WORCESTER POLYTECHNIC INSTITUTE

Robust Auto-encoders

by

Chong Zhou

A thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Data Science

April 2016

APPROVED:

Professor Randy C. Paffenroth, Adviser:

_____

Professor Carolina Ruiz, Reader:

_____

Contents

WORCESTER POLYTECHNIC INSTITUTE

Abstract

Data Science

Master of Science

by Chong Zhou

In this thesis, our aim is to improve deep auto-encoders, an important topic in the deep learning area, which has shown connections to latent feature discovery models in the literature. Our model is inspired by robust principal component analysis, and we build an outlier filter on the top of basic deep auto-encoders. By adding this filter, we can split the input data X into two parts $X = L_D + S$, where the L_D could be better reconstructed by a deep auto-encoder and the S contains the anomalous parts of the original data X. Filtering out the anomalies increases the robustness of the standard auto-encoder, and thus we name our model "Robust Auto-encoder". We also propose a novel solver for the robust auto-encoder which alternatively optimizes the reconstruction cost of the deep auto-encoder and the sparsity of an outlier filter in pursuit of finding the optimal solution. This solver is inspired by the Alternating Direction Method of Multipliers, Back-propagation and the Alternating Projection method, and we demonstrate the convergence properties of this algorithm and its superior performance in standard image recognition tasks. Last but not least, we apply our model to multiple domains, especially, cyber-data analysis, where deep models are seldom currently used.

Chapter 1

Introduction

Recently, the success of deep learning in multiple areas has drawn great interests of researchers [3, 7, 11–13]. Deep learning is part of a broader family of machine learning methods based on learning representations [3]. Learning and discovering informative features are deep learning's keys to success. Among deep learning models, we particularly focus on auto-encoders which show a promising connection to latent feature discovery models [7]. A typical auto-encoder tries to recover the input X through an encoding phase $E(\cdot)$ and a decoding phase $D(\cdot)$. The best recovery of the input suggests that the encoder $E(\cdot)$ and the decoder $D(\cdot)$ are informative feature discovery processes. As shown in [7], the auto-encoder model is identical to the principal component analysis if $E(\cdot)$ and $D(\cdot)$ are both linear projections and the reconstruction cost is the Euclidean distance [7]. However, people tend to apply non-linear encoders and decoders which are more efficient in tackling complex real-world problems. Thus, the auto-encoder is usually used as a generalized dimension reduction framework [3, 4, 7]. However, this dimension reduction process may be misled by outliers.

We improve the basic auto-encoder model by building an outlier filter. The idea of our improvement comes from robust principal component analysis [5, 8, 9] which splits the input X into L_0 and S_0. The S_0 matrix contains element-wise outlying parts while the L_0 retains the majority of X. We apply this idea where input X could be split into S and L_D. The S contains the outliers and L_D is the input of an auto-encoder which will archive lower reconstruction error than the basic one. Here, S is an outlier filter and the left part L_D will be easily reconstructed by an auto-encoder. After training, the hidden layers in the auto-encoder are informative features that widely influence the majority of observations while S has outlying parts that usually hint at negative labels. This splitting, $X = L_D + S$, makes the traditional auto-encoder solver,

the back-propagation method [3, 17, 21], unsuitable in this new case, since the outlier filter S can not be trained by back-propagation. We follow the idea of the alternating direction method of multipliers(ADMM) [15] which are modified based on inspiration from Dykstra's work [31]. Our solver alternatively makes use of the back-propagation and the shrink function [5], then applies Dykstra's idea to maintain the constraint through iteration until the object converges. Lastly, we test our model on the MNIST data set, the notMNIST data set, and also a packet data set.

1.1 Motivation

Principal component analysis (PCA) is a widely used dimension reduction method [8], which orthogonally projects a set of observations from a high dimensional space to a lower dimensional linear manifold. It returns a low dimensional representation where each elements of this representation are linearly independent with each other. This projection is a lossy compression [7]. Lossy compression means that PCA only keeps the information on the manifold and omits the information outside the manifold. The general purpose of data compression and dimension reduction is to minimize the information loss. When we apply PCA, we assume that the observations of input data are laying *near* a *linear* manifold in a high dimensional space, and the data are *slightly* corrupted. PCA will keep the information on the manifold, thus, when we project data on this manifold, the majority of information will be represented on a low dimensional manifold. PCA achieves dimension reduction without large information lose. However, the assumption of linearity and uncorrupted is not always held on realistic data sets. First, if the data lies near a non-linear manifold, the linear manifold will not be a sufficient representation for capturing the non-linear relationship between observations, and thus requires introducing non-linearity into the projection process. We use auto-encoders [3] to implement a generalized projection framework, and the non-linearity is introduced by a non-linear activation function of an auto-encoder. Second, outlying observations may mislead the direction of the manifold. Thus, a method for eliminating the outlying influential observations is necessary to apply. Robust principal component analysis (RPCA) [9] offers a way to mitigate the negative effects of outliers.

1.2 Contribution

Our research is based on auto-encoders [3] and draws inspiration derived from the idea of Robust PCA. We build a novel model which improves both shallow and deep auto-encoders by adding sparse outlier detection. This combination inherits advantages from both techniques, using robust PCA to provides robustness to anomalies and auto-encoders to provide non-linear feature representations. We call this novel method a Robust Auto-encoder.

Further, we extend the idea of a shallow robust auto-encoder to a robust deep learning model. Auto-encoders are a building block commonly used in deep learning [3]. Inspired by the concept of deep learning we extend our model to robust deep auto-encoder for capturing more complex non-linear relationships.

Third, we developed a new training algorithm for our model. Back-propagation is an essential element of deep auto-encoder training. While optimization techniques exist for back-propagation and training deep auto-encoder [3, 7] and robust PCA (e.g., by using the ADMM) [15], we are not aware of any other techniques that can simultaneously optimize both. In our work, we alternatively train the auto-encoder using back-propagation and the outlier filter using shrinkage function. Finally, inspired by an idea from the Dykstra's alternating projection method [29] we build a projection operator to satisfy the constraint and give an optimization scheme for the combined algorithm.

Finally, we apply our robust deep auto-encoders and standard auto-encoders on image recognition data sets and a cyber analysis data set.

Chapter 2

Background

In this chapter, we will introduce necessary backgrounds that form our ideas and algorithm. Our research starts from PCA. PCA is a fundamental dimension reduction method in data analysis. Robust PCA [5] and auto-encoder [3, 7] improve PCA in two distinct ways. Robust PCA enhances the reliability of PCA dimension reduction process while auto-encoders introduce non-linearity to dimension reduction. We merge Robust PCA and auto-encoders to inherit both merits. In this section, we first introduce the starting point, PCA, and then describe how robust PCA learns robust features. Further, we step into the deep learning area [3, 7]. Among various models in deep learning, the auto-encoder is one of the cutting-edge research directions in this area. We describe what the auto-encoder model is and how the auto-encoder introduce non-linearity to its dimension reduction process.

The second part of this chapter lists training methods in the literature. We describe some details about back-propagation which is used for training the auto-encoders [7]; an optimization method called alternating direction method of multipliers (ADMM) [15] for optimizing robust PCA; and Dykstra's alternating projection method which inspires us to develop our training algorithm [29].

2.1 Principal Component Analysis

High dimensional data often suffers from an issue called "the Curse of Dimensionality" [3], which refers to various phenomena that arise when analyzing data in high-dimensional spaces that do not occur in low-dimensional spaces, such as the overfitting, and a common solution for this problem is dimension reduction. PCA is a widely used machine learning algorithm for

dimension reduction. PCA projects data from a high N dimension space to a lower M orthogonal dimensional space($M < N$) which has k-highest variance among the all M-dimensional space [19]. This low dimensional space is a linear manifold in the high dimensional space, and thus, the relation of data is compressed and represented by this linear manifold and the rest of the information outsides this manifold is ignored. PCA learns a linear representation where the entries of this representation are linear combinations of the original data. In the picture 2.1, we give an intuition example about PCA projection.

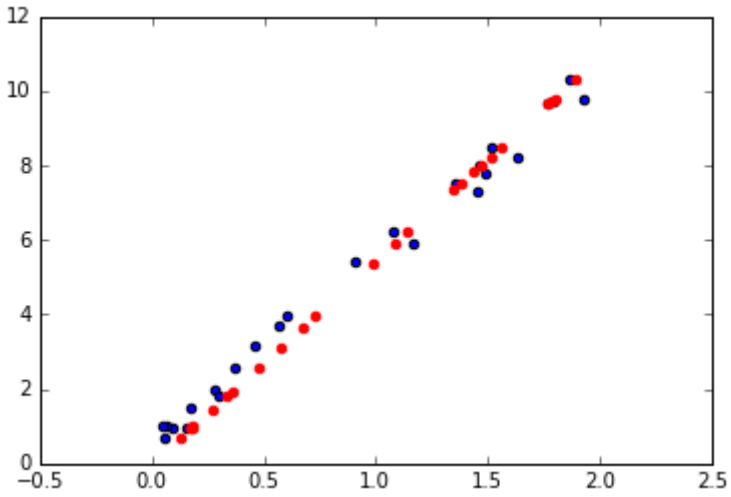

Figure 2.1: PCA works well while all observations lie near a linear manifold and data are slightly corrupted. The blue points are simulated data of two dimensions. In this case, one can find that the PCA projection (red line) successfully captures the trends of these observations.

2.1.1 PCA and Singular Value Decomposition

PCA projection can be efficiently implemented by the singular value decomposition (SVD) [19]. Given a matrix $M \in R^{m \times n}$, the singular value decomposition is $M = U\Sigma V^T$, where the U and V are unitary matrix. The singular values of M lie on the diagonal of the matrix Σ. The rank of M is equal to the number of non-zero singular values. Additionally, the rank is equal to the number of linearly independent rows (or columns) the matrix contains [30]. For a given matrix $M \in R^{m \times n}$, the m is the number of observations, and the n is the dimension of each observation. We assume that $n >> m$. Thus, the rank of M depends on the number of linearly independent columns of the matrix M. Linear dimension reduction on M is equal to reducing the rank of M.

Thus, the key of PCA projecting M from n dimensions to k dimensions ($k < n$) is the SVD decomposition. PCA first decomposes M though SVD; then keeps the k-highest singular values

in Σ and sets the rest of the $n-k$ singular values to 0 to get a $\overline{\Sigma}$; lastly, PCA reconstructs M by $\overline{M} = U\overline{\Sigma}V^T$ [30]. $\overline{M}$ only has k dimensions since the number of its non-zero singular values is k.

2.2 Robust PCA

PCA seeks a low dimensional representation of an input data, but it only works well when the input data are slightly corrupted. One shortcoming of PCA is its sensitivity to large corruptions and outlying observations [5]. Robust Principal Component Analysis (RPCA), introduced by J. Wright, Y. Peng, etc., decomposes an input matrix X into a low-rank matrix and a sparse matrix $X = L_0 + S_0$. The low-rank matrix L_0 contains our interested pattern [5] and the sparse matrix S_0 consists of element-wise outlying parts which can not be captured by low-rank pattern L_0. We constrain the rank of matrix L_0 as low as possible and the sparse matrix S_0 element-wisely as sparse as possible. The L_0 could be represented by a linear manifold while the S_0 is a filter that peels the faraway part from the linear manifold.

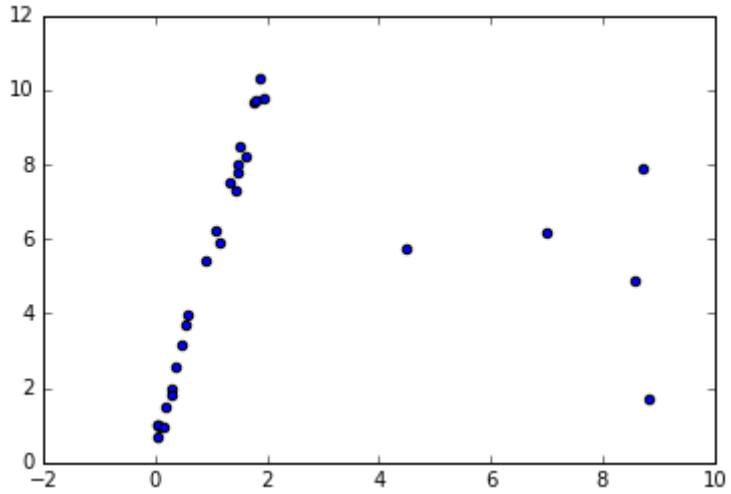

Figure 2.2: PCA does not work well with outliers. This set violates one assumption of PCA that all observations are slightly corrupted. In this set, there are some outlying observations. These outlying observations, for some known or unknown reason, are heavily corrupted. The whole set can be split into two parts: the majority and the outliers.

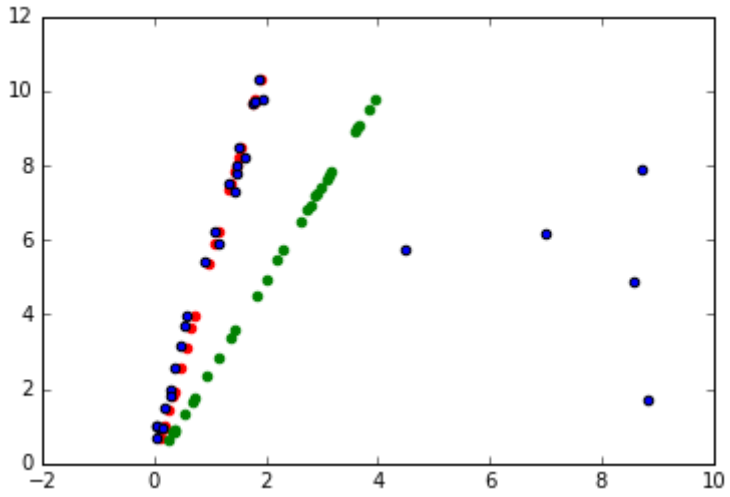

Figure 2.3: This toy data set shows that PCA does not work well when outliers exist. The blue points are simulated observations in two dimension. We want to project the data on one dimension and retain as much information as possible. One can tell that the majority of points follows a clear trend (the red points) from lower left to upper right, but also, there are some outliers far away from this trend. The green points show the result of PCA projection of all observations (include the outliers). It is clear that this linear manifold of PCA shifts to the right to offset the huge errors of those faraway outliers. This shifting will harm the information preservation for those normal observations. We say PCA projection is distracted by the outliers and sensitive to outlying observations. Robustness is needed to eliminate the influence of those outliers.

Robust principal component analysis (RPCA) refines PCA by making PCA robust to outliers. RPCA allows for the careful teasing apart of sparse outliers so that the remaining low-rank approximation is faithful to the true low-rank subspace describing the raw data [5, 8–10]. We argue that our input data M can be decomposed into three parts

$$M = L_0 + S_0 + \varepsilon,$$

where L_0 is a low-rank matrix, S_0 is a sparse matrix which can not be captured by the low rank features, and ε is a point-wise error matrix. This matrix decomposition is determined as the following optimization problem [8]:

$$\underset{L_0, S_0}{argmin} \quad ||L_0||_* + \lambda ||S_0||_1$$

$$s.t. \; |M - L_0 - S_0| \preceq \varepsilon$$

where the $||\cdot||_*$ is the nuclear norm, and $||\cdot||_1$ is the one norm. The nuclear norm of a matrix M

is defined as $||M||_* = \Sigma_{j=1}^{k}\sigma_j(M)$, or, the sum of the k largest singular values of M [5]. Linear dimension reduction of M is also equal to reducing the nuclear norm of M as the singular values of 0 will not contribute to the sum.

2.2.1 Feature Learning with Robust PCA

Robust PCA splits the input data X into two parts: a low-rank matrix L_0 and a sparse matrix S_0. The low-rank matrix L_0 can be interpreted as the background features that widely influence all observations. In other words, L_0 matrix can be almost projected to a manifold through a linear projection, and this manifold is a collection of features that are extracted from the original input data. It is the best estimation of every observation through a linear combination of these features. Meanwhile, the sparse matrix S_0 contains anomalies which can not be captured by low-dimensional features. This sparse matrix is like a strainer that filters out distinguishing parts for every observation. This "strange" part is hard to estimate through the linear combination of background features, so after peeling, L_0 is more accurate to capture background features of the majority. The robust PCA discovers the linear background features L_0 and outliers features S_0 at the same time. Anomalies in S_0 do not necessarily imply negative or positive labels, but such sparsely correlated phenomena often bear closer examination in real-world problems [5].

