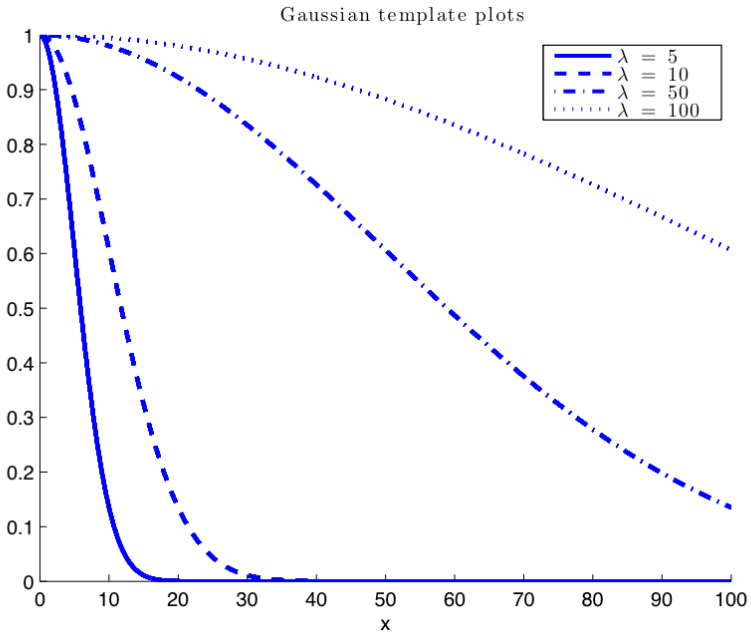

Figure 5.9: Gaussian template function plots for different λ values

gaussian template function with different λ values are shown in figures 5.10 to 5.13.

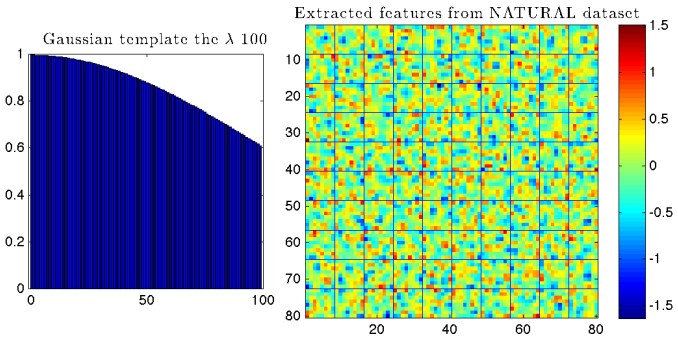

Figure 5.10: Features extracted from natural image patches using Gaussian template with $\lambda = 100$.

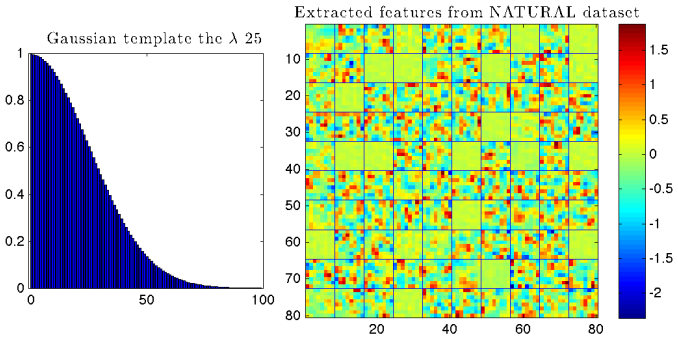

Figure 5.11: Features extracted from natural image patches using Gaussian template with $\lambda = 25$.

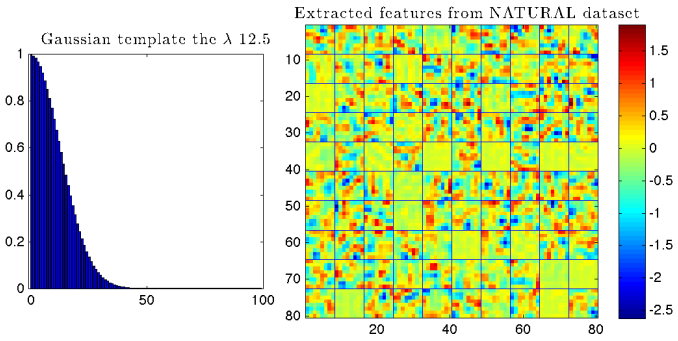

Figure 5.12: Features extracted from natural image patches using Gaussian template with $\lambda = 12.5$.

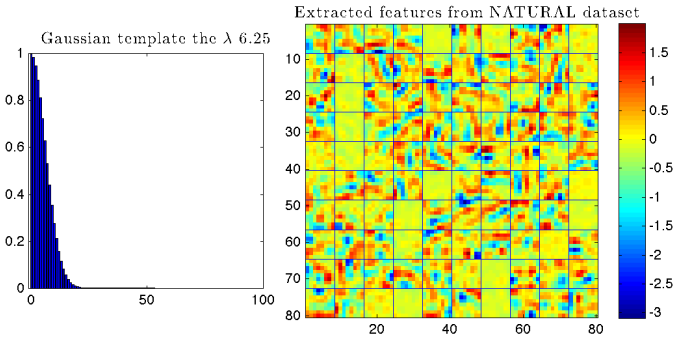

Figure 5.13: Features extracted from natural image patches using Gaussian template with $\lambda 6.25$.

