Domain Adaptation By Preserving Topology

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Debajyoti Bhowmik

[Roll No: CS-1602]

under the guidance of

Prof. Nikhil R. Pal
Professor
Electronics and Communication Sciences Unit

Indian Statistical Institute
Kolkata-700108, India

July 2018

Indian Statistical Institute , Kolkata

Kolkata 700108

M.Tech(CS) DISSERTATION THESIS COMPLETION CERTIFICATE

This is to certify that the Dissertation entitled **"Domain Adaptation by preserving topology",** submitted by **Debajyoti Bhowmik** is a record of bonafide work carried out by him, in the partial fulfilment of the requirement for the award of degree of Master of Technology (Computer Science) at the Indian Statistical Institute, Kolkata. This work is done during the year 2017-2018, under my guidance.

_____

(Prof. Nikhil R. Pal)

Supervisor

Acknowledgements

I would like to thank my supervisor Prof. Nikhil R. Pal for introducing me to this research problem and for his continued guidance and support throughout the year.

Abstract

Domain adaptation is highly researched area among machine learning experts. In domain adaptation we use a domain with enough class label (source domain) and try to predict class label of a different data which doesn't have any class label (target domain) . Both source and target domain share the same features space. Many approaches are there but one popular approach is to reduce the distance between source and target domain data distributions. There are many algorithm that tries to project the source and target data into a latent space so that the distance between data distribution of source and target domain reduces. But when we try to project data into a latent space the shape of data may change . Most domain adaptation algorithm does not try to preserve the shape of data explicitly. Projection of data into a latent space may change the shape of data or precisely the topology of the data. If we can preserve the topology of the data when we are projecting the data into a latent space then we may achieve better accuracy . In this thesis we have developed a method so that we can preserve the topology of the data at the time of projecting it into a latent space . For projection in to a latent space we have used auto-encoder that create encoded version of data at hidden layers. Hidden layer representation of an auto-encoder is a projection of data into a latent space . The distance between data distribution of hidden layer representation of source and target data reduces . In an auto-encoder ,the shape of the data may not be preserved i.e auto-encoder is not forced to preserve any topological property of the data. In this thesis we have added a extra constraint in a auto-encoder so that auto-encoder can preserve topological property of the data . Preserving topology of data can enhance the ability of the classifier that is trained on source data to correctly classify target data .

Contents

1 Introduction **5**

2 Preliminary Knowledge: **7**
2.1 Auto-encoder: 8

3 Autoencoder in domain adaptation: **12**

4 Domain Adaptation preserving topology: **14**

5 Preserving topology in autoencoder **16**
5.1 Adding topology constraint in autoencoder: 16
5.2 Outline of Learning Procedure for modified auto-encoder: 21
5.3 Neighbourhood selection for a point: 23
5.4 Algorithm for creating a batch: 24
5.5 Procedure for training: 24

6 Experimental Result: **26**

Chapter 1

Introduction

Many machine learning methods work well only under a common assumption: the training and test data are drawn from the same feature space and the same distribution. When the distribution changes, most statistical models need to be rebuilt from scratch using newly collected training data. In many real world applications, it is expensive or impossible to recollect the needed training data and rebuild the models. In such cases, knowledge transfer or transfer learning between task domains would be desirable. In a normal machine learning method people assume that the distributions of the labelled and unlabelled data are the same . Transfer learning, in contrast, allows the domains, tasks, and distributions used in training and testing to be different. In the real world, we observe many examples of transfer learning . For example, we may find that learning to recognize orange might help to recognize mousambi. The study of transfer learning is motivated by the fact that people can intelligently apply knowledge learned previously to solve new problems faster or with better solutions.[1]
Transfer learning has different sub category. Here we work in the field of domain adaptation . Domain adaptation is needed when we have no labelled data in the target domain but a lot of labelled data in source domain and feature space for both domains is same with different marginal probability distribution . There are different kind

of algorithm for domain adaptation. Generally we can categorise those algorithm into three distinct category.

Three algorithmic principles for domain adaptation:

Re-weighting algorithms: The objective is to re-weight the source labelled sample such that it "looks like" the target sample (in term of the error measure considered)[2][3].

Iterative algorithms: A method for adapting consists in iteratively "auto-labelling the target examples. The principle is simple:

1. a model h is learned from the labelled examples
2. h automatically labels some target examples
3. a new model is learned from the new labelled examples

Note that there exists other iterative approaches, but they usually need some labelled examples for the target domain[4].