2.3 Deep Learning

Deep learning is a subfield of machine learning that is based on learning several levels of representations, corresponding to a hierarchy of features, factors, or concepts, where higher-level concepts are defined from lower-level ones, and the same lower level concepts can help to define many higher-level concepts. Deep learning is part of a broader family of machine learning methods based on learning representations [3]. One promise of deep learning is to replace handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction [6]. Most deep architectures use unsupervised learning across multiple layers [3]. Hidden layers in a deep model extract representations from data with the goal of constructing new features that make standard learning algorithms more efficient. The key idea of deep learning is to learn representations, and then make a prediction based on representations learned [3]. Deep learning is widely used for pattern recognition and feature learning. The procedure of deep learning processing includes many layers of non-linear

processing stages. The output of the lower layer is used as input for the next layer's processing.

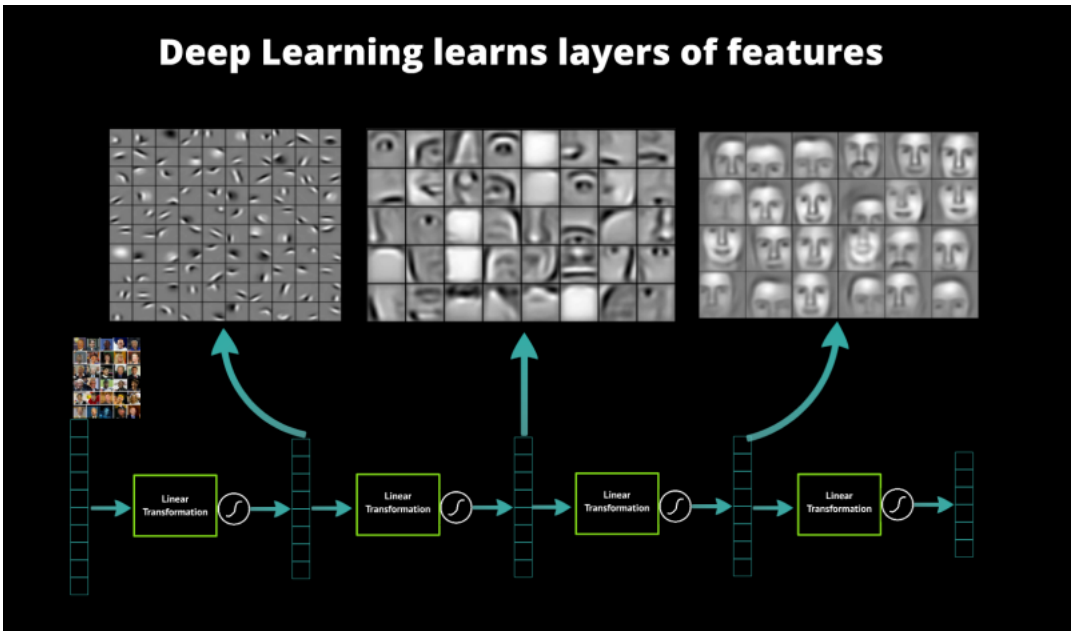

Figure 2.4: An illustration of representation learning: Deep learning makes predictions based on representations. This picture shows that the raw input representation is pixels, and it is hard to predict human face based on raw pixels since pixels vary dramatically, and every pixel gives a little distinctive information. People can hardly to predict the label of a given picture from the collection of pixels. A good prediction needs to extract information from pixels and construct the informative features. The left bottom is the raw image input consisting of pixels. The first layer extracts a new representation from these pixels. This new representation looks like all different kinds of lines which are some non-linear combination of raw pixels, and these lines are the input to the next layer and will be further combined with each other to form higher level representations. The second layer combines the input of "lines" and then forms a higher level representation which looks like different parts of faces. These fractional parts of faces then are continuously learned by the next layer to extract "face-like" features. The final prediction is made on this features: how much similarities of a new given face to these "face-like" features. Making a prediction on these features is much more accurate than the one based on original input represented by raw pixels and any representation learned in the previous layers [3]. Hence, the key idea of deep learning is to learn the representations that helpful to predict labels. These representations are extracted based on a forward multiple layer neural network that each layer learns representation based on a previous layer. This picture comes from [23].

Modern deep learning provides a very powerful framework for supervised learning. By adding more layers, a deep network can represent functions of increasing complexity [7].

2.4 Auto-encoders

An auto-encoder is a feed forward multi-layer neural network in which the output target is the input itself. An auto-encoder is trained to copy its input to its output. This process seems speciously trivial, but the meaningful part is the dimension-reduced hidden layers learned to reproduce the input and thus these low dimensional hidden layers are trained to be lowest loss representations of the input. A typical auto-encoder with one hidden layer consists of three layers: an input layer, an output layer, and a hidden layer. The mapping from the input layer to the hidden layer is an encoder phase $E(\cdot)$ and the mapping from the hidden layer to the output layer is a decoder phase $D(\cdot)$. The mapping function of encoder $h = f(x)$ maps input x to a lower dimensional representation h. The decoder decodes the h in an opposite way. It is a mapping function $\overline{x} = g(h)$ which the reconstruction $\overline{x}$ has same dimension with the input x. Minimizing the reconstruction error $e = x - g(h(x))$ and back-propagating the reconstruction error is the training process. The low dimensional representation h contains the original information with lower dimension and it is the information extraction target of auto-encoders. Intuitively, the similarity of the output and the input in a trained auto-encoder ensures that the hidden layers h are informative features. The encoder and decoder work mutually to draw informative features: 1. The encoder codes the input data to low dimension space, but there are millions of ways to code the input data. 2. The decoder constrains the encoding phase by requiring the encoded hidden layer could be reconstructed. A typical auto-encoder structure is shown as following:

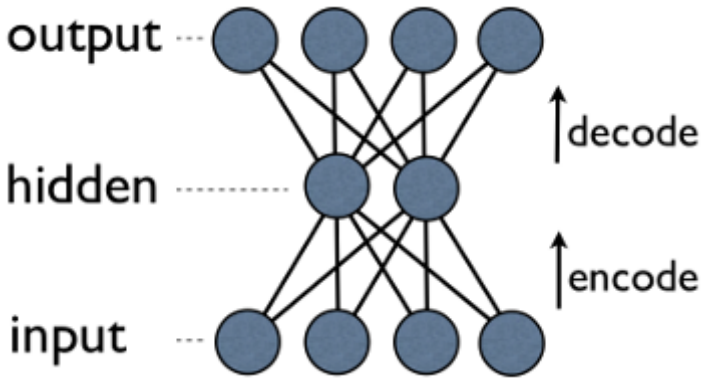

Figure 2.5: In an auto-encoder, the bottleneck-shaped hidden layer has lower dimension than the input layer. The hidden layer representation contains sufficient information in the lower dimension. The minimized cost support that the output has lowest information loss through an encoding-decoding channel. Thus, we say that the information is extracted by auto-encoder.

2.4.1 Non-linear Activation Functions

If we apply a linear mapping function as an encoder $h = W \cdot X$ and a decoder $\overline{X} = W^T h$, and choose the Euclidean distance $||X - \overline{X}||_2$ to measure the error between original input and reconstruction $\overline{X}$. Then this auto-encoder is exactly as same as PCA. To introduce non-linearity into auto-encoder, we apply non-linear activation functions after linear projection. The encoder with non-linear activation function is $h = f(W \cdot X + b)$, and the decoder has the similar formula $h = f(W^T \cdot X + b)$. The commonest activation function is sigmoid function [7]:

$$f(t) = \frac{1}{1 + e^{-t}}$$

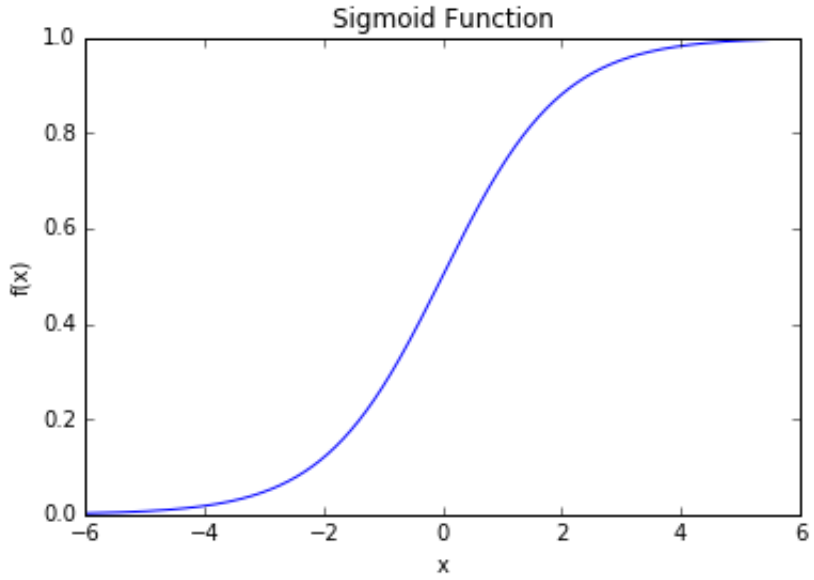

Figure 2.6: This picture shows the sigmoid function. The role of the sigmoid function is to introduce non-linearity into auto-encoder.

2.5 Deep Auto-encoders

An auto-encoder with a single encoder and decoder is usually considered as a shallow model. The way of extending shallow auto-encoders to deep auto-encoders is to add more encoding and decoding phases. Researchers [3, 4, 6, 7] have developed different ways to extension. In this section, we briefly introduce two kinds of deep auto-encoder the standard deep auto-encoder and the stacked deep auto-encoder. When a model is extended to deep, non-linearity is required to make deep models non-trivial. For example, a deep auto-encoder with linear encoders $E(\cdot)$

and linear decoders $D(\cdot)$ has the same functionality as only one hidden layer auto-encoder, since the associative law of matrix dot product.

$$A_1 \cdot A_2 \cdot x = A \cdot x$$

This property indicates that multiple linear projection $A_1 \cdot A_2$ of an input x is equal to a single linear projection A. A successful deep auto-encoder requires the encoder mapping $g(\cdot)$ nonlinear.

2.5.1 Deep Auto-encoders

Usually, an auto-encoder with more than one hidden layers is called a deep auto-encoder [3]. One more hidden layer needs an additional pair of encoder $E(\cdot)$ and decoder $D(\cdot)$. Usually, the more encoders a deep auto-encoder has, the more complex a deep auto-encoder is. Notably, the multiple decoding phases should decode in an opposite way to the encoder. The last encoder should be decoded first. For instance, suppose a deep auto-encoder has three hidden layers, and three hidden layers need three pairs of encoder and decoder which are denoted as $E1(\cdot)E2(\cdot)E3(\cdot)$ and $D1(\cdot)D2(\cdot)D3(\cdot)$. The encoding phase of this deep auto-encoder is that $E1(\cdot)$ encodes the input x, then the $E2(\cdot)$ encodes the output of $E1(\cdot)$ and so on. The final code of encoding phase is $E3(E2(E1(x)))$. The decoding phase is executed in the opposite way with the last encoder being decoded first while the first encoder will be decoded last [3]. The final reconstruction of decoding phase is $Recontruction = D1(D2(D3(E3(E2(E1(x))))))$. The cost function is a distance function between the input x and $Reconstruction$.

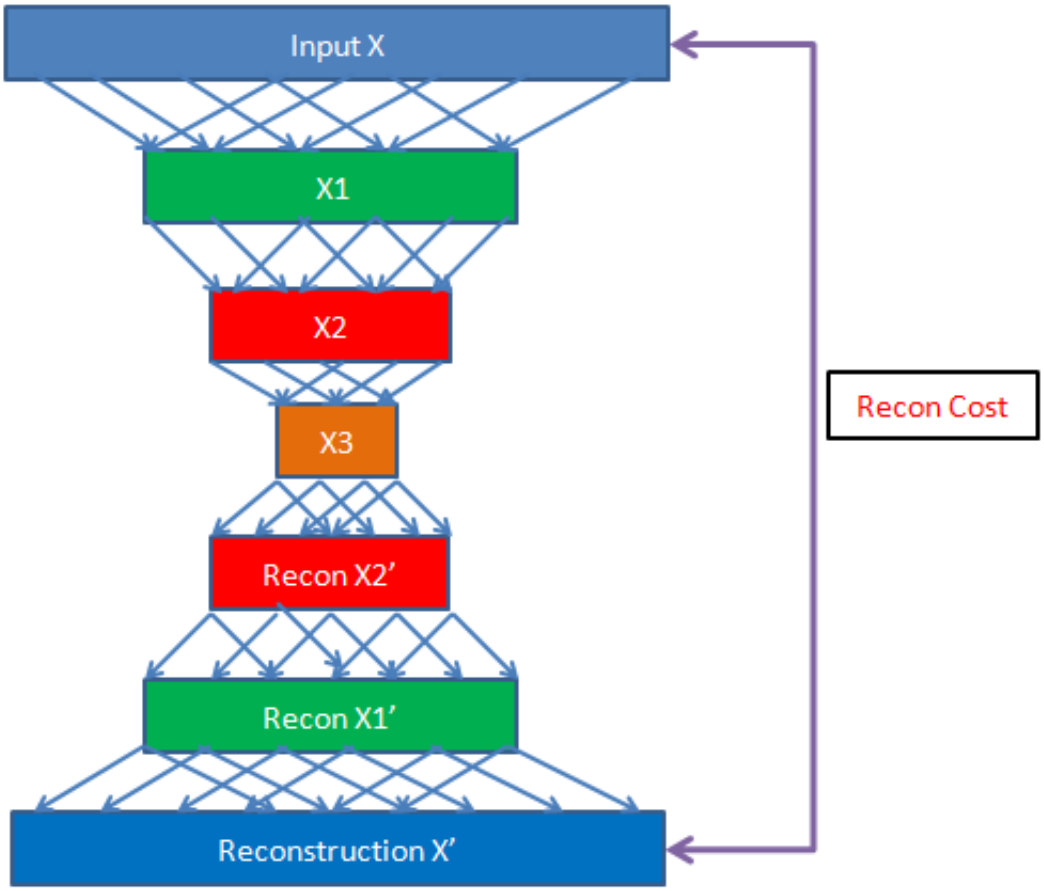

Figure 2.7: A deep auto-encoder is an auto-encoder with more than one hidden layer. It follows the idea that first encoder will be decoded last. On the top of the picture, X is the input data . In the encoding phase, the first encoder is $X1 = E1(X)$, the second encoder is $X2 = E2(X1)$, and the third encoder is $X3 = E3(X2)$. In the decoding phase, the last encoder should be decoded first. Thus, we decode the third encoder first $\text{Recon}X2' = D3(X3)$, the second decoder is $\text{Recon}X1' = D2(\text{Recon}X2')$, and the last decoder is decoding the first encoder $\text{Recon}X' = D1(\text{Recon}X1')$. The input data X goes through the pipeline to get the reconstruction $\text{Recon}X = D1(D2(D3(E3(E2(E1(X))))))$. The cost of whole deep auto-encoder is defined as $||X - \text{Recon}X||$.

2.5.2 Stacked Auto-encoders

A stacked auto-encoder is a multiple layer neural network [11]. A stacked auto-encoder has two parts: the frontal layers form the first part which is pre-trained by auto-encoders, and the target of these layers is to discover useful features. The second part only have the last layer. The final prediction makes use of representation learned from the first part. The weights of each layer are pre-trained by shallow auto-encoders. Every shallow auto-encoder is trained in a greedy layer-wise way that once one layer is trained, it will be set and will not be updated by back-propagated error. The input of next layer is the output of the previous layer, but

training process will not take other layers into consideration. The last layer of the stacked auto-encoder is a logistic regression layer for the final prediction. The fine-tuning phase starts after every layer gets pre-trained. In the fine-tuning phase, the raw input data will be encoded by the weights that are trained in pre-train phase. Then the logistic regression layer makes a prediction based on the final representation.