The third class of template functions we experiment with is the decaying exponential also used as an example in the prior chapters:

$$T_{exp}(x, \lambda) = e^{-\lambda x} \tag{5.3}$$

where $\lambda > 0$ is the rate of decay of exponential function. Plots of features extracted from natural image patches using decaying exponential template function with different λ values are shown in figures 5.14 to 5.17.

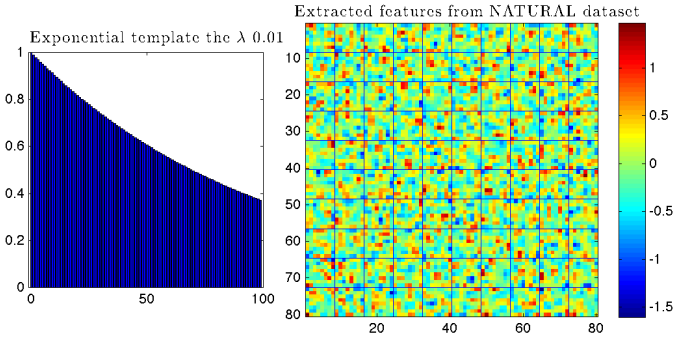

Figure 5.14: Features extracted from natural image patches using decaying exponential template with $\lambda = 0.01$.

In figure 5.5 to 5.8 we see the ability of our model to extract sparse or distributed features using linear templates. This is also shown in other types of templates, Gaussian 5.10 to 5.13, and decaying exponential 5.14 to 5.17. Because of the sparse nature of decaying exponential function most of the features extracted using these templates are sparse. There are subtle differences even among the sparse features extracted using different types of template functions. For example, some templates extract edge features while others extract blobs and curve.

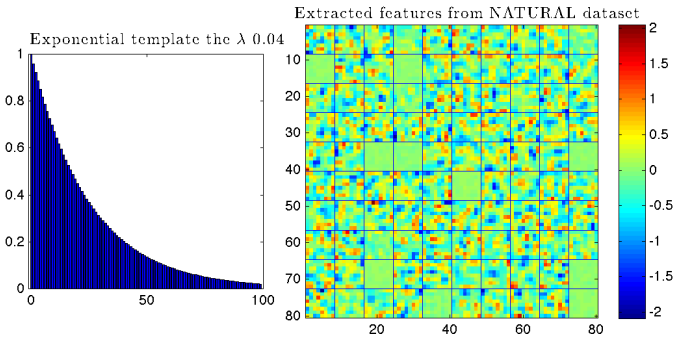

Figure 5.15: Features extracted from natural image patches using decaying exponential template with $\lambda = 0.04$.

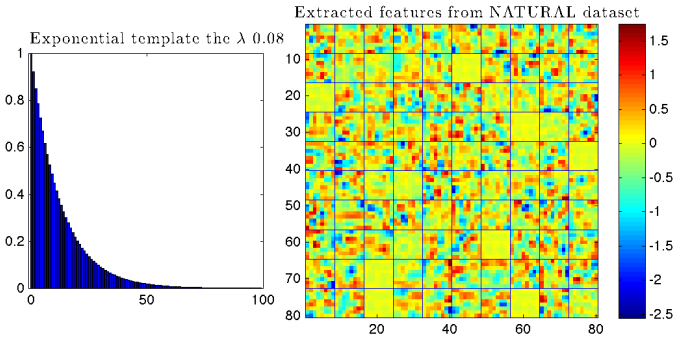

Figure 5.16: Features extracted from natural image patches using decaying exponential template with $\lambda = 0.08$.

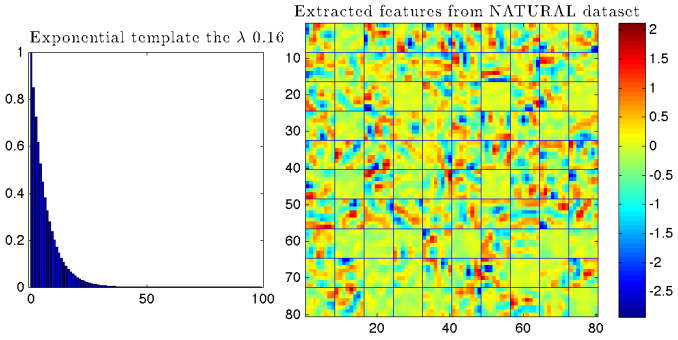

Figure 5.17: Features extracted from natural image patches using decaying exponential template with $\lambda \approx 0.16$.

Using the trained networks shown in 5.5 to 5.17, we show their ability to do reconstruction of image patches. A random image patch is extracted from an image that has not been seen by the network. To show how each network perform we show the image patch, reconstruction, activations cause by that image patch and the top 16 features. Figures 5.18 to 5.21 are for networks trained by linear templates with different values of λ. As the templates become sparser you can see the effect on the activation subplots.