Search of a common representation space: The goal is to find or construct a common representation space for the two domains. The objective is to obtain a space in which the domains are close to each other while keeping good performances on the source labelling task. This can be achieved through the use of Adversarial machine learning techniques where feature representations from samples in different domains are encouraged to be indistinguishable [5][6].

In this thesis we will discuss a algorithm related to searching of common representation space . For creating a common representation space we have used modified auto-encoder . We will discuss all the necessary details one by one.

In chapter 2 and 3 we will discuss some preliminary knowledge and auto-encoder. In chapter 4 we will discuss why topology matters in case of domain adaptation. In chapter 5 we will show how topological factor has been added to auto-encoder. And in last chapter i.e chapter 6 we will show our experimental results.

Chapter 2

Preliminary Knowledge:

Before starting the discussion we have to explain a few definitions which are important in the context of domain adaptation.

Marginal Probability Distribution:

Marginal distribution functions of $X$and Y mean individual distribution functions $F_x(X)$ and $F_y(Y)$ of X and Y respectively.Given, joint distribution function of X and Y: $F_{xy}(X,Y)$. Then $F_x(X) = P(X \leq x) = P(X \leq x, Y < \infty)$.$F_x(X)$ is called marginal probability distribution of X. Similarly we can derive marginal probability distribution for Y.

Domain and tasks:

A *domain* D consists of two components: a feature space χ and a marginal probability distribution $P(X)$,where $X = \{\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_n}\} \in \chi$.For example,if our learning task is document classification, and each term is taken as a binary feature, then χ is the space of all term vectors,$\mathbf{x_i}$ is the i th term vector corresponding to some documents, and X is a particular learning sample. In general, if two domains are different, then they may have different feature spaces or different marginal probability distributions.

Given a specific domain, $D = \{\chi, P(X)\}$,a *task* consists of two components: a label space Y and an objective predictive function $f(.)$,which is not observed but can be learned from the training data,which consist of pairs $\{\mathbf{x_i}, y_i\}$, where $\mathbf{x_i} \in X$ and $y_i \in Y$.The

function $f(.)$ can be used to predict the corresponding label,of a new instance x.

Domain adaptation is a process of transferring knowledge when feature space of source and target domain are same as well as the source and target domain *task* but marginal probability distributions are different.

Before discussing our method for domain adaptation we are going to discuss a simple auto-encoder (used in our method) in this chapter.

2.1 Auto-encoder:

An auto-encoder is a simple neural network model that has same number of nodes in the output layer as in the input layer and it has one or more hidden layers between input and output layers. It tries to reconstruct the input patterns at the output layer[7].

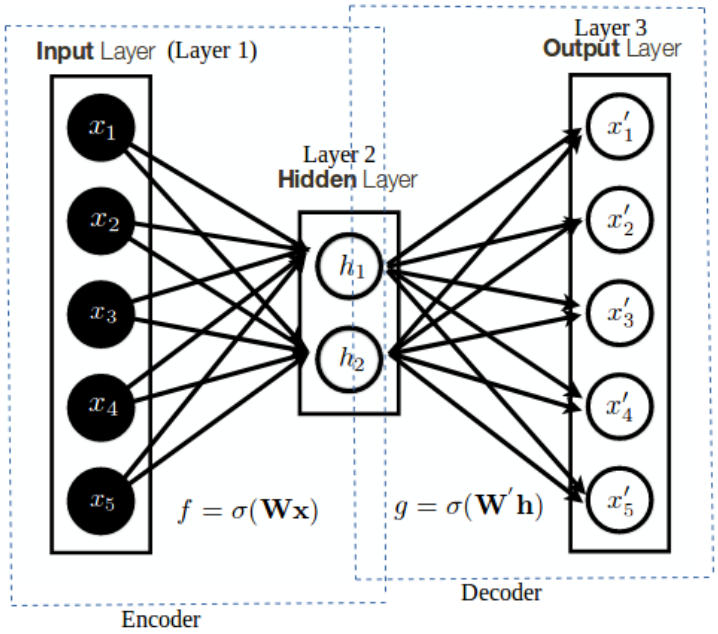


Figure : 1

The auto-encoder in Figure 1 has 5 nodes in input layer , 2 nodes in hidden layer and 5 nodes in output layer. $X^{'}$ is output and X is input; $X^{'} = f(g(X))$.
W is the weight matrix between the input and the hidden layer and $W^{'}$ is the weight matrix between the input and the hidden layer.
$L(X, X^{'})$ is the loss function where is X the input and $X^{'}$ is the output.$L(X, X^{'})$ is called reconstruction loss. Here f is encoder function and is g decoder function.