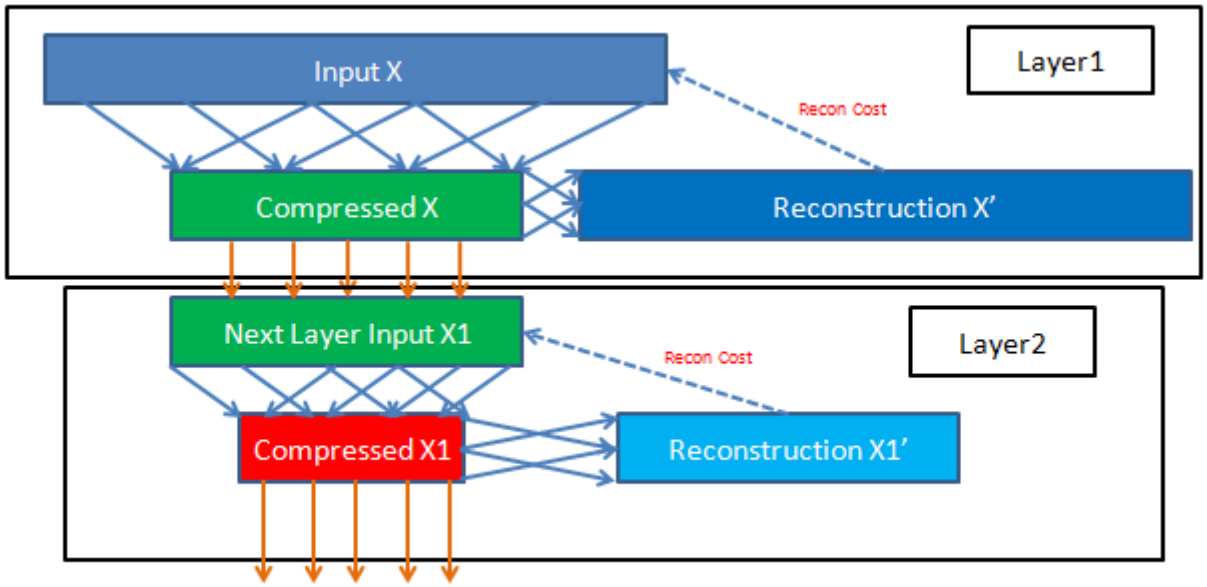

Figure 2.8: This picture shows the pre-train phase of a stacked auto-encoder: Each layer is a shallow auto-encoder. Multiple auto-encoders form a pipeline that the output of the previous auto-encoder is the input of next auto-encoder. Once one auto-encoder is trained, it will be set. The purpose of pre-train phase is to extract useful features. The features will be further used for prediction in fine-tuning phase.

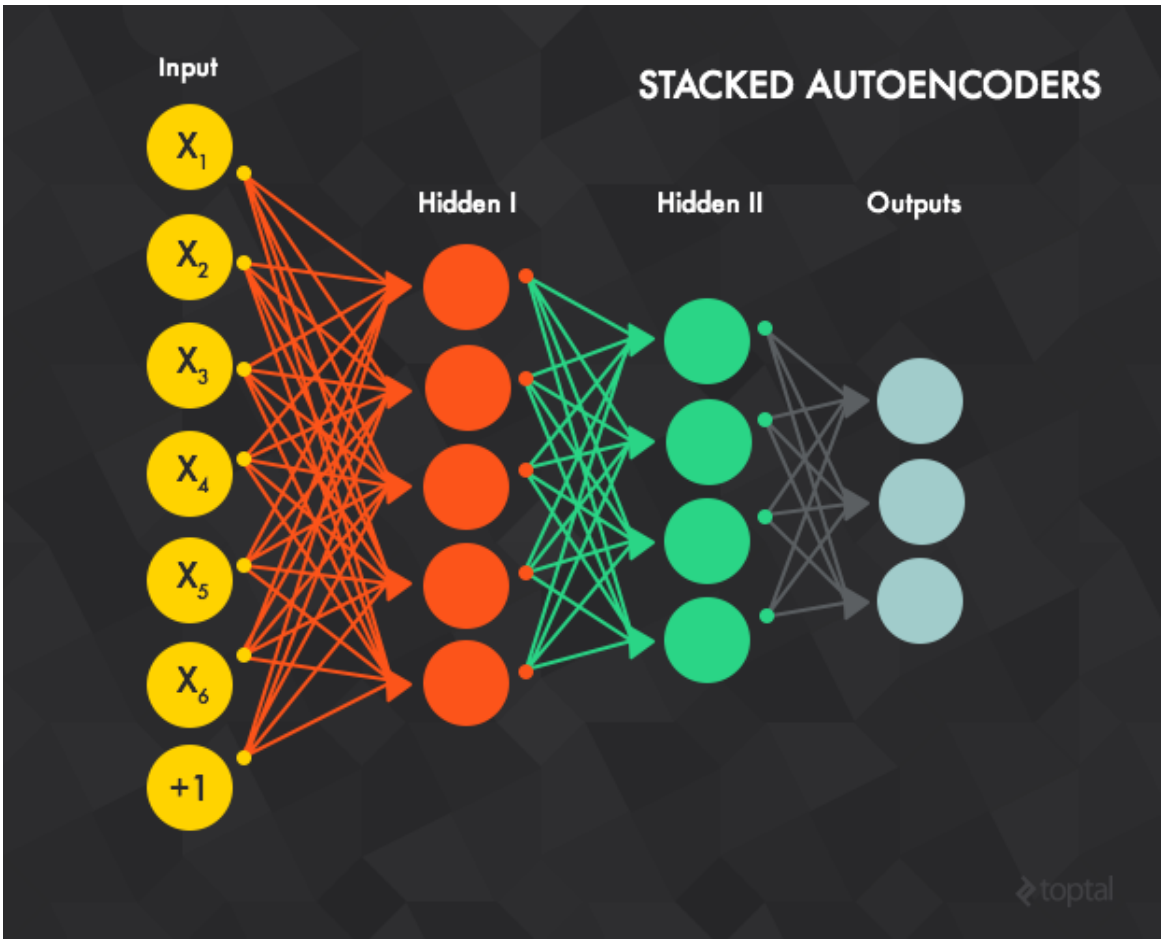

Figure 2.9: This picture shows a typical stacked auto-encoder. The weights of each layer are pre-trained by a shallow auto-encoder as shown in Figure 2.8. These pre-trained weights are the features learned from shallow auto-encoders. In this picture, the pre-train phase means learning the links between layers using shallow auto-encoders. We first apply a shallow auto-encoder to learn the red links between input layer (yellow layer) and hidden layer I (red layer). After the first shallow auto-encoder's training, we apply a second shallow auto-encoder to learn the green links between hidden layer I and hidden layer II (the green layer). These shallow encoders are stacked and trained in a greedy way. After pre-training, the fine-tuning phase uses these learned features to make the prediction (the gray output layer) [28].

2.5.3 Non-linear Feature Learning

PCA projects data onto a linear manifold in high dimensional space. However, this classic linear analysis is not ideal for discovering the non-linear low dimension representations, and we need to use some non-linear dimension reduction method. An auto-encoder is a promising method for introducing non-linearity. The auto-encoder maps an input data X to a lower dimensional representation and reconstructs the input data from the low dimensional representation of the original input dimension. Auto-encoders are generalized frameworks of non-linear dimension reduction processes of implementing projection and extracting informative features [7]. By

applying and adjusting non-linear activating function in the encoder and decoder, auto-encoders will be used to implement non-linear projection which is used in more sophisticated situations. The non-linear function between the layers enables us to curve the linear manifold. Hence, the auto-encoder offers generality to the dimension reduction process. The complexity and variability of many real world problems naturally require non-linear projection method.

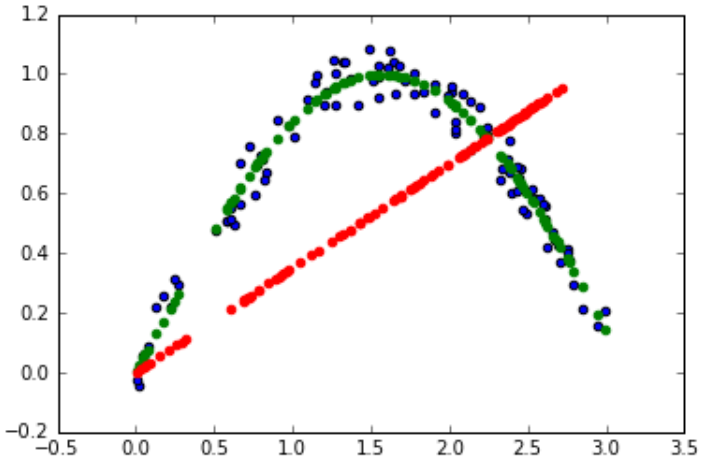

Figure 2.10: The blue points are simulated data that lie near a non-linear manifold. The green points show an optimized non-linear projection of data. The red points show the linear manifold which is projected by PCA. From this picture, we can see that PCA, as a linear projection, can not capture the non-linearity relationship of the data. The linear projection and reconstruction will contain a heavy information loss which ruins the dimension reduction process. However, the points in this data set indeed follow a low dimensional curve (the green points). Thus, we need to introduce non-linearity into our projecting process.

Deep auto-encoders are often used for learning representations or effective encodings of the original data, in the form of input vectors in hidden layers [3]. However, some classic types of auto-encoders could suffer losses in the presence of outliers. In particular, when learning a non-linear manifold from noisy data, outlying observations may mislead the direction or skew the curvature of the manifold. Eliminating the influence of outlying observations is a promising way to improve the robustness of deep models and makes them more applicable to real-world problems.

2.6 Training Methods

The auto-encoders and deep auto-encoders, as multi-layer neural network models, usually use the back-propagation method to train [7], while the robust PCA uses the ADMM method [5,15]. Our model is a combination of the auto-encoder and the robust PCA, but the training method

can not directly combine the ADMM and the back-propagation. Our new training method borrows an idea from the alternating projection method [29] [31]. In this section, we briefly introduce back-propagation, ADMM, and alternating projection method. These three methods work mutually to compose our algorithm that consists of two separating training steps: a back-propagation trainer of the auto-encoder and a shrinkage trainer for the outliers filter. Between them, the alternating projection accompanies with both two trainers to force them satisfying the constraint. In the area of deep learning, back-propagation of errors is a common method to train the weights while minimizing non-smooth one norm term often use a shrink function. Alternating projections method is less popular, but it is a powerful tool to check the intersection of two convex set. We use alternating projection method to check the constraints.

2.6.1 Back-propagation

Back-propagation, an abbreviation for "backward propagation of errors", is a common method for training deep neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respect to weights in the auto-encoder. The gradients are used to update the weights, in an attempt to minimize the loss function [21]. A typical back-propagation has two phases: 1. Forward propagation to compute the output of neural network and 2. Back propagation of errors to update the weights of the neural network. For example, we build a auto-encoder with an encoder $E(\cdot)$, a decoder $D(\cdot)$ parameterized with the weights $W \in R^{m \times n}$ and bias term b, and a activating function $f(\cdot)\ R^n \Rightarrow\ R^n$. The first phase is to compute the reconstruction $\overline{X}$ and the reconstruction cost which is defined as $||X\ -\ \overline{X}||_2$. The forward propagation phase to compute the reconstruction cost [22]:

Input $X \in R^{N \times m}$

$h\ =\ f(\ W\ \cdot\ X\ +\ b)$

$\overline{X}\ =\ f(\ W^T\ \cdot\ X\ +\ b)$

$cost =\ ||X\ -\ \overline{X}||_2$

return *cost*

Then, the second phase is to backward propagation of the cost to update the weight W and bias term b [22]:

1. Compute the gradient using the chain rule of calculus

$$\delta_W = \frac{\partial\ cost}{\partial\ h}\ \frac{\partial\ h}{\partial f(W)}\ \frac{\partial f(W)}{dW}$$

$$\delta_b = \frac{\partial\ cost}{\partial\ h}\ \frac{\partial\ h}{\partial f(b)}\ \frac{\partial f(b)}{db}$$

2. Update the weights by the gradient

$$W^{l+1} = W^l - \alpha \cdot \delta_W$$

$$b^{l+1} = b^l - \alpha \cdot \delta_b$$

2.6.2 Alternating Direction Method of Multipliers

The key idea of alternating direction method of multipliers (ADMM) is that a complex convex optimization problems can be broken into smaller pieces, and each piece is easy to solve [15,16]. For an optimization problem:

$$minimize \quad f(x) + g(z)$$

$$subject\ to \quad Ax + Bz = c$$

typical ADMM algorithm consists of the following iterations [15]:

while not converged:

$$x^{k+1} := argmin_x L_\rho(x,\ z^k,\ y^k) \tag{1}$$

$$z^{k+1} := argmin_x L_\rho(x^{k+1},\ z,\ y^k)) \tag{2}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1}c) \tag{3}$$

where $\rho > 0$. This algorithm teaches us how to split our complex objective function into small pieces, and each small piece is easier to handle. ADMM allows us to separately optimizing different parts of the objective function with other parts fixed. The ADMM splits original optimization problems into three pieces: an x-minimization step (1), a z-minimization step (2), and a dual variable update (3) Each minimization step can be easily implemented by a simple optimizing algorithm we mention before, e.g. gradient descent or stochastic gradient descent. This method has been proven the efficiency and correctness in [15]. However, our optimization problem does not strictly obey the conditions of ADMM. The first term is to reduce the reconstruction error by tuning the weights W and bias b in the auto-encoder, and the W and b does not appear on the constraints $X - L_D - S = 0$. But ADMM requires the input

parameter x of first term $f(x)$ must follow the constraint $Ax + Bz = c$. As a consequence, computing the Lagrangian multiplier according to x is impossible. So we can not directly use the Lagrangian multiplier and the ADMM, and we just borrow the key idea of ADMM, individually applying different optimizer on different parts of objective function will result in a converge a minimum (Details in section 3).

2.6.3 Alternating Projections Onto A Convex Set

The ADMM is used to minimize the two parts separately. Further, we learn an idea from the Dykstra's algorithm which helps us check the constraint in our model. Dykstra's algorithm is a method that projects a point onto the intersection of convex sets and is a variant of the alternating projection method (also called the projections onto convex sets method). In its simplest form, this algorithm finds a projected point on the intersection of two convex sets by iteratively projecting a point onto each of the convex set [29,31]. Figure 2.11 shows an example of this iterative projecting process. Dykstra's algorithm enlightens us, and we apply a similar algorithm to force multiple asynchronous optimizers following the constraint.

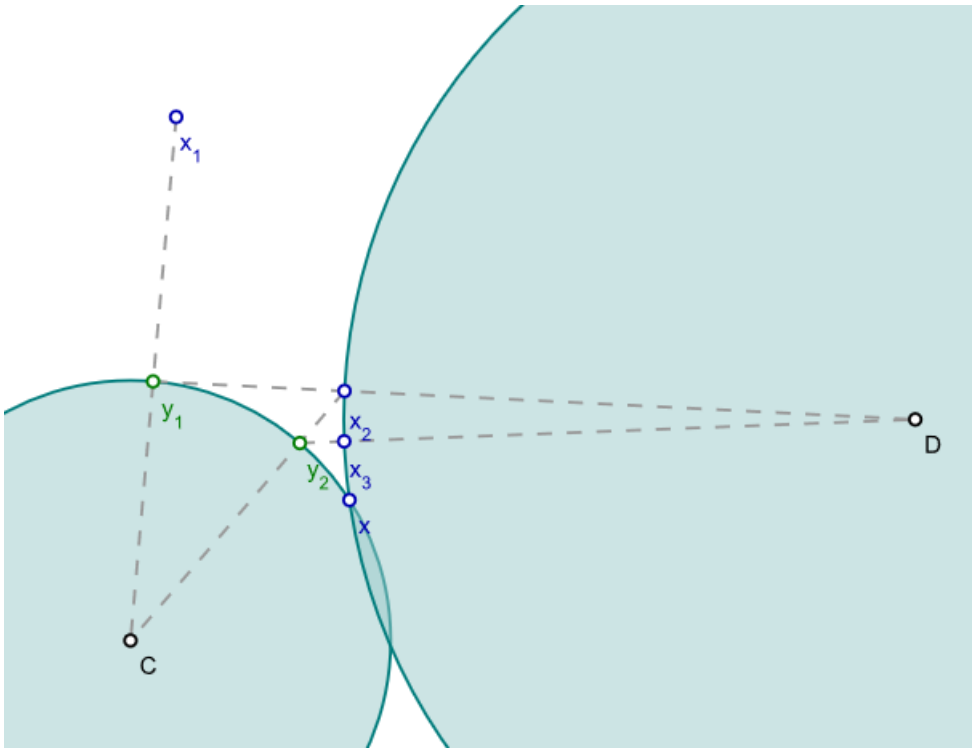

Figure 2.11: Alternating project a point onto an intersection of two convex set. From this picture, one can see: x_1 is a point outside of two convex sets. Dykstra's algorithm tells us how to find a way to project x_1 on the *intersection* of two sets x. In this picture, we follow the Dykstra's algorithm that starting with x_1 first projects x_1 on the convex set C to get y_1 and replaces the x_1 by y_1. Then we start with y_1, projects y_1 on another convex set D to get x_2 and replaces y_1 by x_2. Alternatively projecting on convex set C and D, we could find the correct projection point x which lies on the intersection of two convex sets.