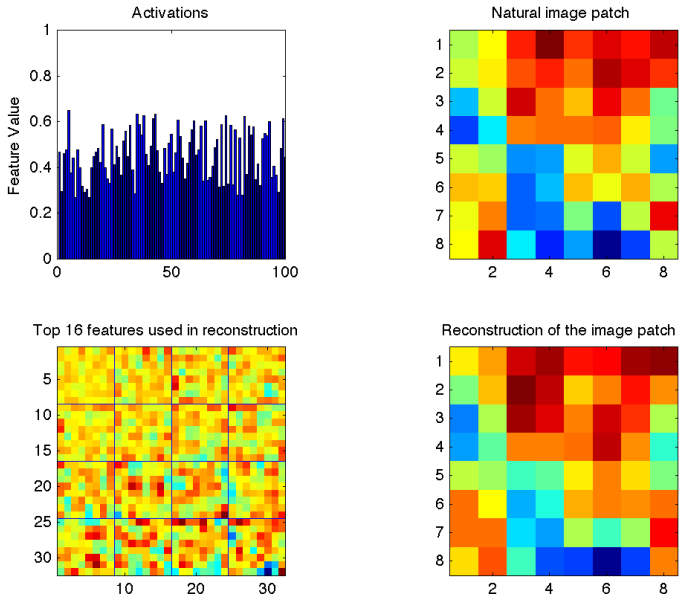

Figure 5.18: Reconstruction in network trained by linear template (figure 5.5)

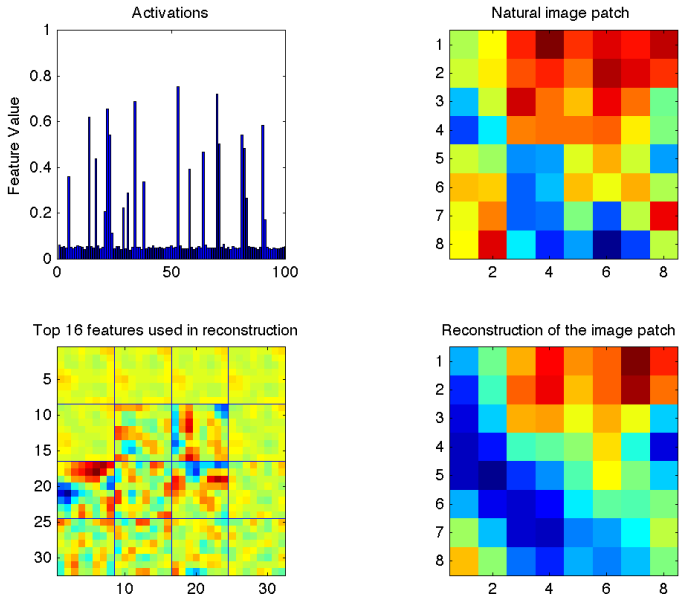

Figure 5.19: Reconstruction in network trained by linear template (figure 5.6)

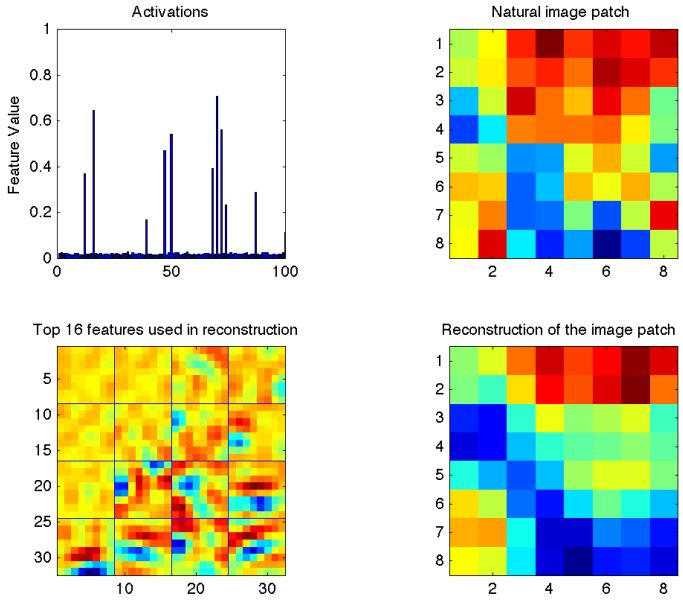

Figure 5.20: Reconstruction in network trained by linear template (figure 5.7)

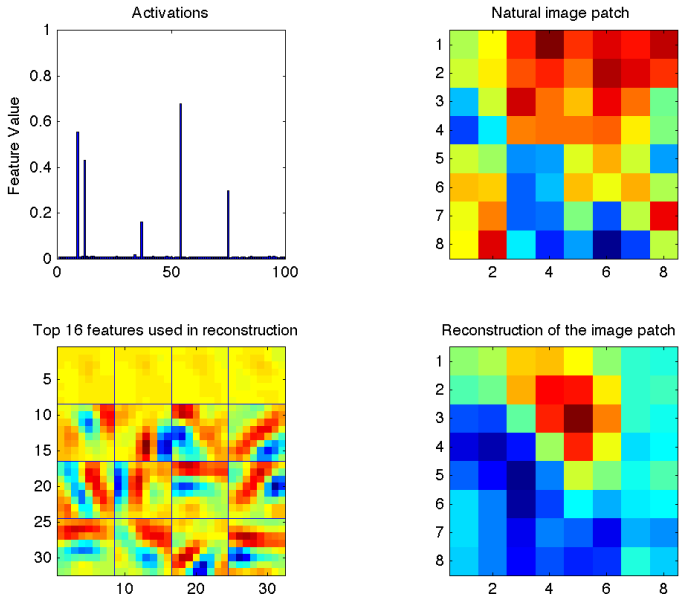

Figure 5.21: Reconstruction in network trained by linear template (figure 5.8)

Figures 5.22 to 5.25 show the reconstruction for networks trained by Gaussian templates with different λ values. Figures 5.26 to 5.29 show the recon-