Encoder part is from input layer to hidden layer and decoder part is from hidden layer to output layer. So first input data are encoded into a hidden form and then from that hidden form original input data are reconstructed.

Generally for the reconstruction loss we use mean squared error. Now we have to find how an auto-encoder is trained. Training of the auto-encoder is same as normal multilayer perceptron having more than one hidden layer. To find optimum weight matrix between layers gradient descent method is used to minimise the loss.

Above example contains only one hidden layer i.e only one encoder and decoder function . We can again use the hidden layer representation and try to reconstruct it using another layer. This layer wise approach is called stacked auto-encoder. It is same as normal auto-encoder but it contains multiple layer of encoding and decoding.
Let us discuss how we train a simple multilayer perceptron with one hidden layer. We are using mean squared error as a loss function.

First we have to define some notation that we will use through out the entire thesis.
We are using u, v for indexing data in a batch because it is convenient for calculating gradient when topology constraint is used.(we will discuss it in chapter 5)
N=number of nodes in input layer
p=number of nodes in output layer (p=N for auto-encoder)
m=number of nodes in hidden layer
n = number of data points in a batch(we are considering batch

learning)
$Y_k(u)$ = value of $k-$th dimension of given input data point in a batch indexed by u
$y_k^3(u)$ = output of unit k at third layer(output layer) for a given input data point in a batch indexed by u
$y_j^2(u)$ =output of jth node at layer 2 for a input data point indexed by u
$y_i^1(u)$ =output of j-th node at layer 1 for a input data point indexed by u
W_{ij}^{12} =weight between i th node of layer 1 and j th node of layer 2
W_{jk}^{23} =weight between j th node of layer 2 and k th node of layer 3
$h_{k,.}^2$ =net input at node k of layer 2, . represents any data point
$h_{k,.}^3$ =net input at node k of layer 3, . represents any data point
$g^{'}(h_{j,u}^2)$ is the derivative of output of j-th node at layer 2 w.r.t net input $h_{j,u}^2$ for data indexed by u.
$g^{'}(h_{k,u}^3)$ is the derivative of output of unit k in layer 3 with respect to the net input $h_{k,u}^3$ for data indexed by u.

We are considering batch learning for auto-encoder.
E_u is mean square error for each input data point indexed by u in a batch

$$E_u = \frac{1}{2}\sum_{k=1}^{p}\left(y_k^3(u) - Y_k(u)\right)^2$$

The mean square error E(i.e $E_{reconstruction}$) is computed over the entire batch.

$$E = \frac{1}{n}\sum_{u=1}^{n} E_u$$

$$E_{reconstruction} = E = \frac{1}{n}(\sum_{u=1}^{n}(\frac{1}{2}\sum_{k=1}^{p}\left(y_k^3(u) - Y_k(u)\right)^2)$$

We will consider one single batch to train our simple multi-layer perceptron. First we have to present each data point of that batch

to the network and then calculate the error E. Then we have to calculate the gradient of the error w.r.t weights.

$$\frac{\partial E}{\partial W_{jk}^{23}} = \frac{1}{n}\sum_{u=1}^{n}\frac{\partial E_u}{\partial W_{jk}^{23}}$$

$$where\ \frac{\partial E_u}{\partial W_{jk}^{23}} = -(Y_k(u) - y_k^3(u))g^{'}(h_{k,u}^3)y_j^2(u)$$

$y_j^2(u)$ is the output of jth node at layer 2 for data point indexed by u.

$$\frac{\partial E}{\partial W_{ij}^{12}} = \frac{1}{n}\sum_{u=1}^{n}\frac{\partial E_u}{\partial W_{ij}^{12}}$$

$$where\ \frac{\partial E_u}{\partial W_{ij}^{12}} = -(\sum_{k=1}^{p}\left\{(Y_k(u) - y_k^3(u))g^{'}(h_{k,u}^3)W_{jk}^{23}\right\})g^{'}(h_{j,u}^2)y_i^1(u)$$

Since, we have calculated the gradient we can update the weights now. If η is the learning rate then we can update the weights as follows:

$$W_{jk}^{23} = W_{jk}^{23} + \eta(-\frac{\partial E}{\partial W_{jk}^{23}})$$

$$W_{ij}^{12} = W_{ij}^{12} + \eta(-\frac{\partial E}{\partial W_{ij}^{12}})$$

Chapter 3

Autoencoder in domain adaptation:

Auto-encoder is generally used to create compressed form of given data. In general simple auto-encoder contains less number of nodes in the hidden layer compared to that in the input layer. We can use more number of nodes in hidden layer but that may not be useful to get any encoded version of the input data. Our goal is to create an encoded version of input data with lesser dimension. The encoder function learns some useful characteristics of the input data and it compresses the input data into a latent representation and decoder reconstructs the original input from the latent representation constructed at the hidden layer .