Dykstra's algorithm starts with $x_0 = r$, $p_0 = q_0 = 0$ and update by

Iterating:

$$y_k = \mathcal{P}_D(x_k + p_k)$$
$$p_{k+1} = x_k + p_k - y_k$$
$$x_{k+1} = \mathcal{P}_C(y_k + q_k)$$
$$q_{k+1} = y_k + q_k - x_{k+1}$$

Chapter 3

Robust Auto-encoder

3.1 Model Definition

In many real world problems, non-linearity and outliers exist at the same time. To solve these two problems simultaneously, we are inspired by the spirit of Robust PCA to improve the basic auto-encoder model, and our improved model is a novel combination of auto-encoders and robust PCA. Following the robust PCA, we introduce a filter layer before normal auto-encoder. The filter layer carefully culls out the outlying observations that are different to reconstruct by an auto-encoder. Since it is a combination of robust PCA and auto-encoders, we call this combined model a *Robust Auto-encoder*. Robust auto-encoders inherit the advantage from both models. The outliers filter introduces robustness, and the auto-encoder gives nonlinearity.

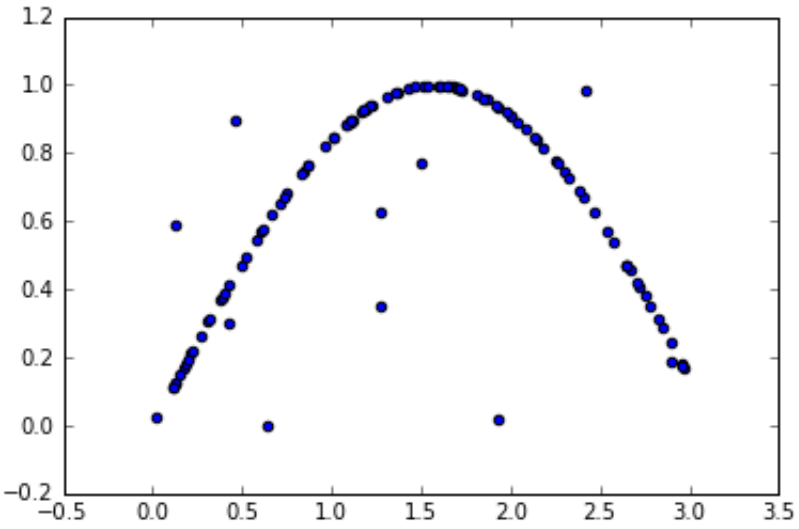

Figure 3.1: Our data contain both outliers and non-linearities. PCA can not capture non-linear relationship inside the data and also a PCA projection will be distracted by the outliers. To achieve both goals, non-linearity and robustness, we split data into two part: L_D and S. We put the L_D into an auto-encoder to capture non-linear relationship and use S to filter out the outlying observations.

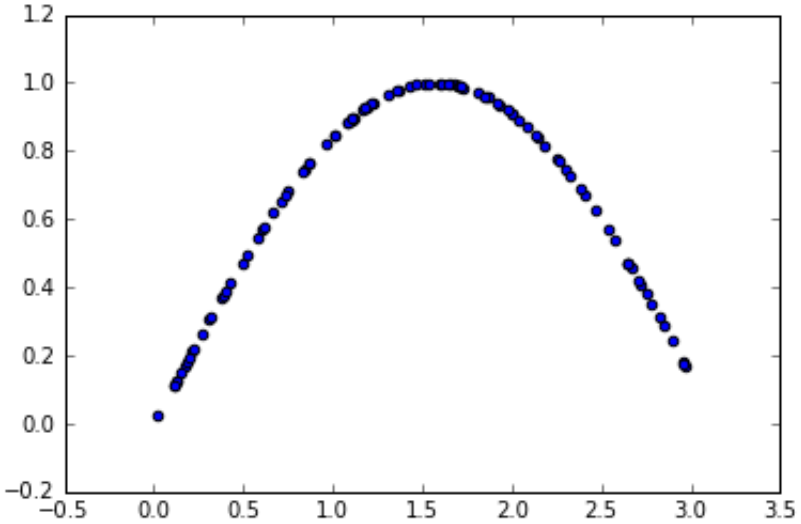

Figure 3.2: The L_D part after splitting. After splitting, the L_D part will be perfectly reconstructed by an auto-encoder. After the reconstruction, the hidden layers in the auto-encoders are a non-linear representation of input X.

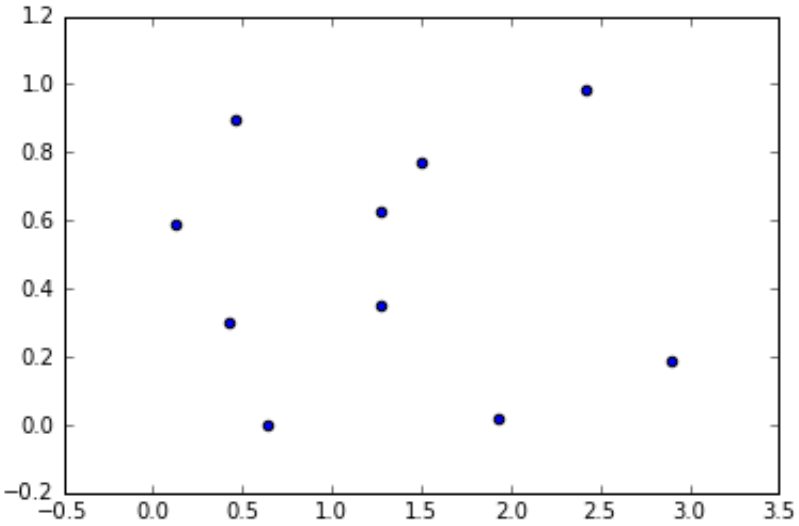

Figure 3.3: The S part after splitting. After splitting, the S part contains filtered out observations which have high reconstruction errors. The large reconstruction errors mean these observations are far away from the manifold learned by the auto-encoder. Thus, if we cast aside these outlying observations, the auto-encoder will get a lower reconstruction error. In this perspective, the S is the container of these outlying observations.

Robust auto-encoder can be extended to a deep model. In this section, we first introduce a simple shallow robust auto-encoder in detail, and then we extend the shallow model to a deep model to make predictions. Last, we discuss some relationships between our work and some other deep auto-encoders. Similar to robust PCA, we decompose our input data X into two parts $X = L_D + S$ where L_D is a matrix that can be represented by a non-linear manifold, and S are the outliers which will corrupt and skew the non-linear manifold. By peeling off the outliers from X into S, the auto-encoder could perfectly recover the remaining L_D. Our loss function for a given input X is the one norm of S and the reconstruction error of L_D in the auto-encoder, namely

$$\underset{W,b,S}{argmin} \;\; ||L_D - D_{W,b}(E_{W,b}(L_D))||_2 \;+\; \lambda||S||_1$$

$$s.t. \;\; X - L_D - S = 0$$

where $E_{W,b}(\cdot)$ denotes an encoder, $D_{W,b}(\cdot)$ denotes a decoder, $W \in R^{m \times n}$ is a projection matrix, b is the bias term, ε denotes an element-wise error, and S captures the outlying observations, L_D is a low dimension manifold and λ is a balancing parameter to tuning the power of sparsity. The constraint says that we want to split the input data X into two part L_D and S. We feed L_D as input data to a standard deep auto-encoder to learn low-dimensional representations. L_D is

input of an auto-encoder $D_{W,b}(E_{W,b}(L_D))$, we train this auto-encoder through minimizing the reconstruction error $||L_D - D_{W,b}(E_{W,b}(L_D))||_2$. The minimized reconstruction error indicates that the L_D can be projected to a low-dimensional nonlinear manifold without heavy information lose. However, some observations are far away from the non-linear manifold learned by the auto-encoder, and if we force these observations project to the manifold, the manifold needs to shift towards these outliers. This distracted manifold has a large reconstruction error for all other observations. Thus, we detach these outliers into the another part, the filter S. The filter S contains all outlying observations which have high reconstruction errors and we believe they can not be interpreted by majority observations. We require the S sparse because we want the auto-encoder to capture the trend of majority observations and filtered out "outliers", by its name, are rare. In this optimization, L_D and S are mutually influenced by the constraint $X - L_D - S = 0$. When minimizing the first term $||L_D - D_{W,b}(E_{W,b}(L_D))||_2$, we want the input of auto-encoder L_D can be perfectly reconstructed. Thus, we need to filter out more observations to S. Similarly, when minimizing the second term $||S||_1$, S will contain less and less non-zero elements. Sparsifying the outlier filter S leaves more errors to L_D and the reconstruction task of auto-encoder becomes harder. Alternatively minimizing $||L_D - D_{W,b}(E_{W,b}(L_D))||_2$ and $||S||_1$ is like a tennis game: There are two players, an auto-encoder and an outlier filter, beating the error to each other until the error small enough. The λ is the tuning parameter which balances the impact of two optimizers. After training the whole model, the S matrix is point-wise outliers, and L_D should retain the majority information of X inside the hidden layer.

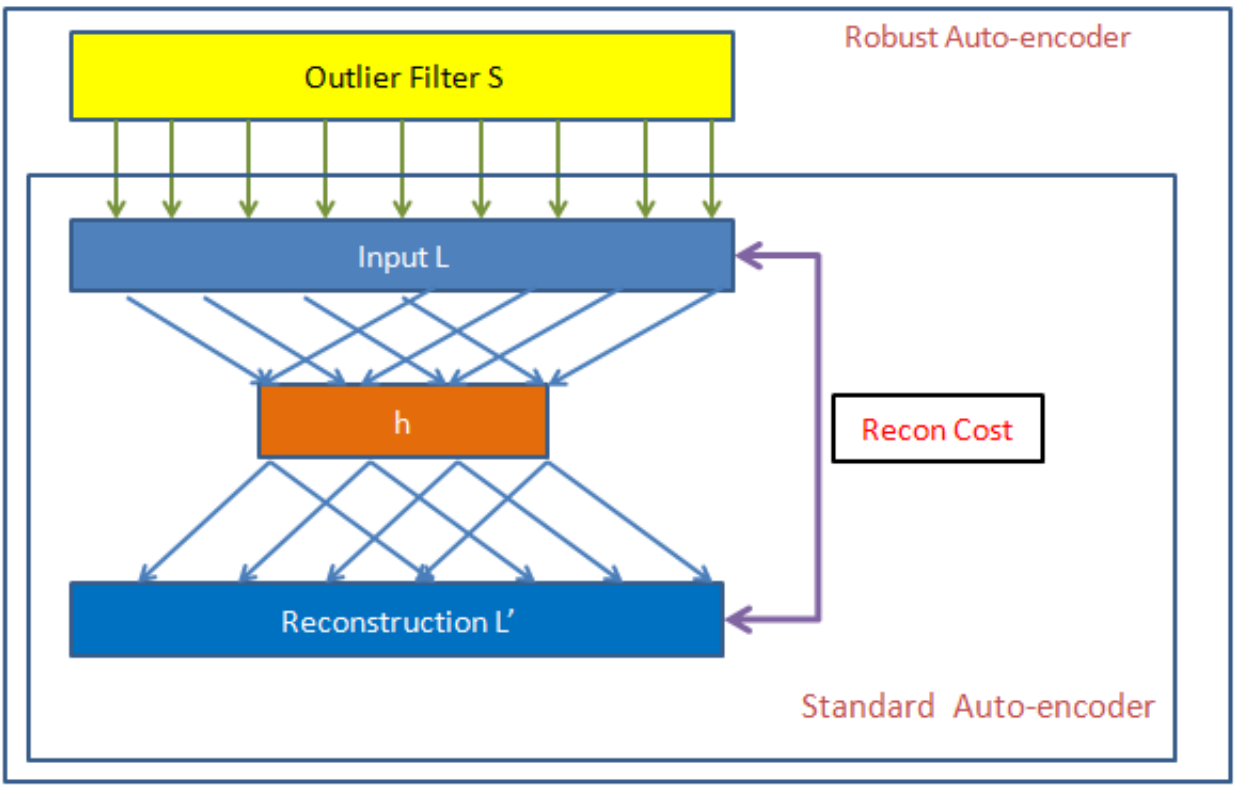

Figure 3.4: A typical structure of the robust auto-encoder. We improve the auto-encoder model by adding an outlier filter S. Thus, a robust auto-encoder consists of two parts: a standard auto-encoder and an outlier filter. After filtering some anomalies, a typical auto-encoder will get lower reconstruction error. Through a robust auto-encoder, we could obtain two kinds of representations: the lower dimensional representation which reflects the majority of data will be captured by the hidden layer h of the auto-encoder and the outlier representation captured by the filter S. The cost function of whole robust auto-encoder consists of two parts: the reconstruction error of the auto-encoder and the sparsity of the outlier filter.

3.2 Training Algorithm

We solve the minimization problem of robust auto-encoder by building a novel solver. Back-propagation [7] is typical optimization algorithm for deep learning, but it requires the objective function to be smooth to take advantage of chain rule of differentiation. This is not the case in our problem, since the second term in our object function, the $||S||_1$, is not smooth, and derivable. Our refined model is based on the idea of alternating direction method of multipliers (ADMM). We break the original objective function into two smaller pieces, each of which is then easier to handle [15]. The original target $||L_D - D_{W,b}(E_{W,b}(L_D))||_2 + \lambda||S||_1$ is broken into two parts: 1. a back-propagation algorithm to minimize the reconstruction cost of an auto-encoder $||L_D - D_{W,b}(E_{W,b}(L_D))||_2$, and 2. a shrinkage function on $||S||_1$ to sparsify S with L_D fixed. 1 and

2 work mutually to compose one iteration. Then we borrow an idea from alternating projection to force both optimizers to obey the constraint. After training the auto-encoder, the input of shrinkage function must be constrained as $S = X - reconstruction(L_D)$ and, similarly, the input of auto-encoder for next iteration must be constrained as $L_D = X - shrink(S)$. The following shows this training method:

The Main Iteration:

Input: $X \in R^{N \times n}$

Initialize the $L_D \in R^{N \times n}, S \in R^{N \times n}$ to be zero matrices and a deep auto-encoder DAE with parameter W and b.

while(iteration):

$L_D = X - S$

Train an deep auto-encoder using back-propagation. This step is to minimize the first term in objective function $||L_D - D_{W,b}(E_{W,b}(L_D))||_2$ *:*

$DAE_{W,b}$.fit(L_D)

Get reconstruction of the deep auto-encoder:

$L_D = D_{W,b}(E_{W,b}(L_D))$

$S = X - L_D$

$S = \text{shrink}(S, \lambda, \epsilon)$

Check the convergence:

1.Check the constrain $||X - D_{W,b}(E_{W,b}(L_D)) - S||_2 < \varepsilon$

c1 = $||X - L_D - S||_2$ / $||X||_2$

2.Check whether L_D *and* S *changed through iteration:*

c2 = $||LS_0 - L_D - S||_2$ / $||X||_2$

if c1 < *boundary* and c2 < *boundary* :

If converged, then break the iteration:

break

Record result of current iteration for convergence checking:

$LS_0 = L_D + S$

Return L_D and S

For example, we build an auto-encoder with an encoder $E_{W,b}(\cdot)$, a decoder $D_{W,b}(\cdot)$ parameterized with the weights $W \in R^{m \times n}$ and bias term b, and a activating function $f(\cdot)$ $R^n \Rightarrow R^n$. The first

phase of back-propagation training is to compute the reconstruction $\overline{X}$ and the reconstruction cost $||X - \overline{X}||_2$ [22]:

Input $X \in R^{N \times m}$

$h = f(W \cdot X + b)$

$\overline{X} = f(W^T \cdot X + b)$

$cost = ||X - \overline{X}||_2$

return *cost*

Then, the second phase is to backward propagate the cost error to update the weights W and bias term b [22]:

1. Compute the gradient using the chain rule of calculus

$$\delta_W = \frac{\partial\, Cost}{\partial\, h} \frac{\partial\, h}{\partial f(W)} \frac{\partial f(W)}{dW}$$

$$\delta_b = \frac{\partial\, cost}{\partial\, h} \frac{\partial\, h}{\partial f(b)} \frac{\partial f(b)}{db}$$

2. Update the weights by the gradient

$W^{l+1} = W^l - \alpha \cdot \delta_W$

$b^{l+1} = b^l - \alpha \cdot \delta_b$

To sparsify the outlier matrix, we use a shrinkage function. The basic idea of the shrinkage function is that we keep the sign for each element and subtract a small positive number λ from the absolute value of each element. Each element will evenly be shrunk towards 0, and small values will first reach 0. If an element is already close enough to 0, it will be set to 0. More elements will be shrunk to zero after several function calling. In this way, the shrinkage function promotes sparsity. The parameter λ controls the speed of sparsifying: a big λ will shrink more elements to 0 while a small λ may need more step to get satisfied sparsity. A brief description of shrinkage function:

Input: $S \in R^{m \times n}$, λ, ϵ

For i in 1 $m \times n$:

    if $S[i] > \epsilon$:

        $S[i] = S[i] - \lambda$

continue

if $S[i] < -\epsilon$:

$S[i] = S[i] + \lambda$

continue

Return S

3.2.1 Visualized Process of the Training Algorithm

We now use MNIST data set (see figure 4.1), and pick out one picture to visualize how this algorithm works.