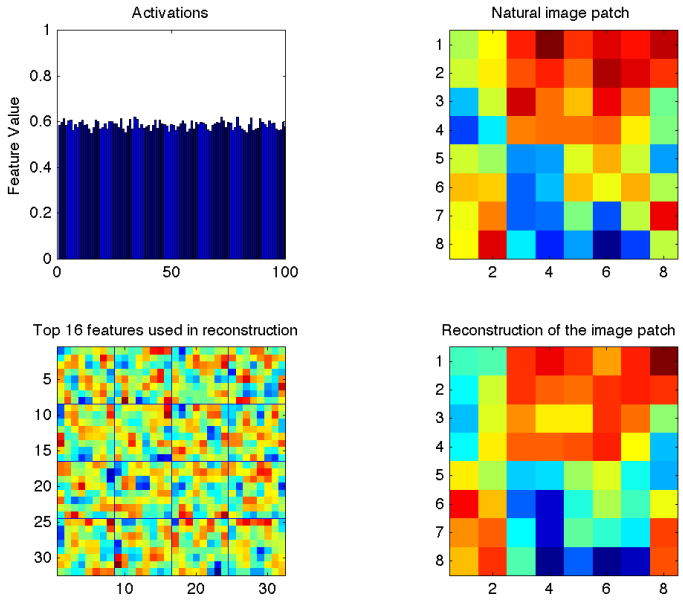

Figure 5.22: Reconstruction in network trained by Gaussian template (figure 5.10)

struction for networks trained by decaying exponential templates with different λ values. In figure 5.18 to 5.21 we show how each trained model is able to reconstruct the original test data very well. Some templates are better than others in reconstruction. We have performed reconstruction on 10k randomly chosen image patches and have measured mean-squared error (MSE) per image patch. Table 5.1 shows the numbers for each of the models trained using different templates. Lowest reconstruction errors are bold-faced for each template type. We trained PCA features over 5,000 image patches. Using the top 100 PCA features the MSE reconstruction error was 0.00018. This is one order of magnitude better than our results; however, our algorithm focuses the form of features and their semantics. One of the first observations is that the sparser templates have better reconstruction error even though in all the reconstruction plots, 5.18 to 5.29, reconstruction of more distributed networks looks more similar to the original image patch.

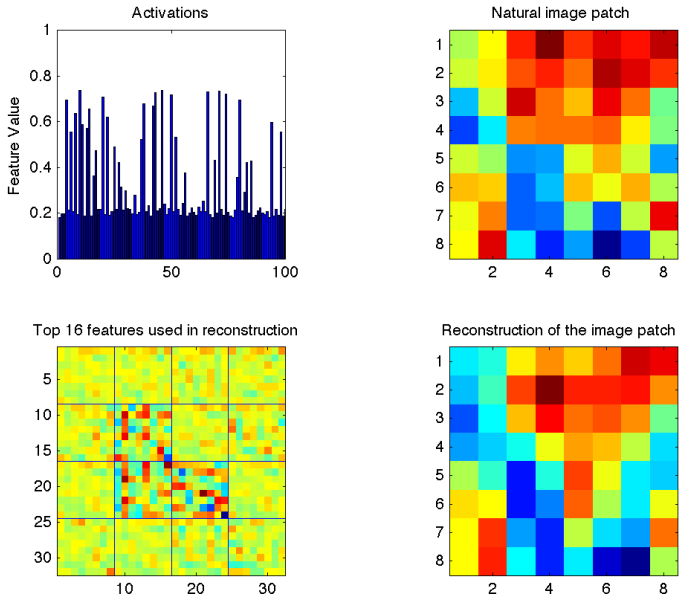

Figure 5.23: Reconstruction in network trained by Gaussian template (figure 5.11)

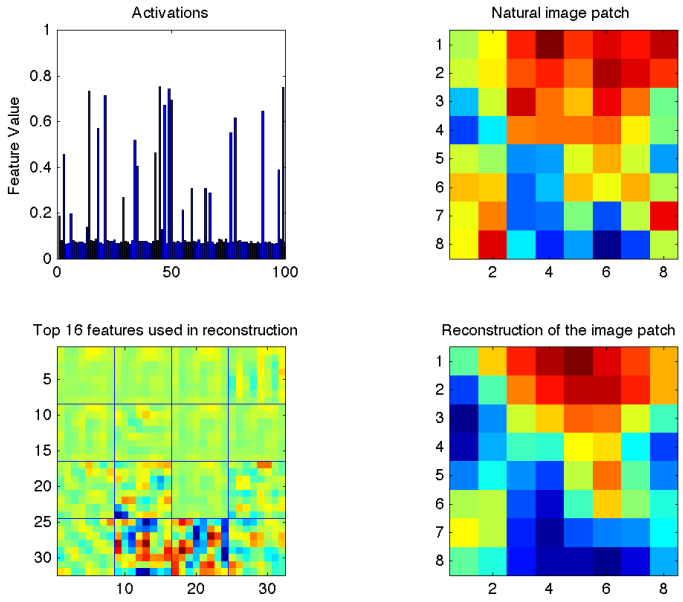

Figure 5.24: Reconstruction in network trained by Gaussian template (figure 5.12)