In domain adaptation we have labelled source data and unlabelled target data . Marginal probability distributions of source and target data are usually different. If we can some how match the data distribution of source and target data then we can classify target data by a classifier trained on the source data . If we try to create compressed representation of both source and target data then in the latent space(i.e hidden layer representation) the distance between data distribution of source and target domain is likely to reduce . We can not say that in latent space representation data distribution of source and target data will exactly match but at least it will be

better than original feature space. In general we create an auto-encoder trained with both the source and target data and then we try to generate latent representation of the total data (both source and target) . In this process auto-encoder learns some interesting features of the total data and in the hidden layer it generates an encoded format . Now we train a classifier on the encoded data of source and test it on target data. Since we are using latent representation,a classifier may do a better job in predicting the class labels of the target data.

In simple auto-encoder we create one encoded form in hidden layer but in stacked auto-encoder we again use the encoded form to create another encoded form in next layer of auto-encoder.A stacked auto-encoder enjoys all the benefits of any deep network of greater expressive power.Further, it often captures a useful "hierarchical grouping" or "part-whole decomposition" of the input. To see this, recall that an auto-encoder tends to learn features that form a good representation of its input. The first layer of a stacked auto-encoder tends to learn first-order features in the raw input (such as edges in an image). The second layer of a stacked auto-encoder tends to learn second-order features corresponding to patterns in the appearance of first-order features (e.g., in terms of what edges tend to occur together–for example, to form contour or corner detectors). Higher layers of the stacked auto-encoder tend to learn even higher-order features.For domain adaptation higher order feature is definitely useful. So,we take the encoded form of the last layer and use it for domain adaptation. There are other variants of auto-encoder [8] that uses different kind of structure and constraint to get better latent representation of input data.

Chapter 4

Domain Adaptation preserving topology:

Domain adaptation is a part of transfer learning . In domain adaptation we have enough labelled source data but no label data for the target domain. The source and target tasks are the same , while the statistical characteristic of the source and target domains are different. For example,the feature space for both source and target domain are the same but marginal probability distribution of input data are different.

One key idea is to find good feature representation that reduce the distance between the two domains. So our goal is to find a common representation for source and target domain. Our assumption is that if we find a common representation then the distance between source and target domain will be reduced. In other words we can say that marginal probability distribution of both source and target data may match if we get a suitable common representation. After getting a common representation , we can train classifier (or regression) on the source domain and test on the target domain data . Actually getting a common representation is projecting the total data into a latent space satisfying some desirable properties. In this latent space source and target data distributions are likely to come closer.

But in this process of finding common representation we are not

considering the shape of the data. At the time of generating a common representation shape of the data may be changed and that can create trouble at the time of testing of classifier algorithm on the target data . Both common representation of the source data as well as of the target data can differ from its original shape in the original feature space.

But the question is whether we should consider geometric shape or topological shape .We generally find a common representation that has less dimensionality compared to original feature space. For an example suppose we have three dimension data having a spherical shape . If we want to find a common representation in two dimension space then the shape of the common representation will be some two dimensional geometric shape . We can't compare two dimensional geometric shape with three dimensional geometric shape. Also geometric shape doesn't indicate relative position of data points which is crucial for training a classifier. So we can check if neighbourhood of a point is intact in lower dimensional space of the common representation.

If two data points are close in the original feature space then in the lower dimensional representation that two points should be close . In other words, we can say that points in neighbourhood of the data point in original feature space should be the points in the neighbourhood of that data point in the common representation(lower dimensional representation). Since we are trying to preserve neighbourhood, topology is the best criteria to measure whether neighbourhood is preserved or not.

Chapter 5

Preserving topology in autoencoder

Preserving topology in autoencoder

As previous discussion of autoencoder we have seen that autoencoder has encoder part and decoder part. Encoder part creates an encoded representation in a hidden layer and decoder part reconstructs the original data from that hidden layer representation . Generally hidden layer has a lower dimension compared to the original dimension of data.

5.1 Adding topology constraint in autoencoder:

We have used sammons stress function as a penalty the loss function of our autoencoder.