In figure 3.5, we flatten one picture in MNIST data set (see figure 4.1). The word "flatten" means connecting every row after the previous row which will transform the 2-D image to 1-D sequence. Then we feed this 1-D sequence to an auto-encoder. After iterations of training, we get the reconstruction of the 1-D sequence. The shrinkage function takes element-wise differences between the input sequence and the reconstructed sequence and minuses a small value on the absolute value of each element. After this shrinking, the small differences will be set to zero. In this way, the S part achieves its goal, sparsity.

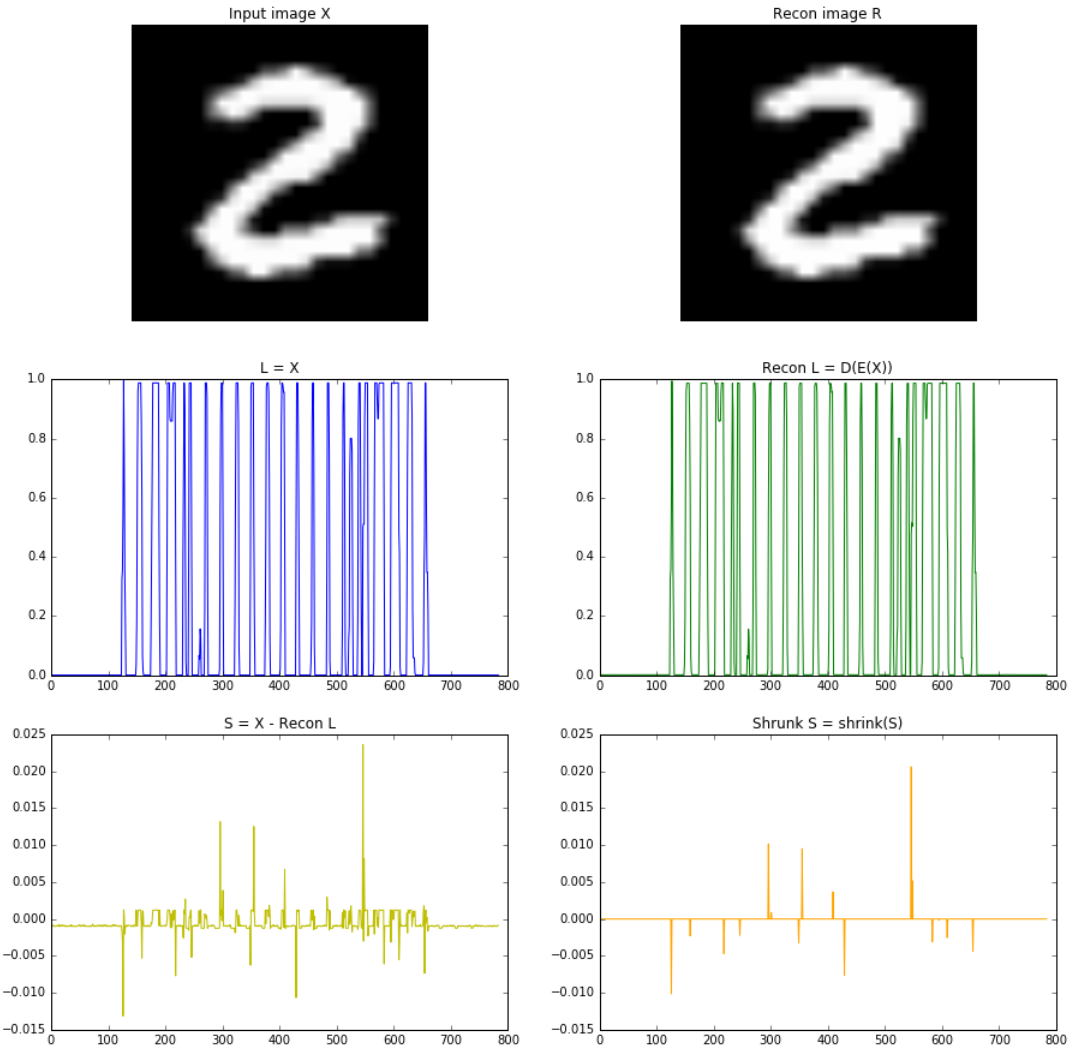

Figure 3.5: The blue line represents the input L_D of the auto-encoder part. At the beginning of the iteration, the auto-encoder part will take the full original input $L_D = X$. Then, after the auto-encoder gets well trained, we reconstruct the input L_D from the auto-encoder. The green line shows the reconstruction of L_D. However, this reconstruction can not reconstruct the full input. The difference of the input L_D and its reconstruction ReconL_D is point-wise subtracts the green line from the blue line. We get the yellow line that shows this difference. This difference shows the auto-encoder can not reconstruct everything in a given digit. We define this yellow line as the outlying parts which will be handled by the outlier container S. We apply shrink function on S to sparse S. The shrunk S is the orange line. Then, we replace $L_D = X - \text{Shrunk}(S)$ and feed auto-encoder the new L_D to start the next iteration.

Then, we narrow down to one pixel to illustrate more details of this algorithm. (Note: These pictures are just to show the process of algorithm, they do not directly demonstrate numerical

results of any data set.) We use one pixel in a picture to demo the optimization process. Since the outlier filter is an element-wise filter, one feature works identically with other features.
In figure 3.6, for a given pixel, its value is represented as the length of a segment. The auto-encoder parts L_D is represented by an ellipse since we want to use the ellipse to hint the two norm reconstruction error. The outlying parts S is represented by a square to hint we want to reach its sparsity through the optimization process.

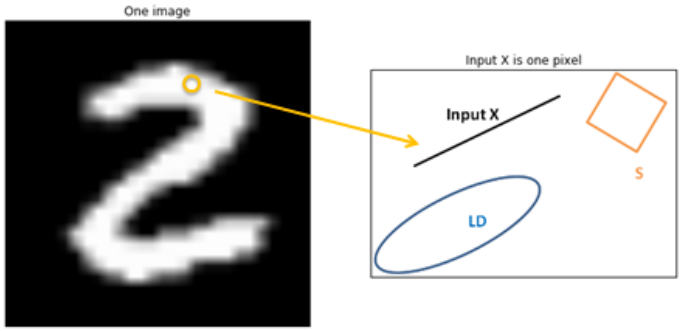

Figure 3.6: We choose one pixel from a picture. The full original input X (black) represents this pixel. The input of auto-encoder L_D is an ellipse (blue). The outlier filter S is a square (yellow).

The left figure of the figure 3.7 shows the starts of the iterations when the L_D takes all input X, and the S is initialized as 0. The right part indicates that after the auto-encoder are trained through 20 30 iterations, the reconstruction of L_D still can not capture every part of the L_D. There is a difference between the reconstruction and the input. The reconstruction of L_D represents the part of X, which is ease to reconstruct, while the difference represents the other part of X, which is hard to reconstruct.

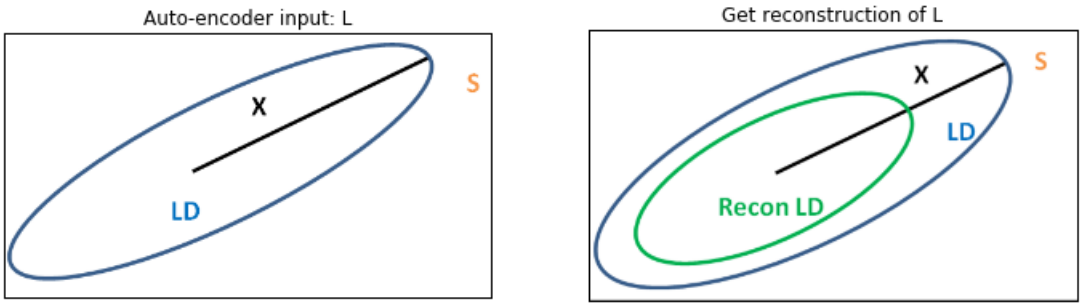

Figure 3.7: At the beginning of the iteration, we initialize the $L_D = X$ and the auto-encoder take the whole original input X. After auto-encoder is trained, we get the reconstruction "Recon L" (green) of L_D. There is a gap between the L_D and ReconL_D. It means auto-encoder can not completely reconstruct the L_D, and this gap should be tackled in another part.

Then, as shown in the figure 3.8 left, we replace the L_D by its reconstruction since we want to retain this easily recoverable part. In figure 3.8 right, the other part, hard recoverable part, is filtered by the outlying filter S.

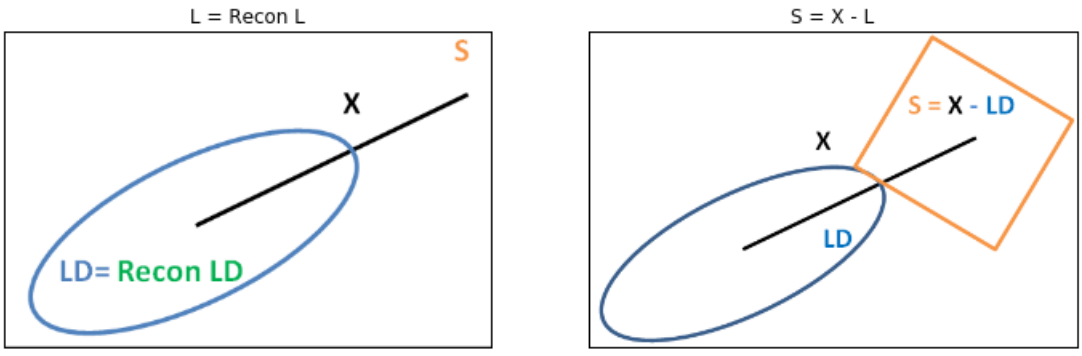

Figure 3.8: We replace the L_D by ReconL_D, and the S takes away the left part $S = X - L_D$.

The figure 3.9 left shows the effect of shrinkage function. Shrinkage function reduces the size of the S by a small step λ. We believe that the gap between the S and the shrunk S still can be captured by auto-encoder. Thus, we replace the S by the shrunk S. The gap is taken by the L_D for the next auto-encoder training (figure 3.10 left).

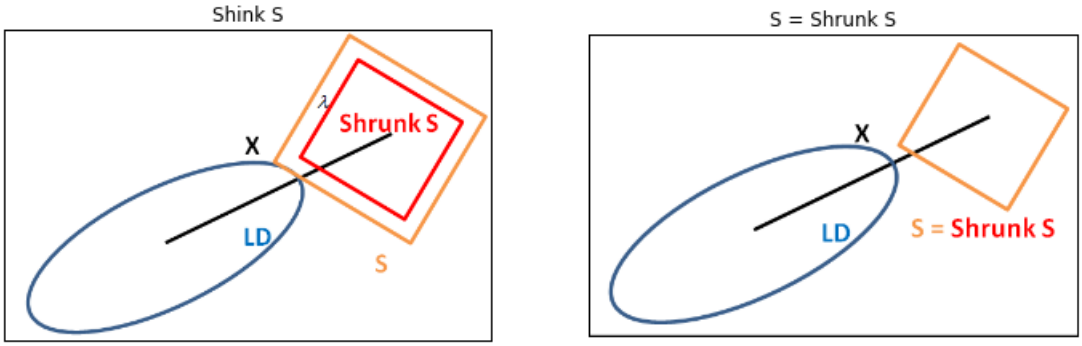

Figure 3.9: Then we apply a shrinkage function on the S, which will reduce the size of the S element-wisely with the L_D fixed, and the orange square represents the shrunk S. There is a small λ between S and shrunk S that indicates the step size of shrinking. After shrinking, the S is replaced by shrunk S. This replacement leaves another gap which will be tackled by the auto-encoder.

The 3.10 right shows the stopping condition of iterations. If the reconstruction of L_D and the L_D are almost same, then replacing L_D by its reconstruction is no difference, and there is no more gap for S to fill. The whole iterations stop. We say, at this time, the robust auto-encoder find a split position of X that the L_D part can be reconstructed by the auto-encoder and S parts contains the unrecoverable part.

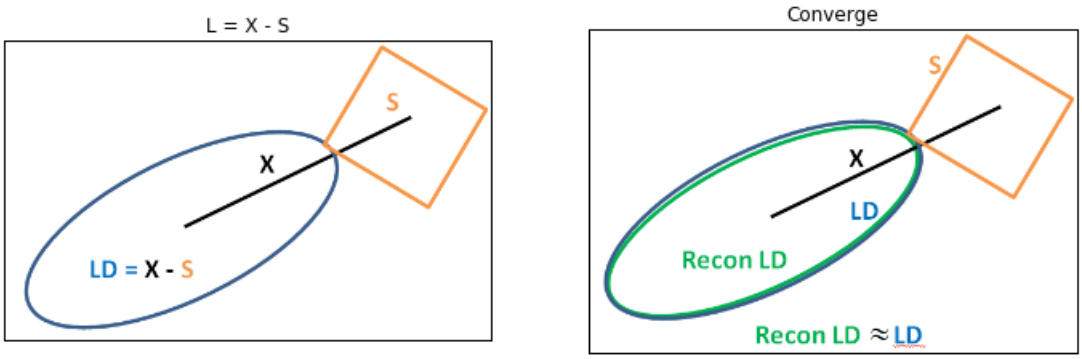

Figure 3.10: The L_D are extended to fill in the gap $L_D = X - S$. This new L_D is the input of the auto-encoder in the next iteration. Repeat the process in figure 17 figure 20, until convergence. The convergence means the reconstruction and input of the auto-encoder are approximately same Recon $L_D \approx L_D$, and then there is no gap created by reconstruction of the auto-encoder. At that time, the L_D takes easily reconstructable parts, while the left parts are hard to reconstruct, so the S filters them out. It is the target of our robust auto-encoder.

3.3 Robust Deep Auto-encoders

An intuitive way to extend a robust auto-encoder to a robust deep auto-encoder is to replace the simple auto-encoder by a deep auto-encoder. An auto-encoder with multiple hidden layers is usually called deep auto-encoder. The complexity of a deep auto-encoder will increase with more hidden layers. However, the augmented complexity is not a free lunch: an over complex auto-encoder indeed could perfectly reconstruct every observation, but it also captures noises. These noises ruin our original target, learning the informative features. Getting a balance between complexity and filtering the outliers is a design decision to make.