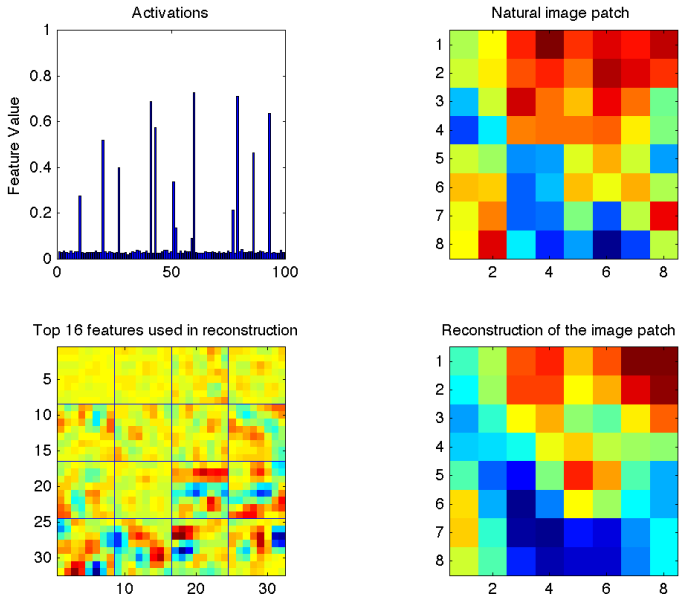

Figure 5.25: Reconstruction in network trained by Gaussian template (figure 5.13)

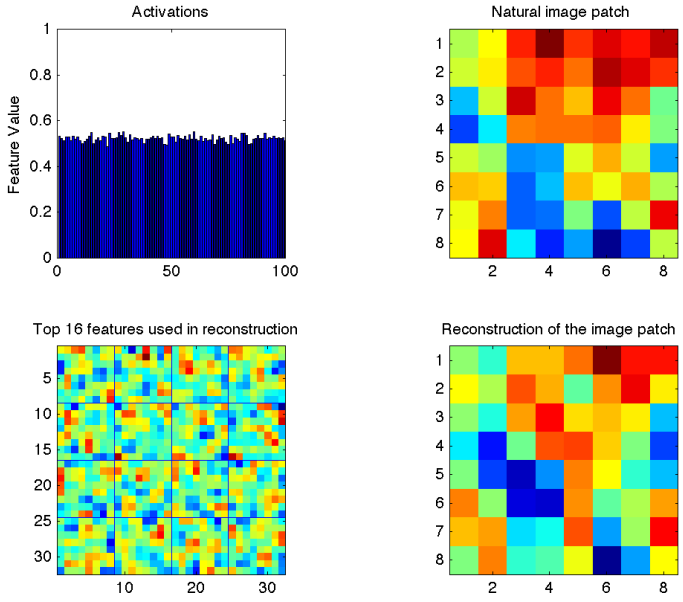

Figure 5.26: Reconstruction in network trained by exponential template (figure 5.14)

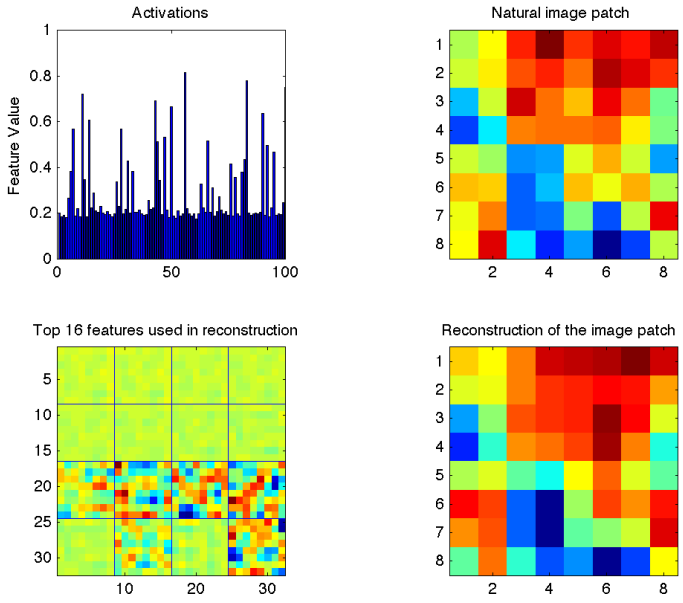

Figure 5.27: Reconstruction in network trained by exponential template (figure 5.15)

Table 5.1: Pixel level MSE for natural image patches reconstruction over 10k image patches, ordered by template sparseness, from least sparsity (1) to most sparsity (4)

	Linear	**Gaussian**	**Exponential**
1	0.008912	0.063471	0.019060
2	0.003454	0.006640	0.007156
3	**0.002798**	0.004819	0.005766
4	0.003503	**0.002455**	**0.002897**

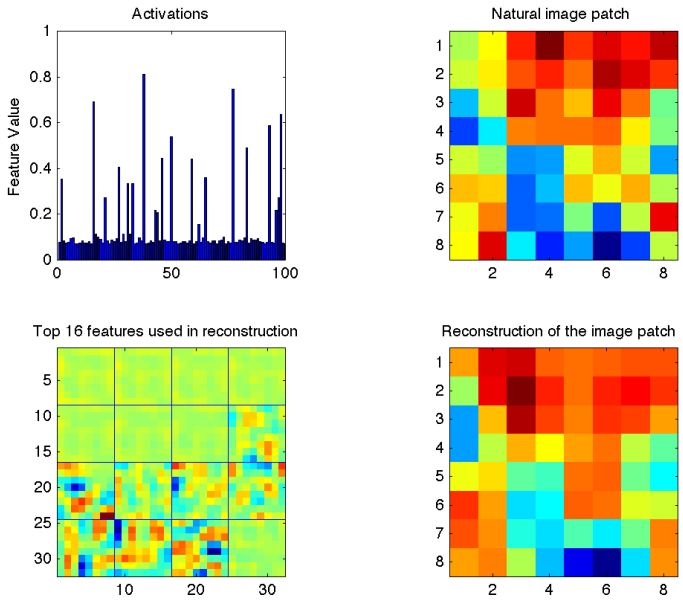

Figure 5.28: Reconstruction in network trained by exponential template (figure 5.16)