Sammon mapping or Sammon projection is an algorithm that maps a high-dimensional dataset to a lower dimensional dataset by trying to preserve the structure of inter-point distances in the high-dimensional space in to the lower-dimension projection. It is particularly suited for use in exploratory data analysis. The method was proposed by John W. Sammon in 1969 [9]. It is a non-linear ap-

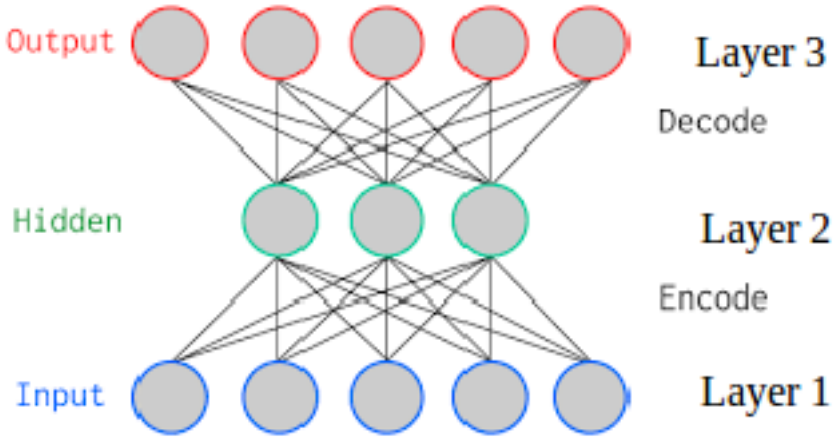
proach as the mapping can't be represented as a linear combination of the original variables as possible in techniques such as principal component analysis, which makes it more difficult to use for classification applications.
We denote the distance between i th and j th objects in the original space by d^*_{ij}, and the distance between their projections by d_{ij}. Sammon's mapping aims to minimize the following error function, which is often referred to as Sammon's stress or Sammon's error:

$$E = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}$$

The tendency to preserve local neighbourhood is favoured by the factor of d^*_{ij} in the denominator of main summation, ensuring that if the original distance between two points is small ,then weight given to their squared difference is high.

The minimization can be performed either by gradient descent, as proposed initially, or by other means, usually involving iterative methods. The number of iterations need to be experimentally determined.
Since normal auto-encoder finds optimal parameter using gradient descent, adding Sammons stress function won't be too difficult. If we add Sammons stress function to the loss function of auto-encoder then normal gradient descent is enough to minimise total loss function .

Autoencoder with Sammon's loss :



The structure is same as normal auto-encoder . We are just using an added constraint. We are not using any other extra constraint(such as sparsity constraint). Also we are using batch learning procedure for training our modified auto-encoder. We have used different strategy to create batches. We will discuss that in the next section.
We are using the notation that we used in chapter 2. We are using reconstruction loss and Sammons loss as our loss function. So, our total loss function is
(Notations are defined in chapter 2).

$$E_{total} = a * E_{reconstruction} + b * E_{sammon} \tag{5.1}$$

$$where\ a \geq 0\ ;\ b \geq 0 \tag{5.2}$$

We are taking a=1 and b=1 for simplification of our gradient calculation.

$$E_{total} = E_{reconstruction} + E_{sammon} \tag{5.3}$$

$$E_{reconstruction} = \frac{1}{n}(\sum_{u=1}^{n}(\frac{1}{2}\sum_{k=1}^{N}(y_k^3(u) - Y_k(u))^2) \tag{5.4}$$

$y_k^3(u)$ = output of unit k at third layer for a given input data point in a batch indexed by u
$Y_k(u)$ = value of unit k of given input data point in a batch indexed by u

$$E_{sammon} = \frac{1}{\sum_{u=1}^{n-1}\sum_{v=u+1}^{n} d^*(u,v)} \sum_{u=1}^{n-1}\sum_{v=u+1}^{n} \frac{[d^*(u,v) - d(u,v)]^2}{d^*(u,v)} \tag{5.5}$$

$$E_{sammon} = \sum_{u=1}^{n-1}\sum_{v=u+1}^{n} E_{sammon}^{uv} \tag{5.6}$$

$$E_{sammon}^{uv} = \lambda \frac{[d^*(u,v) - d(u,v)]^2}{d^*(u,v)} \tag{5.7}$$

$$where\ \lambda = \frac{1}{\sum_{u=1}^{n-1}\sum_{v=u+1}^{n} d^*(u,v)}$$

E_{sammon}^{uv} is the Sammon's loss for two data point indexed by u,v.
$d^*(u,v)$ =distance between two input data indexed by u,v in original dimension
$d^(u,v)$ =distance between hidden layer representation of two input data indexed by u,v

$E_{reconstruction}$ is same as used in normal auto-encoder[10].We have already given the gradient of $E_{reconstruction}$ with respect to the weights in chapter 2. We have to consider E_{sammon} and find its gradient. Sammons loss desn't depend on the output of layer 3 (i.e output layer) hence gradient of E_{sammon} w.r.t weight between layer 2 and layer 3 is zero. i.e $\frac{\partial E_{sammon}}{\partial W_{jk}^{23}} = 0 \forall j,k$ in layer 2 and layer 3; where j is index in layer 2 and k is index in layer 3. So,for all weights between layer 2 and layer 3 gradient will be similar to normal auto-encoder.