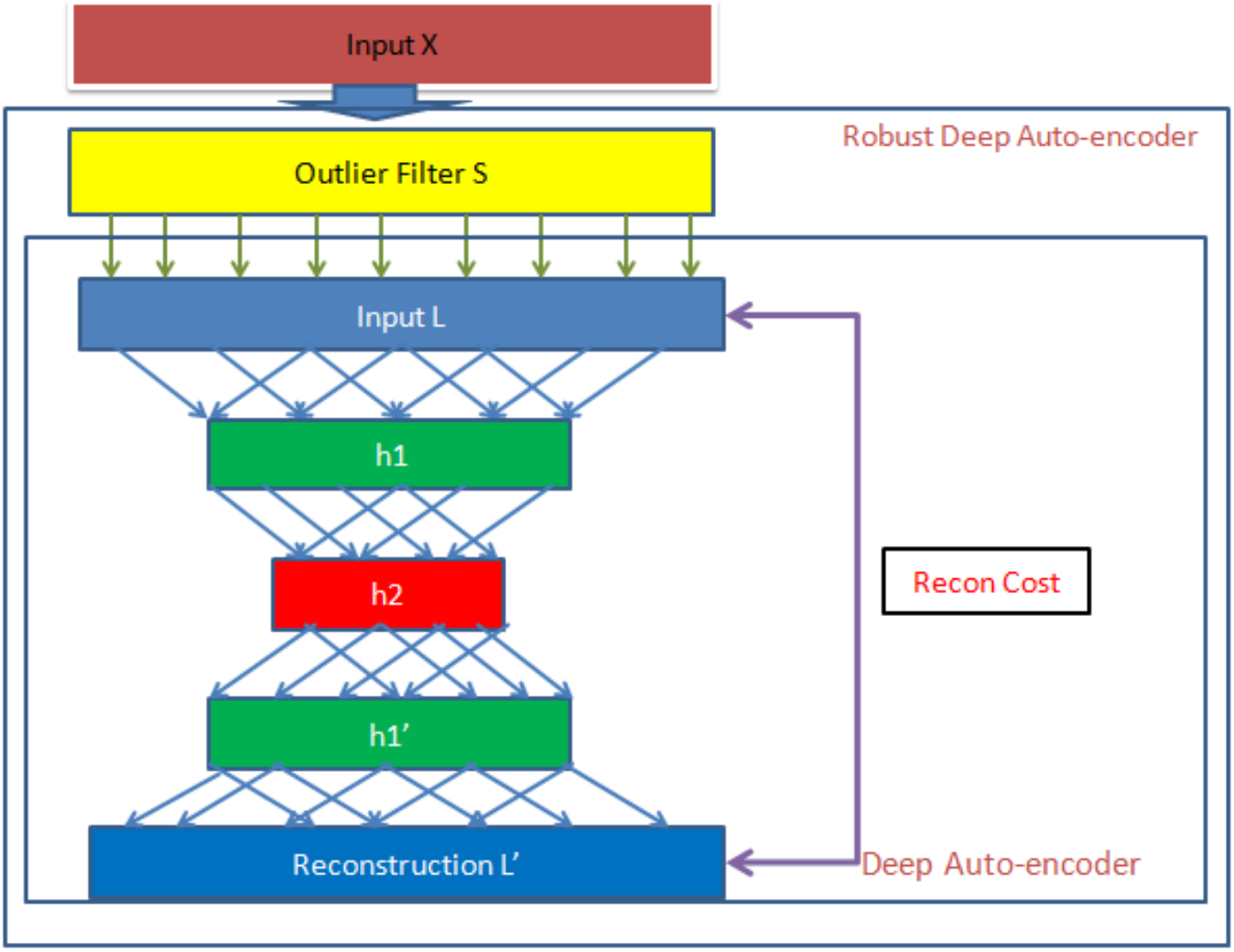

Figure 3.11: This picture shows a typical structure of a *Robust Deep Auto-encoder*. Similar to the robust auto-encoder, a robust deep auto-encoder is an improved model of normal auto-encoder by adding an outlier filter S. We feed the deep auto-encoder L_D which is residual of the raw input X after filtering. The L_D could be better reconstructed than the raw input X. Inside the deep auto-encoder, the L_D will be first encoded to a low dimension representation $h1$, then $h1$ will be continually encoded to a lower dimension representation $h2$. By these two non-linear encoder phase, $h2$ will be a complex non-linear representation of input L_D. The S still is the outlying parts which part will return high reconstruction error.

3.4 Feature Learning with Robust Auto-encoders

The two parts of robust auto-encoder, the auto-encoder and the outlier filter, have two distinct tasks to discover two different kinds of features. For the majority of the data, the auto-encoder aims to capture a low dimensional representation of the majority of the observations. While, the outlier filter contains the unusual parts which are hard to reconstruct. The low dimension representation and the outlier filter are both informative features discovered by the robust

auto-encoder. The low dimensional representation learned by the auto-encoder are compressed features that reflect the trend of the majority of the observations while the outlying parts blocked by the filter S does not necessarily mean different labels, but the abnormal always give the first clue to distinguish the difference. Also, we believe these two kinds of features can not be mixed, since if we directly apply an auto-encoder without a filter on raw input X, the auto-encoder will try to reconstruct outliers and to capture information of outliers. It will cause the low dimensional representation is not able to correctly reflect the majority of data, and also, outlying observations are no longer distinguishable. By this split, we could learn improved features, when the L_D and the S work distinctly.

3.5 Relation to Denoising Auto-encoders

The idea of introducing robustness into standard auto-encoders is not novel. It has been implemented in many different ways and shown in the literature [3, 4, 7, 12]. The denoising auto-encoder model is one such popular model [3, 7]. In this section, we briefly introduce the denoising auto-encoder and illustrate the differences between the denoising auto-encoder and our model, the robust auto-encoder. The denoising auto-encoder is a refined auto-encoder which randomly corrupts the input data and is trained to recover the uncorrupted data from the corrupted one [12]. A denoising auto-encoder with one hidden layer is described as following:

1. Randomly corrupt the input data X to $\overline{X}$
2. Encoding the corrupted data $h = E(\overline{X})$.
3. Decoding the hidden layer to get reconstruction $r = D(h)$.
4. The cost function is defined as distance between the *uncorrputed* input data and the reconstruction from the *corrupted* data $||X - r||$.

where $E(\cdot)$ denotes an encoder, $D(\cdot)$ denotes a decoder, and $||\cdot||$ can be any cost function or norm.

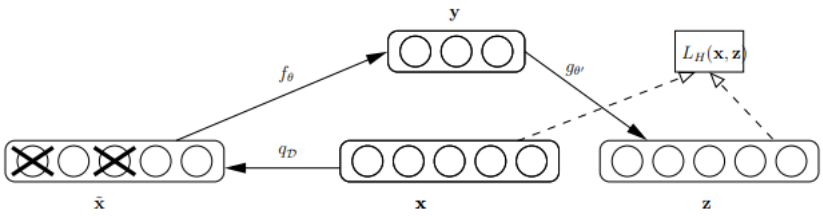

Figure 3.12: Denoising Auto-encoder. In this picture, x is the input data, and then we randomly corrupt some observations to get the corrupted $\overline{x}$. Then $\overline{x}$ is encoded by encoder f_θ, and decoded by g_θ. Our target is to minimize the cost function $C(x, f_\theta(g_\theta(\overline{x})))$ given original input x and reconstruction $f_\theta(g_\theta(\overline{x}))$ [12].

The denoising auto-encoder attends to learn the precise prediction of corrupted data from those uncorrupted data or recover that corrupted one from its neighbor. Denoising auto-encoder include robustness to outliers by introducing some random noise in the training. The minimized cost function give us evidence that we can successfully fill the corrupted data, and it means we can learn to correct the whole input just from parts of it. Both denoising auto-encoder and robust auto-encoder introduce robustness, but we think our model is more advanced since robust auto-encoder tries to distinguish the outliers from the original input while denoising auto-encoder still use the original input as learning target. If there are some outliers in the initial input, denoising auto-encoder will still recover these outliers and thus these outliers will distract features learned.

Chapter 4

Numerical Results

4.1 Data Sets

We tested our model on three different data sets. We work on MNIST data set, notMNIST data set, and packets data set. MNIST data set [25] is a famous data set for handwritten digits recognition problem. Handwritten recognition on MNIST data set is a supervised learning problem. The training set contains 50,000 pictures, and each picture is a 28×28 picture, and each picture is labeled as a human handwritten digit. Our goal is to predict the label of each picture, "which digit does each picture show?", from its raw pixels. This problem is a typical classification problem. Moreover, from some experiments, we know that human being has 2.5% error rate on this data set. Figure 4.1 shows first 400 observations in the MNIST set.

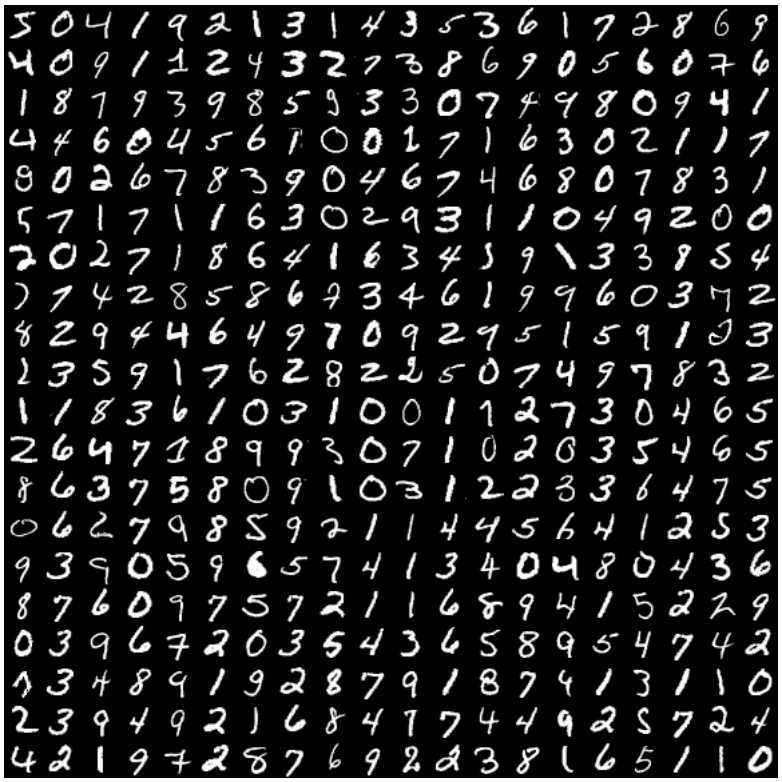

Figure 4.1: First 400 training instance in MNIST data set

The second test data set is the notMNIST data set created by Yaroslav Bulatov [26]. He took some publicly available fonts and extracted glyphs from them to make each observation has the similar dimension with the MNIST set. The notMNIST is also a classification problem. Each observation is labeled as a letter it presented within the range A-J.

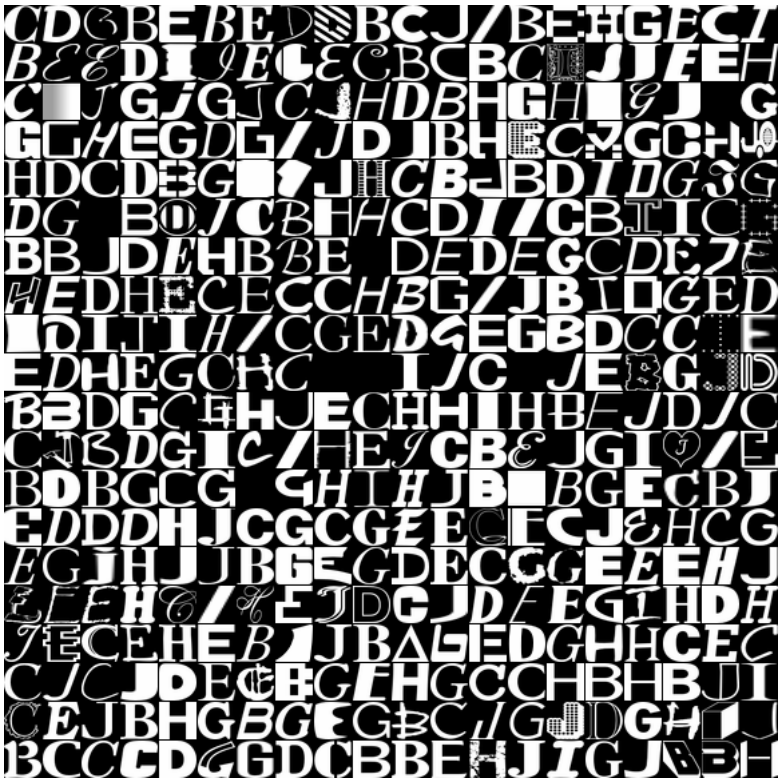

Figure 4.2: First 400 training instance in notMNIST data set

Judging by these examples, we note that this data set is a harder task than MNIST. The creator claims that he applied the standard stacked auto-encoder on notMNIST and got approximately 11% error whereas the same approach gives got 2% error rate on MNIST. This dataset consists of about 19k observations which are manually labeled. Each instance is a 28×28 small picture, that is, similar to the MNIST, each instance has 784 features.

We also apply our model to the PACKET data set. Packet data comes from computer sensor networks. It is data transforming on the internet. Each packet is a 1512 bit long sequence, and the data set contains 10060 observations. Our data is a 10060×1512 matrix. To give the first impression of this data set, we convert first 400 observations and transfer these observations from 1512 bit long sequence to a 36×42 picture. Note that each 36×42 picture does not mean each instance is a 36×42 picture. We want to demo this date set in a consistent way with the previous data sets. These packets are labeled by their source IP, destination IP, source port, and destination port.

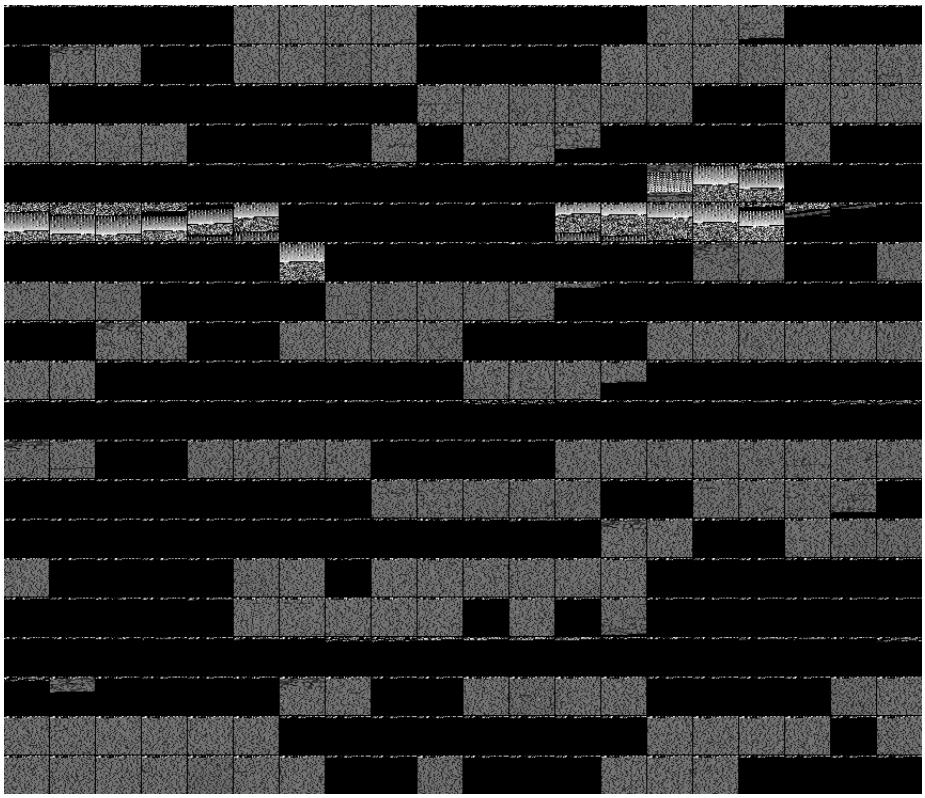

Figure 4.3: First 400 training instances in PACKET data set

From this demo, one can get a quick view of the PACKET data set. The black point shows bit "0" while the white point shows bit "1". Some packets fully contain information that while some packets only contain information at their first line, and the left part is a big black block. These "header"-only packets are activities that terminals try to communicate with each other and build connections. Like TCP connection, clients/terminals and servers which want to establish a connection must first check the availability of both side, so the "the three-way handshake" use packets only contain the "head".

4.2 Results

4.2.1 Robust Auto-encoders for Feature Discovery

We applied the robust auto-encoder model on these three data sets. In this section, we demo the low dimensional features and the outlying parts learned by the robust auto-encoder.

The figure 4.4 and 4.5 show the results of MNIST set. We use robust auto-encoder to project the observations in MNIST from 784 dimensions to 196 dimensions. The figure 4.4 shows how to non-linearly combine the original 784 features to 196 features. Each small picture is one

kind of combination and the figure 4.4 shows 196 different kinds of combinations. We use the $h = S(L_D \cdot W)$, where the $L_D \in R^{N \times m}$ is the input of auto-encoder parts, the $W \in R^{m \times n}$ is the projection matrix, and the $S(\cdot)$ is the sigmoid function. In this experiment, the $m = 784$ and the $n = 196$. Each column in W is one small picture in figure 4.4. Since n is 196, we get 196 different pictures.

The figure 4.5 shows the outlying parts for the first 100 observations. Each small picture is a filter for one observation. These filtered out parts are considered as tough parts for reconstruction.