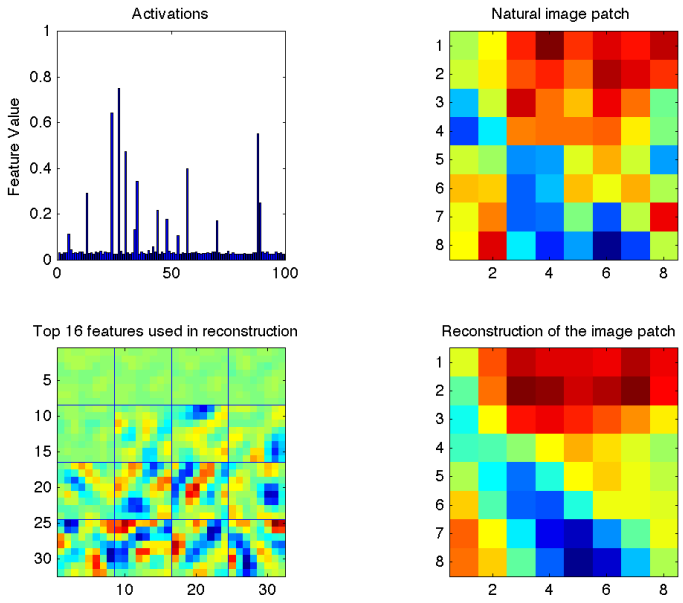

Figure 5.29: Reconstruction in network trained by exponential template (figure 5.17)

We now go over the learning details of our experiments using our supervised version of the algorithm. All classification models use a two-layer network with input, output and a hidden layer. The dimensionality of the input is the actual pixels normalized to range between 0 and 1, and the dimensionality of the output layer is the number of classes in the problem. Each class is represented by a node in output layer. In vector form, each class is represented by setting one bit to 1 and the rest to 0.

We then select the number of hidden units, learning rate, and epochs using 5-fold cross-validation. Cross-validation is a technique for measuring how the results of a statistical analysis will generalize to an independent data set. It is commonly used in finding parameters of predictive model and can measure how it will perform in practice. One round of cross-validation involves partitioning the data set into two subsets, training set and validation set. The model is trained on training subset and tested on the validation set. To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds. K-fold cross-validation is commonly use in evaluating the performance of predictive models.

In K-fold cross-validation, data set is partitioned into K equal size subsets. At each round of cross-validation a different partition is set aside for validation of the model that is trained on the rest of the subsets. This is repeated K times (K "folds"), for a different partition as the validation set. All the results are averages and used to evaluate the model. The model is evaluated once for each validation set, making the average of the rounds a good measure for performance.

The best parameters found in cross validation are used. In cross validations, one parameter changes while other parameters are kept at a reasonable constant that resulted in improvement at each epoch. We first determine the number of hidden units followed by learning rate and finally the number of epochs. We also used cross-validations to choose the template for classification task. From cross-validation results, decaying exponential template with $\lambda = 2/N$, where N is the number of hidden units shows the best performance among all other templates.

The task in MNIST classification is to classify each input image to one

of the digit classes 0 to 9. We reshape each image from 28×28 to a vector of size 784. The values of each image pixel is an 8-bit integer with value between 0 (black) to 255 (white). We divide these numbers by 255 to linearly scale them to single floating point value 0 to 1. Cross-validations in MNIST dataset is done using 50,000 training and 10,000 for cross validating from the 60,000 training. We then train the network using the best number of epochs and learning rate from cross-validation, using all the 60k training examples. Cross-validations shows that 3000 hidden units is the best number of hidden units for the algorithm to run efficiently. Although higher hidden units improvs the classification, but tradeoff between speed and accuracy is too large to justify increasing the number of hidden units. We use learning rate 0.005 over 100 epochs, where each epoch is training over the whole training set once. The classification results are discussed in the next section.

Task performed on Yale face dataset is to do identity classification. There are 15 classes, 1 for each individual and 11 pictures from each class. Each image is reshaped from 60×80 to a vector of size 4800. The values of each image pixel is an 8-bit integer with value between 0 (black) to 255 (white). We divide these numbers by 255 to linearly scale them to single floating point value 0 to 1. In Yale face database, 88 images, 8 image per individual is chosen for training set and 33 images, 3 image per individual is chosen for test set. Cross validation is done using 5 for training and 3 for validation to determine the parameters. We then train the network using the best number of epochs and learning rate from cross-validation, using all the 88 training examples. Since this is a small data set we used 5-fold cross validation by randomly splitting the original set into training and testing sets, from the original set. Result is the average of all 5 runs. Cross-validations shows 36 hidden units to be the best number of hidden units to be able to achieve highest accuracy. We use learning rate of 0.0005 over 50 epochs, where each epoch is training over the whole training set once. The classification results are discussed in the next section.