But for all weights between layer 1 and layer 2 gradient will be different . We have to calculate gradient of E_{sammon} w.r.t to the weights between layer 1 and layer 2

$$\frac{\partial E_{sammon}}{\partial W_{ij}^{12}} = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \frac{\partial E_{sammon}^{uv}}{\partial W_{ij}^{12}}$$

where

$$\frac{\partial E_{sammon}^{uv}}{\partial W_{ij}^{12}} = \frac{\partial E_{sammon}^{uv}}{\partial d(u,v)} \frac{\partial d(u,v)}{\partial [y_j^2(u) - y_j^2(v)]} \frac{\partial [y_j^2(u) - y_j^2(v)]}{\partial W_{ij}^{12}}$$

$$\frac{\partial E_{sammon}^{uv}}{\partial W_{ij}^{12}} = (-2\lambda \frac{d^*(u,v) - d(u,v)}{d^*(u,v)})(\frac{y_j^2(u) - y_j^2(v)}{d(u,v)})(g^{'}(h_{j,u}^2) y_i^1(u) - g^{'}(h_{j,v}^2) y_i^1(v)) \tag{5.8}$$

where $g^{'}(h_{k,.}^2)$ is the derivative of the output of unit k in layer 2 with respect to the net input $h_{k,.}^2$ and $g^{'}(h_{k,.}^2) = (1 - y_k^2(.))(y_k^2(.))$ if sigmoid is our activation function . Since we have calculated $\frac{\partial E_{sammon}^{uv}}{\partial W_{ij}^{12}}$, we can calculate $\frac{\partial E_{sammon}}{\partial W_{ij}^{12}}$. Also we can calculate the total gradient $\frac{\partial E_{total}}{\partial W_{ij}^{12}}$. Hence we are writing the gradients of E_{total} with respect to W_{ij}^{12} and W_{jk}^{23} .

$$\frac{\partial E_{total}}{\partial W_{jk}^{23}} = \frac{1}{n} \sum_{u=1}^{n} \big\{ - (Y_k(u) - y_k^3(u)) g^{'}(h_{k,u}^3) y_j^2(u) \big\} \tag{5.9}$$

$$\frac{\partial E_{total}}{\partial W_{ij}^{12}} = \frac{1}{n} \sum_{u=1}^{n} \big\{ - (\sum_{k=1}^{p} \big\{ (Y_k(u) - y_k^3(u)) g^{'}(h_{k,u}^3) W_{jk}^{23} \big\}) g^{'}(h_{j,u}^2) y_i^1(u) \big\}$$
$$+ \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \big\{ (-2\lambda \frac{d^*(u,v) - d(u,v)}{d^*(u,v)})(\frac{y_j^2(u) - y_j^2(v)}{d(u,v)})(g^{'}(h_{j,u}^2) y_i^1(u) - g^{'}(h_{j,v}^2) y_i^1(v)) \big\} \tag{5.10}$$

5.2 Outline of Learning Procedure for modified auto-encoder:

The basis procedure of gradient descent-based learning for our modified auto-encoder is outlined as follows in steps 1 - 6.

1.Select a a pattern from the batch and present it to the modified auto-encoder.
2.Compute activations and signals of input,hidden and output layer neurons in that sequence (basically it's a feed forward of input data)
3.After presenting all input pattern from the batch ,compute the error(E_{total}) at the output layer by using equation no (5.3),(5.4) and (5.5).
4.Use the error calculated in Step 3 to compute the change(gradient) in the hidden to output layer weights and the change in input to hidden layer weights according to equation no (5.9) and (5.10).
5. Update all weights using the change(gradient) computed in step 4.

$$W_{jk}^{23} = W_{jk}^{23} + \eta(-\frac{\partial E_{total}}{\partial W_{jk}^{23}})$$

$$W_{ij}^{12} = W_{ij}^{12} + \eta(-\frac{\partial E_{total}}{\partial W_{ij}^{12}})$$

6. repeat Steps 1 through until E_{total} falls below a predefined threshold.