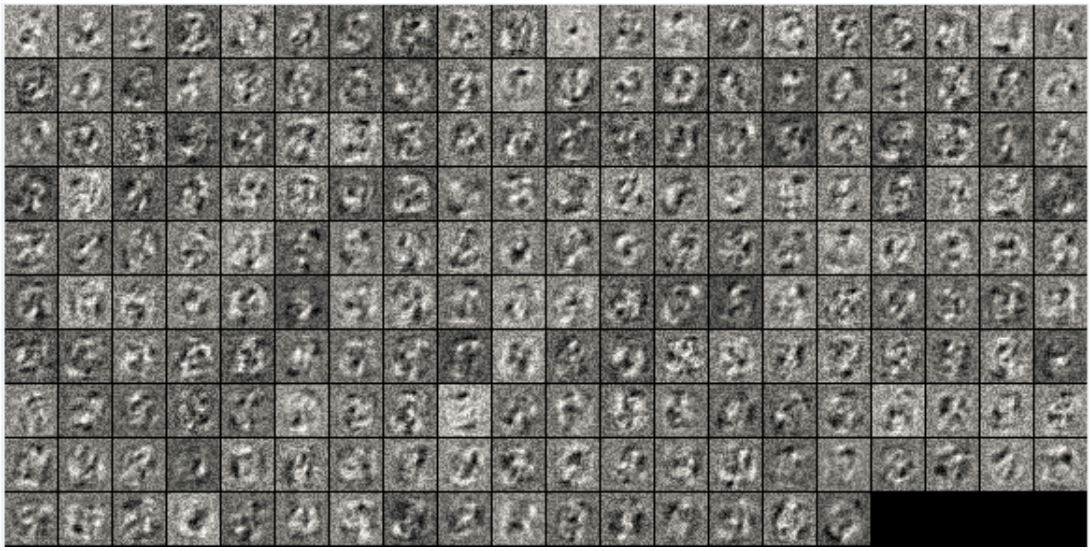

Figure 4.4: Discovering features by the robust auto-encoder on MNIST data. In this picture, there are 196 small pictures that each picture shows one way to combine original 784 features. These 196 small pictures represent 196 different ways to non-linearly combine the original features. From this figure, we note that there are clear "structures" or "blocks" in each picture, and we hold that these "structures" or "blocks" are discovered features that will contribute to improving the accuracy of predictions. Also, after such projection, all the observations in MNIST set can be reconstructed by non-linear combinations of these features.

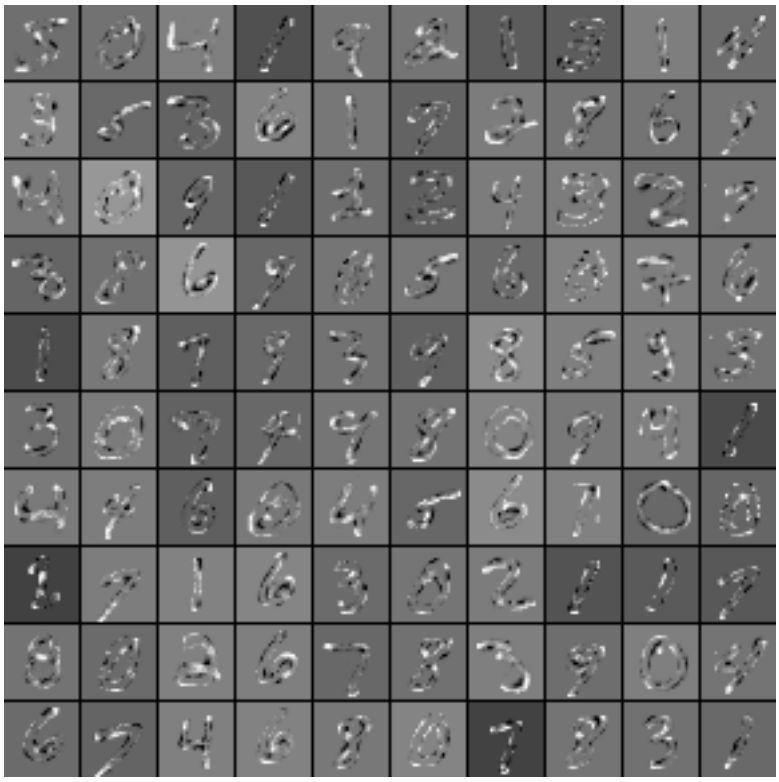

Figure 4.5: Outlying parts of first 100 observations in MNIST data. One can find that these outlying parts look like the profile of each digit. It indicates the edge parts between the digits and the black background are hard to reconstruct, so these parts are filtered out.

The figure 4.6 and 4.7 show the results of *notMNIST* data set. Similarly to MNIST experimental settings, we still use robust auto-encoder to project the observations from 784 dimensions to 196 dimensions. The figure 4.6 shows how to non-linearly combine the original 784 features to 196 features. Each small picture is one kind of combination and the figure 4.6 shows 196 different kinds of combinations.

Figure 4.6: Discovered latent features by the robust auto-encoder on notMNIST data. All the observations in notMNIST set can be reconstructed by non-linear combinations of these features.

The figure 4.7 shows the outlying parts for the first 100 observations. Each small picture is a filter for one observation. These filtered out parts are considered as tough parts for reconstruction.

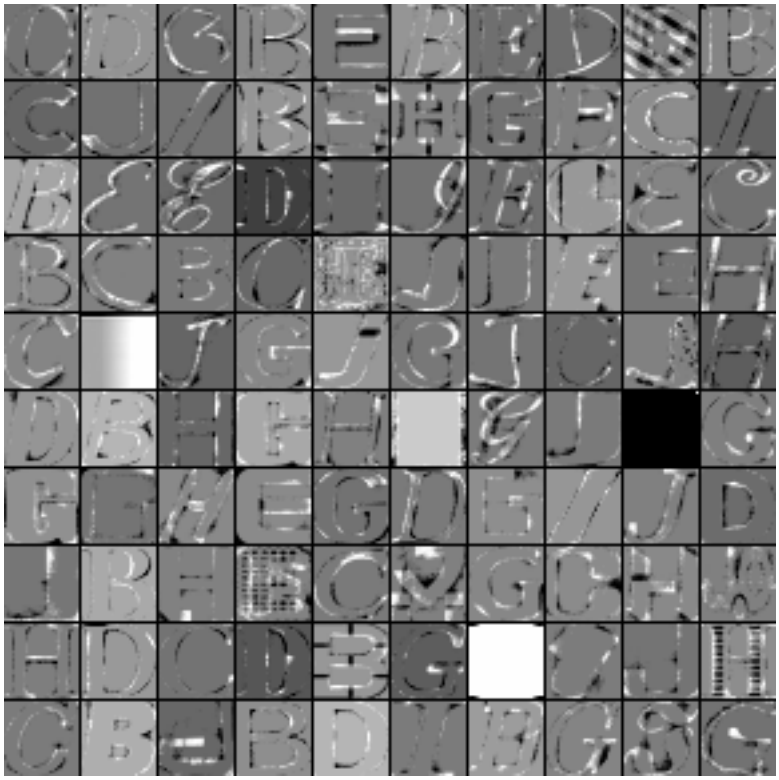

Figure 4.7: Outlying parts of first 100 observations of notMNIST data. The outlying parts look like the profile of each character. It is similar to the MNIST data set that the edges between characters and the black background are hard to reconstruct, so these parts are filtered out.

The figure 4.8 and 4.9 show the results of *Packets* data set. Unlike the previous experimental settings, the robust auto-encoder now is used to project the observations from 1512 dimensions to 784 dimensions. The figure 4.8 shows how to non-linearly combine the original 1512 features to 784 features. Each small picture is one kind of combination and the figure 4.8 shows first 400 different kinds of combinations in the 784 total combinations.

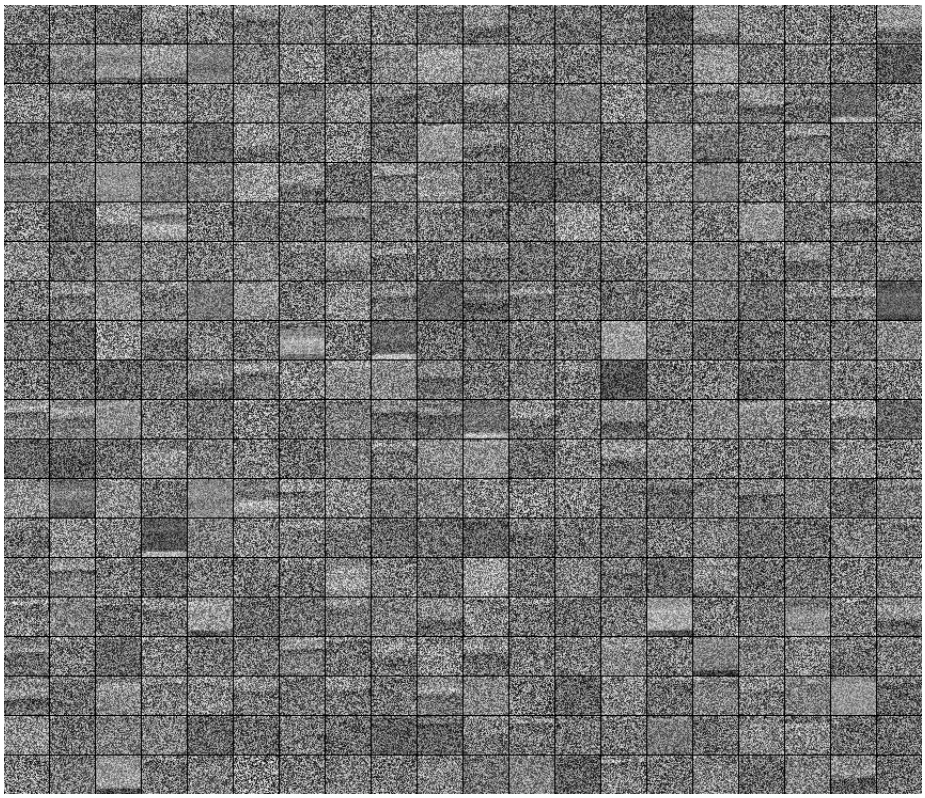

Figure 4.8: Discovered latent features by the robust auto-encoder on Packets Set. All the observations in notMNIST set can be reconstructed by non-linear combinations of these features. However, the Packets data is cyber packets transformed through a network, so these pictures do not contain the human-eye recognizable structures.

The figure 4.9 shows the outlying parts for the first 400 observations. Each small picture is a filter for every observation. These filtered out parts are considered as tough parts to reconstruct for observations. So these parts are filtered out. One can note that the cyber data contains more outlying parts due to its complexity.

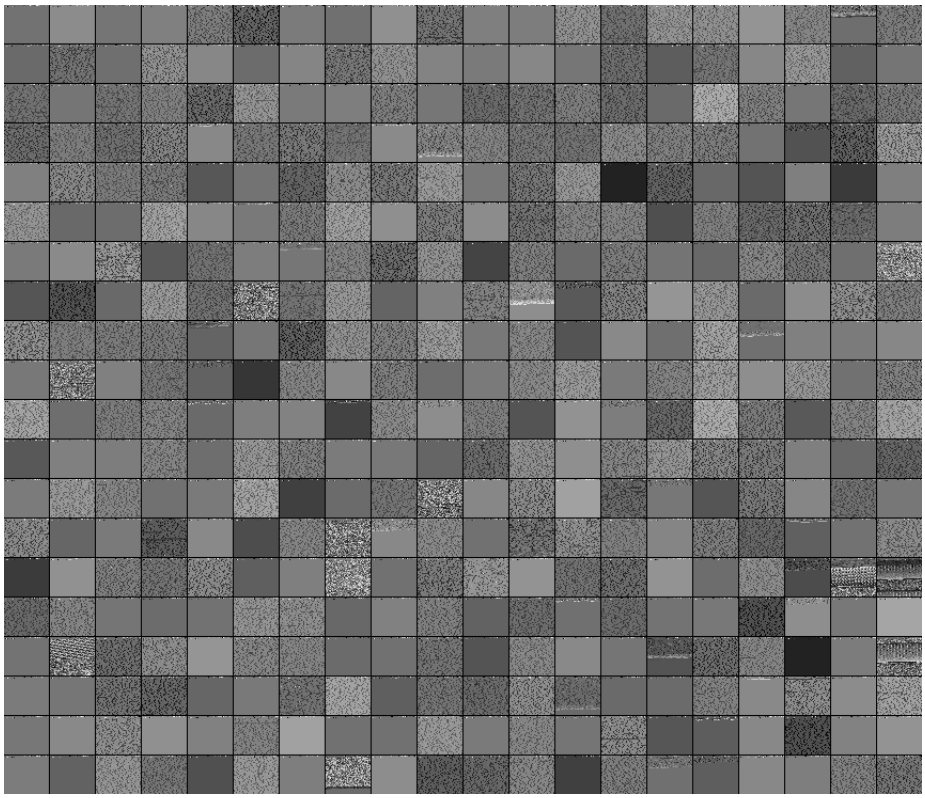

Figure 4.9: Outlying parts of first 400 observations of the Packets Set. Since the cyber data are usually considered more complex than the image data, each small picture looks containing more noise than the image data and these outlying parts are no longer the "edges".

4.2.1.1 Robust Auto-encoders for Outlier Detection

In the robust auto-encoder, the λ is a balancing parameter for the auto-encoder part and the outlier filter part. If we set the λ is an arbitrarily large number, the second part $\lambda||S||_1$ will be large. The optimizer will emphasize on minimizing this part. A large λ means we prefer the sparsity of outlier filter S. Meanwhile, by reducing λ, we gradually change optimizer interests on the first term, the auto-encoder reconstruction cost $||L_D - D_{W,b}(E_{W,b}(L_D))||_2$. The optimizer will filter out more and more outlying parts to reduce the reconstruction cost. If the λ is 0, there will be no penalty on the outlier filter S, and the optimizer will put every observation into S and left the L_D being 0. Then, there is no reconstruction cost of 0 matrix. In general, a larger λ enforces S sparser. From the figure 4.10, with the λ increasing, the outlier filter S becomes sparser.

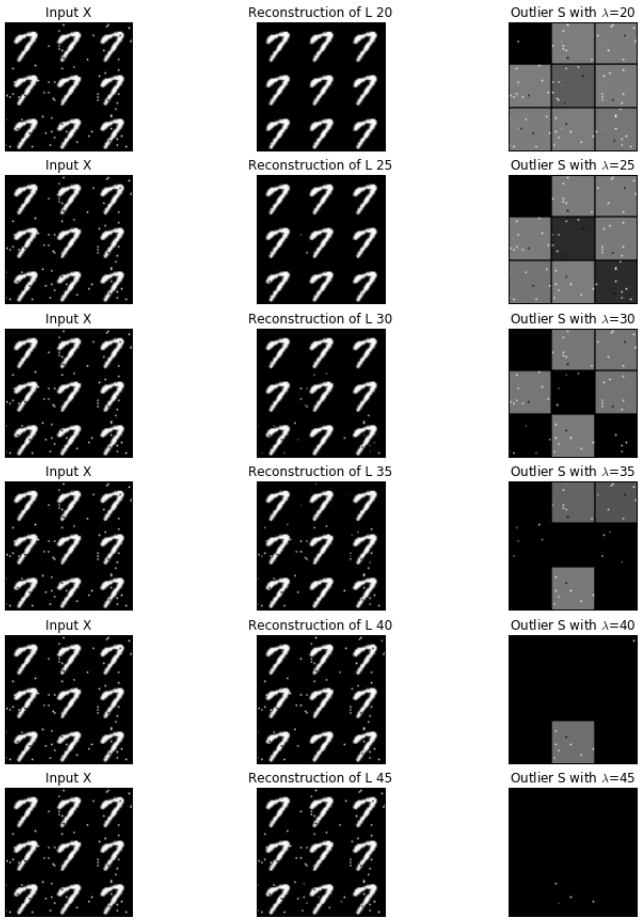

Figure 4.10: The influence of λ. In the robust auto-encoder, the λ is the tuning parameter of the outlier filter and the auto-encoder. A larger λ enforces S sparser. We tested the influence of λ on MNIST data set. In our experiment, we choose 5 different λ from 20 to 45. Each row represents one λ setting. The first column is the input X with few Gaussian random noise, and they are identical for each row. We want the robust auto-encoder to reconstruct the digit 7 and filter out those noises. The second column shows the reconstruction of robust auto-encoder. The third column shows the outlier filter S. One can find that the S becomes more and more sparse with λ increase.

4.2.1.2 Convergence of Robust Auto-encoders

After peeling off the outlying parts of data, the rest parts will be easy to reconstruct, and we will get lower reconstruction errors than the standard auto-encoders'. Also, filtering out the outlying parts of data will make the auto-encoder quickly converge. The figure 4.11 shows a comparison of reconstruction errors between a robust auto-encoder and a standard auto-

encoder. It shows that the robust auto-encoder gets lower reconstruction errors and converges more quickly than standard auto-encoder.