5.3 Classification Results

In supervised learning experiments, we perform classification tasks with high accuracy. In Yale dataset [7], we achieve 98.9 percent accuracy. Table 5.2 shows classification results for the Yale face dataset, where it is trained on 4 and 8 image per class for 50 epochs and tested on the remaining images. Figure 5.30 shows the 36 features extracted using the supervised algorithm from the Yale dataset. These features resemble the individuals in the dataset with their most prominent and discriminating feature having high values. Hair style, forehead, and eyes are among some of these features.

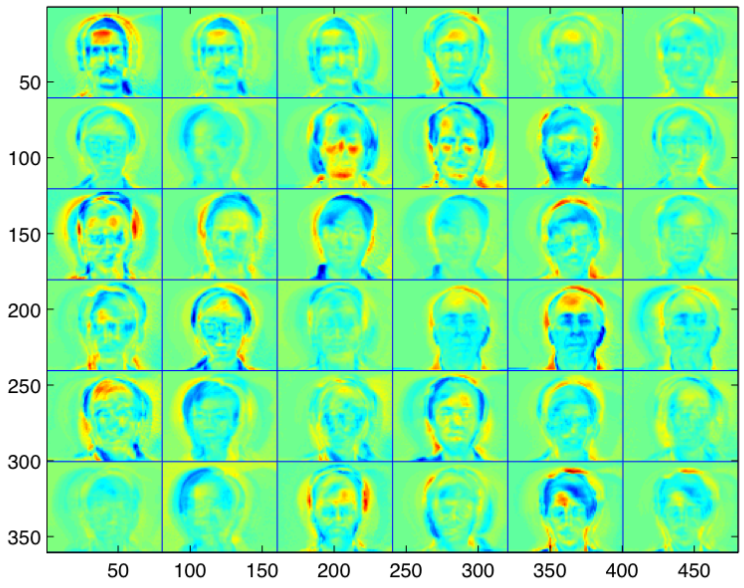

Figure 5.30: 36 features extracted from Yale database [7] using the supervised algorithm.

In MNIST dataset [15] we achieve 97.2% test accuracy. This results is competitive with other non-convolutional neural networks models. Most classification results on MNIST dataset are hosted on `http://yann.lecun.com/exdb/mnist/`. Our model is trained on 60,000 examples for 100 epochs and tested on 10,000 examples. Figure 5.31 shows 100 out of 3k features extracted using the supervised

algorithm from the MNIST dataset. These features are mostly the non-overlapping regions of digits that are common among a digit. These features are very useful in classification of one digit from all other classes. From experiments, we observe fast

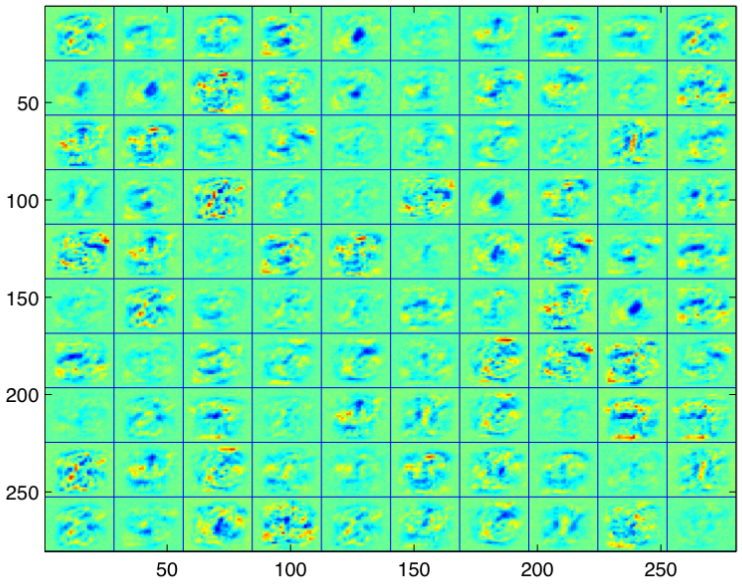

Figure 5.31: 100 features extracted from MNIST dataset [15] using the supervised algorithm.

drop in classification error during the first few epochs. Training and test error drop after small number of iteration over the training set. On average two pass over the training set results in around 30% error and drops to close to 0 after the 5th iteration. However, this is highly dependent on choosing the right learning rate, but a good learning rate can easily be determined in the cross-validation process.

Another interesting observation of this learning algorithm is the decrease in the validation error pass the iteration that makes the training error zero. The reason for the continuing improvement may can be explained by that learning algorithm is never trained on minimizing classification error but it is extracting features that separate the input space very well for the discrimination task. Even after the training set classification accuracy reaches 100%, the algorithm keeps on separating the space with larger margins. Our experiments also show that

Table 5.2: Test Error of Different Models on the Yale Set

Method	**Error(4 training)**	**Error (8 training)**
Our Model	2.2%	1.3%
MGFR+$(2D)^2$PCA [25]	1.8%	1.1%
2D Laplacianfaces [5]	4.7%	2.2%

the supervised model is able to extract and select features that are powerful in classification task.

Chapter 6

Conclusion and Discussion

Our experimental results show that our algorithm can learn activation profiles and features that would otherwise require very complex neural networks with lateral connections in each hidden layers. Although our analysis is limited in the unsupervised case, from the extracted features, we can conclude by observation that our features have interesting semantics. For example, features extracted from natural images resemble edges and small blobs, much like the ones seen in the human visual pathway, more specifically V1. Our unsupervised model has also shown to be a good generative model.