Since we are using coefficient(a,b) then we can regulate the amount of importance given to a particular loss(reconstruction or Sammon's loss).

In the above formulation we are considering batch learning. Since we are using topology so we have to use more than one data point to train the auto-encoder. But the question is how to create a batch accordingly so that topology of original can remain intact. We can create batch randomly as generally we do for training a neural

network. Taking random points in a batch wont be a best choice for preserving topology. So here we are creating a batch by taking a point and its neighbourhood. In the next section we are going to discuss how we are creating batches.

5.3 Neighbourhood selection for a point:

Neighbourhood selection for a point is very simple. Neighbourhood of a data point means a set of points close to that particular point. So,we have to find the close points that surrounds that particular point. We take a point and try to find some fixed number of points which are close to that particular point . Then we create a batch using those closest points . If dimension of data is small then finding neighbourhood is easy using brute force method. But if dimension is large then brute force method will take a lot of time. It will increase the preprocessing time. We can reduce the time of batch creation by using probabilistic approach to find close points of a given point.

We have used locality-sensitive hashing in order to speed up the batch creation process.Locality-sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same buckets with high probability (the number of buckets being much smaller than the universe of possible input items). LSH differs from conventional and cryptographic hash functions because it aims to maximize the probability of a collision for similar items. Locality-sensitive hashing has much in common with data clustering and nearest neighbour search.

LSH is based on the simple idea that, if two points are close together then after a projection operation these two points will remain close together. Two points that are close together on the sphere are also close together when the sphere is projected onto the two-dimensional page. This is true no matter how we rotate the sphere. Two other points on the sphere that are far apart will, for some orientations, be close together on the page, but it is more likely that the points will remain far apart[11].

The accuracy of an LSH is determined by probability that it will find the true nearest neighbour. Although LSH is probabilistic in nature it is widely used in clustering of high dimensional data. Also in randomised algorithm LSH is widely used.

5.4 Algorithm for creating a batch:

D=[$\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_n}$] are the total data points for which we want to create batches.
We want to divide the whole dataset into batches with size k.

B is an empty set that is used for storing batches ;
while *D not empty* **do**
  if *size of D is less than equal to k* **then**
    Create a batch using the points in D , store the batch in B and break
  else
    Create hash table with D using LSH technique;
    select one point P randomly from D;
    find the set S of k-1 nearest points of P by searching in the hash table;
    Create a batch using the point P and it's nearest point set S, store the batch in B;
    delete the points in S and the pont P from D;
    repeat
  end
end
Output: a set B of batches for training the auto-encoder

Algorithm 1: Create Batch Algorithm

We have used this algorithm to create batches from dataset(both source and target dataset) to train our modified auto-encoder.

5.5 Procedure for training:

Source dataset is denoted as S and target dataset is denoted as T. Total dataset consists of both source and target dataset.

Our method of training:

1.Create batches from source dataset S using CreateBatch algorithm and also create another batches from target dataset T using same CreateBatch algorithm.
2.train an auto-encoder with Sammon's loss with the total batches (i.e source plus target dataset batches) as discussed in section 5.2
3.After training, generate hidden layer representation of total dataset using that trained auto-encoder. Hidden layer representation of source dataset is denoted as H_s and Hidden layer representation of target dataset is denoted as H_t
4. Divide the latent projection of source data into training set $H_s{}'$ and validation set V_s.
5.train a SVM classifier C_l using $H_s{}'$.
6. do a cross-validation to get optimum parameter for the SVM classifier C_l using the validation set V_s.
7.test the classifier C_l on H_t (i.e the hidden layer representation of target data set) .

Chapter 6

Experimental Result:

For experiments , we consider the Office [12] datasets. The dataset contains 31 objects categories collected from three different sub-datasets: Amazon, DSLR (digital single-lens reflex camera), and webcam. Following the same settings applied in [13], we select the 10 over-lapping object categories of Office for experiments, and produce three different domains of interest: Amazon (A),DSLR (D), and webcam (W). As a result, a total of 6 different cross-domain pairs will be available (e.g.,A->W,W->D etc.).We extracted 10 classes common to all four datasets: BACKPACK, TOURING BIKE, CALCULATOR, HEADPHONES, COMPUTER KEYBOARD, LAPTOP-101, COMPUTER MONITOR, COMPUTER MOUSE,COFFEE-MUG,AND VIDEO PROJECTOR. There are 8 to 151 samples per category per domain, and 2533 images in total. We report in the main text our results on the 10 common classes.