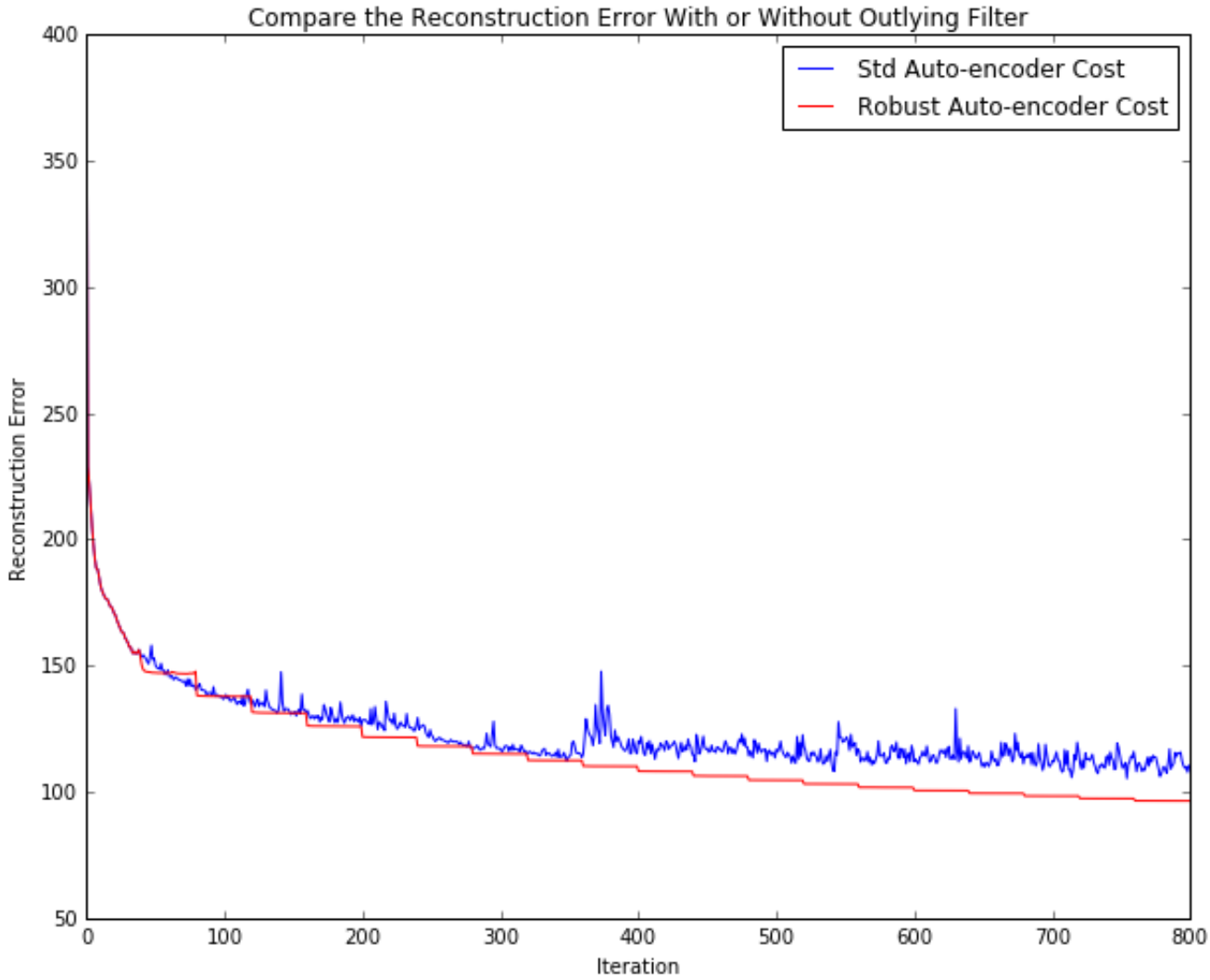

Figure 4.11: This picture shows a comparison of reconstruction costs between a robust deep auto-encoder and a normal deep auto-encoder through iterations. The x-axis is the training iteration while the y-axis is the reconstruction costs. The blue line represents the reconstruction costs of a standard deep auto-encoder while the red line represents the reconstruction costs of the deep auto-encoder inside our robust deep auto-encoder. We set same parameters for both deep auto-encoder, and the only difference is with or without an outlier filter S. We filter out outlying parts for every 40 iterations. From the picture, one can find that the red line shows a stairs shape which indicates that filtering outlying observations will reduce the reconstruction error of the deep auto-encoder. However, the blue line goes down first but still shaking after hundreds training iteration which reproduces a well-known problem: deep models are hard to train.

4.2.2 Prediction

4.2.2.1 Uncorrupted Input

Our supervised evaluation contains two steps: First, we apply several unsupervised dimension reduction methods to projection high dimensional data to low dimension and then we use random forests as our prediction model to predict the label and compare with the true label to calculate the error rates. Due to the limit of computational power, we sample 500 observations from the MNIST and the notMNIST set and 1000 observations from the packets data set. The unsupervised dimension reduction methods include the following:

1. RDAE : Robust Deep Auto-encoder with two hidden layers which projects the input data from 784 dimensions to 196 dimensions, then projects 196 dimensions to 49 dimensions and the filtering balancing parameter is $\lambda = 5.0$.
2. DAE : Deep Auto-encoder with two hidden layers which projects the input data from 784 dimensions to 196 dimensions, then projects 196 dimensions to 49 dimensions.
3. PCA : Linear projection from 784 dimensions to 49 dimensions.
4. KPCA(poly) : Kernel PCA projects from 784 dimensions to 49 dimensions using polynomial kernel function with the degree= 3.
5. KPCA(rbf) : Kernel PCA projects from 784 dimensions to 49 dimensions using radial basis kernel function with the $\gamma = 1/784$.
6. KPCA(cosine) : Kernel PCA projects from 784 dimensions to 49 dimensions using cosine kernel function.

In supervised prediction phase, the prediction model is random forest, the number of trees is 200. $\frac{2}{3}$ projected observations are used for training and $\frac{1}{3}$ observations for testing.
Table 1 shows the error rates of different data sets projected by different unsupervised methods:

Table 1 : Error Rates of *Uncorrupted* Image Data

Dataset		RDAE	DAE	PCA	KPCA(poly)	KPCA(rbf)	KPCA(cosine)
MNIST Digit	1st	13.3%	13.9%	14.5%	14.5%	13.9%	18.2%
notMNIST	1st	15.7%	18.2%	21.8%	18.8%	20.6%	18.2%

Table 2 shows the error rates of predicting different targets on packets data set:

Table 2 : Error Rates of *Uncorrupted* Cyber Data							
Target		RDAE	DAE	PCA	KPCA(poly)	KPCA(rbf)	KPCA(cosine)
Source subnet	5th	45.5%	24.8%	24.8%	25.2%	53.3%	19.4%
Destination subnet	5th	16.1%	8.5%	8.8%	10.6%	29.1%	6.7%
Source port	5th	17.3%	11.2%	10.6%	12.7%	31.5%	7.9%
Destination port	5th	42.7%	21.5%	21.8%	20.0%	55.2%	17.0%

From Table 2, we note that robust auto-encoder is not the best projection methods of getting representations. Also, the standard auto-encoders can not get low error rates either. The cosine kernel PCA in this experiment is the best projection method that finds the lowest error representations.

4.2.2.2 Corrupted Input

We randomly corrupt some features using the Gaussian noise. The corrupted samples for MNIST data, notMNIST data, and Packets data are shown in Figure 4.12 In this experimental setting, robust deep auto-encoder should always work better than the standard deep auto-encoder.

Figure 4.12: Corrupted MNIST, notMNIST, and Packet data are shown from the left to the right. According to our assumption, the Gaussian noise corruption should be very hard to reconstruct. The robust auto-encoders can eliminate these noise by outlier filter and get lower reconstruction error than standard auto-encoders.

Table 3 : Error Rates of *Corrupted* Image Data							
Dataset		RDAE	DAE	PCA	KPCA(poly)	KPCA(rbf)	KPCA(cosine)
MNIST Digit	1st	17.0%	18.8%	20.6%	20.6%	18.8%	20.6%
notMNIST	1st	16.4%	17.6%	20.6%	20.6%	21.8%	19.4%

Table 4 : Error Rates of *Corrupted* Cyber Data							
Target		RDAE	DAE	PCA	KPCA(poly)	KPCA(rbf)	KPCA(cosine)
Source subnet	4th	32.4%	34.5%	20.6%	22.7%	52.1%	15.2%
Destination subnet	4th	20.9%	35.5%	8.2%	9.4%	31.5%	7.9%
Source port	4th	21.2%	36.7%	10.3%	12.1%	32.7%	8.5%
Destination port	4th	33.3%	35.2%	23.0%	24.2%	53.6%	17.0%

Table 3 shows comparison results on corrupted data while other experimental settings are identical with the previous experiment. On corrupted data, the *Robust Auto-encoder* still performs better than methods. This result corresponds to our expectation. From Table 4, still, robust auto-encoder and standard auto-encoder are not desired projection methods of getting representations. The cosine kernel PCA in this experiment is the best projection method that finds the lowest error representations.

4.2.2.3 Significance Test

We observed some improvement of robust auto-encoder comparing with the standard auto-encoder. We also tested whether this improvement is statistically significant when we take the size of the test data into consideration. This test is to check whether two groups of binomial trials are statistically significant. T_1 and T_2 are two groups of binomial trials and p_1 and p_2 are the success rate of T_1 and T_2. Then we build null hypothesis versus alternative hypothesis as following:

$$H_0 \ : \ p_1 = p_2$$
$$H_A \ : \ p_1 \neq p_2$$

Then we calculate the test statistic z as:

$$z = \frac{p_1 - p_2}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{n_1} + \frac{1}{n_2})}}$$

where $\hat{p} = \dfrac{n_1 p_1 + n_2 p_2}{n_1 + n_2}$. The p-value is $1 - cdf(z)$, where the $cdf(\cdot)$ is the cumulative distribution function (CDF) of the normal distribution when n_1 and n_2 are large. If the p-value < 0.05, we reject the null hypothesis and accept the alternative hypothesis that p_1 and p_2 are statistically different.

Table 5 : Significant Test for Different Sets				
Data Sets	RDAE Correctness Rate	DAE Correctness Rate	Test Size	p-value
Corrupted MNIST	86.7%	80.6%	165	0.07
Uncorrupted MNIST	71.5%	69.7%	165	0.36
Corrupted notMNIST	84.8%	81.2%	165	0.19
Uncorrupted notMNIST	87.9%	84.8%	165	0.21

From Table 5, we note that our model gets relatively better p-value on the corrupted MNIST data set that the improvement of our model is good while other p-values do not show desired results.

Chapter 5

Conclusion

5.1 Contribution

During this thesis research, we explored some auto-encoder models and improved the basic auto-encoder by adding a sparse outlier filter. We demonstrated that after the outlier filtering, the auto-encoders will get lower reconstruction errors. Our model is a novel combination of auto-encoders and robust PCA and inherits both advantages. It can non-linearly project high dimensional data to lower dimension while peels off the outlying parts of the data. We also proposed a new training algorithm for this new model. We separately train the auto-encoder using back-propagation and the outlier filter using shrinkage function. We train one part with the other part fixed. The constraints are checked following the each part of the optimization. This new training algorithm is a mixture of the back-propagation and the ADMM. Also, we learned the idea of the Dykstra's alternating projection method to compose our constraint checking process. In our experiments, the robust auto-encoder succeeds in filtering out random noises. Further, its reconstruction errors are lower than the standard auto-encoders' as our expectation. For the supervised prediction part, the robust auto-encoder gets some improvements compared with the standard auto-encoders on two image data sets but still need further studies on cyber data analysis.

5.2 Future Work

We propose three directions of the future work:

First, the way we extend our model to deep is directly adding a filter on the top of the standard

auto-encoder, but building filters inside the deep auto-encoder is a novel idea.

Second, the training algorithm we proposed for our model has not been mathematically proved. The experimental results show that this training algorithm works as consistently as we expect it, but mathematical proof is a worthy direction to continue.

Third, our model is not ideal in the cyber domain, and the standard deep auto-encoder does not perform well either. Applying some other kinds of auto-encoders like stacked auto-encoder is our next step.

Bibliography

[1] Marcus A. Maloof "Machine Learning and Data Mining for Computer Security: Methods and Applications" *Advanced Information and Knowledge Processing ISSN 1610-3947 ISBN-10: 1-84628-029-X*

[2] Banerjee, Sudipto; Roy, Anindya (2014), *Linear Algebra and Matrix Analysis for Statistics, Texts in Statistical Science (1st ed.)*, Chapman and Hall/CRC, ISBN 978-1420095388

[3] Deng, Li, and Dong Yu. "Deep learning: methods and applications." *Foundations and Trends in Signal Processing 7.34 (2014): 197-387.*

[4] Bengio, Yoshua. "Learning deep architectures for AI." Foundations and trends in Machine Learning 2.1 (2009): 1-127.

[5] Paffenroth, Randy, et al. "Space-time signal processing for distributed pattern detection in sensor networks." *Selected Topics in Signal Processing, IEEE Journal of 7.1 (2013): 38-49.*

[6] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends in Machine Learning 2.1 (2009): 1-127.*

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning" *Goodfellow-et-al-2016-Book, Book in preparation for MIT Press, 2016*, http://goodfeli.github.io/dlbook/

[8] J. Wright; Y. Peng, Y. Ma, A. Ganesh, S. Rao (2009). "Robust Principal Component Analysis: Exact Recovery of Corrupted Low-Rank Matrices by Convex Optimization". *Neural Information Processing Systems, NIPS 2009.*

[9] Emmanuel J. Candes; Xiaodong Li; Yi Ma; John Wright. "Robust Principal Component Analysis?".

[10] Ringberg, Haakon, et al. "Sensitivity of PCA for traffic anomaly detection." *ACM SIGMETRICS Performance Evaluation Review. Vol. 35. No. 1. ACM, 2007.*

[11] Gehring, Jonas, et al. "Extracting deep bottleneck features using stacked auto-encoders." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[12] Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[13] Vincent, Pascal, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." The Journal of Machine Learning Research 11 (2010): 3371-3408.

[14] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." Neural computation 18.7 (2006): 1527-1554.

[15] Boyd, Stephen, et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers." Foundations and Trends in Machine Learning 3.1 (2011): 1-122.

[16] Boyd, Stephen, and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.

[17] Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." Advances in neural information processing systems 19 (2007): 153.

[18] Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *Proceedings of the 25th international conference on Machine learning. ACM, 2008.*

[19] Hastie, Trevor, et al. "The elements of statistical learning: data mining, inference and prediction." *The Mathematical Intelligencer 27.2 (2005): 83-85.*

[20] Ringberg, Haakon, et al. "Sensitivity of PCA for traffic anomaly detection." *ACM SIGMETRICS Performance Evaluation Review. Vol. 35. No. 1. ACM, 2007.*

[21] Wikipedia. Backpropagation URL *https://en.wikipedia.org/wiki/Backpropagation*

[22] Stanford Online Course URL *http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm*

[23] Picture resource *https://www.datarobot.com/blog/a-primer-on-deep-learning/*

[24] Picture resource *http://www.slideshare.net/david.kh/promises-of-deep-learning*

[25] Picture resource *http://yann.lecun.com/exdb/mnist/*

[26] Picture resource *http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html*

[27] Picture resource *http://stats.stackexchange.com/questions/114385/what-is-the-difference-between-convolutional-neural-networks-restricted-boltzma*

[28] Picture resource *https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks*

[29] Boyle, James P., and Richard L. Dykstra. "A method for finding projections onto the intersection of convex sets in Hilbert spaces." Advances in order restricted statistical inference. Springer New York, 1986. 28-47.

[30] Wikipedia. Matrix rank- Wikipedia, the free encyclopedia, 2015. URL *https://en.wikipedia.orgwikiRank_(linear_algebra)*

[31] Wikipedia. Dykstra projection algorithm URL *https://en.wikipedia.org/wiki/Dykstra%27s_projection_algorithm*

[32] Wikipedia. Projections onto convex sets URL *https://en.wikipedia.org/wiki/Projections_onto_convex_sets*

[33] Deep learning tutorial. URL *http://www.jtoy.net/2016/02/14/opening-up-deep-learning-for-everyone.html*

[34] Packet processing software *https://www.snort.org/*

[35] Cross Validate Question: Test two binomial distributions are statistically different *http://stats.stackexchange.com/questions/113602/test-if-two-binomial-distributions-are-statistically-different-from-each-other*