In the supervised setting, even though the learning is very similar to the unsupervised learning, a different way of extracting features emerges. Our supervised features are designed to maximize the discrimination without minimizing any cost function directly related to classification error. In the MNIST dataset, we extracted features mostly containing non-overlapping regions of digits that are common among a digit class. These features are very useful in classification of one digit from all other classes. In the Yale dataset, we extracted features resembling the individuals in the dataset with their most prominent and discriminating feature being emphasized. Hair style, forehead, and eyes are among some of these features.

We show that one can capture some of the complex properties of a sophisticated system with a simple approximation such as what we have done by using templates to model the effect of lateral connection and create an efficient algorithm

that can run in real time on todays personal computers.

Most feature extraction methods cannot be done online and those that are done online do not result in features found in human visual system. In addition, our algorithm has high amount of flexibility because of the many choices of templates.

Our algorithm is fast and efficient and converges quickly. Another advantage in using our algorithm is its online nature. Online algorithms can be used when the input data is streaming or when we have limited memory to store and have to work with smaller batches of training data. Our vision system is developed over time with the streaming inputs from our retina. However our algorithm has some disadvantages.

A disadvantage of our learning algorithm is its susceptibility to small transformations in the input space. This is a big problem for most algorithms. For example, in image data transformations, like translation, rotation, and scale are very common. A solution to this problem is to use convolutional networks that can handle some amounts of translation and rotation. Convolutional networks use the idea that each neuron in early visual system has a limited receptive field and cover small patch of the image in the field of view.

6.1 Future Work

For future work, we would like to use our algorithm in a semi-supervised setting. We suggest that by switching between the two algorithms, supervised and unsupervised, introduced in this work we would discover interesting effects in both features and classification results.

Another interesting experiment is learning models that have more than two layears. Models that use sparser templates for higher layers can be used to capture more abstract concepts and create a hierarchical model similar to the human visual system. Even though it is difficult to visualize features in higher layers, applying multi-layer of feature extraction and comparing the classification result can show how useful sparse features that are extracted from distributed features in lower layer can be.

As discussed in the conclusion, our learning algorithm is not invariant to transformations in the data. For example, in images an object could have number of transformations such as translation, rotation or scale. These translations are very commmon. To resolve this problem we propose a convolutional network learned using our method, where each layer is composed of many small networks that cover a small portion of the input. Convolutional network have in past been used to tackle the invariance problem.

Bibliography

[1] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[2] GL Barrows and JC Sciortino Jr. A mutual information measure for feature selection with applicationto pulse classification. In *Time-Frequency and Time-Scale Analysis, 1996., Proceedings of the IEEE-SP International Symposium on*, pages 249–252, 1996.

[3] P.N. Belhumeur, J.P. Hespanha, D.J. Kriegman, et al. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.

[4] R. Bellman. Adaptive control processes: a guided tour. 1961.

[5] N. Ben, S.C.K. Shiu, and S.K. Pal. Two Dimensional Laplacianfaces Method for Face Recognition. *LNCS RSCTC*, 2006.

[6] G. Carneiro and N. Vasconcelos. Minimum Bayes error features for visual recognition by sequential feature selection and extraction. In *Computer and Robot Vision, 2005. Proceedings. The 2nd Canadian Conference on*, pages 253–260, 2005.

[7] Yale Database. The yale face database. `http://cvc.yale.edu/projects/yalefaces/yalefaces.html`.

[8] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification*. Wiley New York, 2001.

[9] Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. 1994.

[10] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 148–156. Citeseer, 1996.

[11] AP Georgopoulos, AB Schwartz, and RE Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416, 1986.

[12] PE Gill, GH Golub, W. Murray, and MA Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, pages 505–535, 1974.

[13] A. Grunewald and E.K. Skoumbourdis. The integration of multiple stimulus features by V1 neurons. *Journal of Neuroscience*, 24(41):9185–9194, 2004.

[14] JJ Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554, 1982.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] R.M. Neal. *Bayesian learning for neural networks*. Springer Verlag, 1996.

[17] S. Pang, S. Ozawa, and N. Kasabov. Incremental linear discriminant analysis for classification of data streams. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(5):905–914, 2005.

[18] R.Y. Rubinstein and D.P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer-Verlag New York Inc, 2004.

[19] T.D. Sanger. A tree-structured algorithm for reducing computation in networks with separable basis functions. *Neural Computation*, 3(1):67–78, 1991.

[20] F. Song, H. Liu, D. Zhang, and J. Yang. A highly scalable incremental facial feature extraction method. *Neurocomputing*, 2008.

[21] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[22] V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 2000.

[23] M. Vasconcelos and N. Vasconcelos. Natural Image Statistics and Low-Complexity Feature Selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 228–244, 2009.

[24] M.J. Wainwright, O. Schwartz, and E.P. Simoncelli. Natural image statistics and divisive normalization: modeling nonlinearities and adaptation in cortical neurons. *Statistical theories of the brain*, pages 203–222, 2002.

[25] L. Wang, Y. Li, C. Wang, and H. Zhang. Face Recognition using Gaborface-based 2DPCA and (2D) 2PCA Classification with Ensemble and Multichannel Model. In *IEEE Symposium on Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007*, pages 1–6, 2007.

[26] MP Young and S. Yamane. Sparse population coding of faces in the inferotemporal cortex. *Science*, 256(5061):1327–1331, 1992.