The above dataset contains different kinds of features but we have used SURF features for our experiment . We have also used SVM(support vector machine) as our desired classifier. Since we are using coefficient a and b (as explained in previous chapter) to control the effect of topological constraint we have to define a fixed value for a and b. In our experiment, we have fixed a=1.0 but used different b values . Besides that, the parameter of SVM is derived using cross validation.

We have used linear kernel in our SVM classifier. Using cross-

validation we have found optimum value for C (penalty parameter of the error term). We have used different values of C for checking validation accuracy. We have started the value of C from 0.001 up to 5 and increased it by 0.001. For each value in that range(0.001 to 5) we have checked validation accuracy and found the best one. The size of SURF feature is 800 and our auto-encoder has 500 nodes in hidden layer. Our batch size is 20. Below table shows accuracy of different algorithm. SA stands for subspace alignment[14] and CORAl stands for Correlation Alignment[15].

method	A -> D	A-> W	D->A	D-> W	W-> A	W-> D
SA	37.6	37.1	38.1	79.2	37.3	78.2
CORAL	38.7	38.3	38.2	81.7	38.8	84.0

Now, we are presenting our experimental result one by one for different b values.

our method	A -> D	A -> W	D -> A	D -> W	W -> A	W -> D
b=0.0	39.896	43.6808	30.1234	62.2356	32.00	75.2345
b=0.0	38.1480	42.255	31.2547	63.5478	32.1245	75.486
b=0.0	39.2345	43.26879	30.1234	63.2356	32.3245	75.1235
b=0.0	38.3215	42.6808	31.5478	63.8478	32.4125	74.345
b=0.0	39.800	43.4724	31.2547	63.3156	31.50	74.5145
Avg	39.08	43.07	30.861	63.236	32.07	75.141

our method	A -> D	A -> W	D -> A	D -> W	W -> A	W -> D
b=0.1	42.857	48.0951	32.821	66.390	34.00	78.571
b=0.1	42.259	48.0651	33.0124	65.1245	34.5	77.057
b=0.1	43.2077	48.1751	32.548	66.102	35.0	78.865
b=0.1	42.2597	47.657	32.821	65.1245	34.5	77.571
b=0.1	42.857	48.0651	33.0124	66.252	35.0	78.865
Avg	42.689	48.011	32.84	65.79	34.6	78.186

our method	A -> D	A -> W	D -> A	D -> W	W -> A	W -> D
b=0.2	43.649	45.106	32.123	65.383	33.750	77.8714
b=0.2	43.00	44.6808	31.2547	66.1246	34.00	77.0571
b=0.2	44.149	44.2553	31.5478	65.1025	34.890	77.5714
b=0.2	43.00	45.10	32.1240	66.3283	34.235	78.00
b=0.2	43.716	45.243	31.2547	65.384	33.750	77.9564
Avg	43.50	44.87	31.66	65.664	34.125	78.09

our method	A -> D	A -> W	D -> A	D -> W	W -> A	W -> D
b=0.3	42.857	46.095	32.3214	65.273	33.450	77.8714
b=0.3	41.26	47.065	33.41246	66.245	34.00	77.920
b=0.3	41.207	46.075	32.5478	65.2125	34.890	77.57142
b=0.3	40.860	47.657	33.01246	66.383	34.245	77.8714
b=0.3	41.26	46.095	33.01246	66.245	33.750	77.542142
Avg	41.49	46.597	32.86	65.871	34.067	77.755

So we can check that for some domain(A->D,etc.) accuracy is much higher compared to other algorithm. So this results prove that topology matters in case of domain adaptation.

References

[1] http://socrates.acadiau.ca/courses/comp/dsilver/nips95_ltl/nips95.workshop.pdf.

[2] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2007.

[3] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[4] Irvan B Arief-Ang, Flora D Salim, and Margaret Hamilton. Da-hoc: semi-supervised domain adaptation for room occupancy prediction using co 2 sensor data. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments*, page 1. ACM, 2017.

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[6] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[9] John W Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 100(5):401–409, 1969.

[10] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[11] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.

[12] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2066–2073. IEEE, 2012.

[13] Nazli Farajidavar, Teofilo de Campos, and Josef Kittler. Transductive transfer machine. In *Asian Conference on Computer Vision*, pages 623–639. Springer, 2014.

[14] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE international conference on computer vision*, pages 2960–2967, 2013.

[15] Baochen Sun, Jiashi Feng, and Kate Saenko. Correlation alignment for unsupervised domain adaptation. In *Domain Adaptation in Computer Vision Applications*, pages 153–171. Springer, 2017.