HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

MASTER THESIS

Mr.
Mehrdad Mohannazadeh Bakhtiari

Mathematical Considerations on Soft Learning Vector Quantisation and Robust Soft Learning Vector Quantisation

2018

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Mathematical Considerations on Soft Learning Vector Quantisation and Robust Soft Learning Vector Quantisation

Author:
Mehrdad Mohannazadeh Bakhtiari

Study Programme:
Applied Mathematics in Digital Media

Seminar Group:
MA15w2-M

First Referee:
Prof. Dr. Thomas Villmann

Second Referee:
Dr. Marika Kaden

Mittweida, April 2018

Bibliographic Information

Mohannazadeh Bakhtiari, Mehrdad: Mathematical Considerations on Soft Learning Vector Quantisation an Robust Soft Learning Vector Quantisation, 88 pages, 0 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences

Master Thesis, 2018

Abstract

Soft Learning Vector Quantisation (SLVQ) and Robust Soft Learning Vector Quantisation (RSLVQ) are supervised data classification methods, that have been applied successfully to real world classification problems. The performance of SLVQ and RSLVQ, however, reduces, when they are applied to more complicated classification problems. In this thesis, we have introduced modifications to SLVQ and RSLVQ, in order to have more capable versions of them. A few possibilities to modify SLVQ and RSLVQ are considered, some of them are not successful enough and they have been included for the sake of completeness. The fruits of the thesis are plenty, including Tangent Soft Learning Vector Quantisation-Strong (TSLVQ-S), together with its more stable version Tangent Robust Soft Learning Vector Quantisation-Strong (TRSLVQ-S), Attraction Soft Learning Vector Quantisation (ASLVQ) and Grassmannian Soft Learning Vector Quantisation (GSLVQ).

Table of Content

1- Introduction

SLVQ (Soft Learning Vector Quantisation) and its modified version RSLVQ (Robust Soft Learning Vector Quantisation), which brings more stability in the learning process, are rather successful classifiers of data in $\mathbb{R}^n$. The two methods were proposed by S. Seo and K. Obermayer [1] and the corresponding paper would be the main reference for what is included, about SLVQ and RSLVQ, in this thesis. Although the methods are known to give good classifications, they have a fundamental disadvantage. To be clear, the Euclidean distance is assumed to be a valid dissimilarity measurement to compare two objects in $\mathbb{R}^n$. This assumption is too naïve for real world problems. An example would be given to make the point more clear, in the first paragraph of section 4. To overcome the problem, we would like to use tangent distances and Geodesic distance (defined on a Grassmannian manifold), as substitutions for the Euclidean distance. The concept of tangent distances is well explained in papers by P. Simard et al. [2] and S. Saralajew and T. Villmann [3]. Although the papers have different assumptions, the underlying concept stays the same. For Grassmannian manifolds and the corresponding distances, defined on the manifolds, basic information can be found on a paper by S.Chepushtanova and M.Kirby [8].

Intuitively, one may replace the Euclidean distance with a tangent distance, which is not quite correct, as it conflicts the philosophy of what is described in the paper by S. Seo and K. Obermayer [1]. Therefore, we would like to mention the right way to make the transition from the original SLVQ to SLVQ, with tangent distances as the dissimilarity measurement. The right way comes with complexity of the corresponding updating rule. Therefore, as inaccurate alternatives, simplifying assumptions would be made and the corresponding updating rules would be derived.

After using tangent distances in SLVQ, we would like to have Grassmannian manifolds as the fundamental structure, on which SLVQ is applied. The motivation is to have a tool that does not depend on the differentiability of the data manifolds (with tangent distances, it is assumed that the data manifolds are differentiable). Grassmannian concept gives us the opportunity to make new abstract points, from the original data points. In the new data space, a new distance measurement (usually geodesic) is introduced. The new space, together with the dissimilarity measurement, is supposed to reflect the non-Euclidean dissimilarity of data points.

Here is a short description of what is coming in the next sections. In section 2, some of the definitions and symbols, that would be used throughout the thesis, are included. There might be symbols that are not defined in section 2. Those symbols are described locally. Sections 3 and 4 give a brief introduction on SLVQ and tangent distances, respectively. Section 5 is dedicated to explain how tangent distances should be integrated in SLVQ. Grassmannian SLVQ is treated in Section 6, where we start with an introduction to Grassmannian manifolds and explain the way it should be combined with SLVQ. The result would be GSLVQ (Grassmannian Soft Learning Vector Quantisation), which is not to be confused with “Generalised Learning

Vector Quantisation” (GLVQ). In sections 7, conclusions have been made. Section 8 contains the Matlab programs, that are written to test the developed methods, described in this thesis. Note that there would be no programs provided for GSLVQ. In sections 9 and 10, a list of abbreviations, that are used in the thesis, and references are included, respectively.

2- Definitions and notations

The main problem is a classification problem, in $\mathbb{R}^n$. For all $\mathbb{X} \in \mathbb{R}^n$, we would like to have a function $F : \mathbb{R}^n \ \rightarrow L$, where $L = \{1,2,...,c\}$ is the set of possible classes. The function $F(\mathbb{X}) = l$ is called a classifier that assigns each $\mathbb{X} \in \mathbb{R}^n$ to a class $l \in L$. We would be interested in a certain classification, given by F, dictated by a finite training set $T = \{(\mathbb{X}_i, y_i) \,|\, \mathbb{X}_i \in \mathbb{R}^n, y_i \in L, i = 1,2,..., \,|\ T\ |\ \}$. Therefore, a primary intension is to find a function F, such that$F(\mathbb{X}_i) = y_i \forall i = 1,2,..., |T|$. A function $C_T(\mathbb{X}_i) = y_i$ is defined to contain the relation $(\mathbb{X}_i, y_i)$, in the set T. $C_T(\mathbb{X})$ is allowed to accept any other values for an $\mathbb{X}$, for which there is no relation in the training set T. $C_T(\mathbb{X})$ would be called the training function, with respect to the training set T.
In order to assess the suitability of function F, with respect to a test set T', classification error rate is commonly used. T' is of the same format as T (it can be T itself), but usually with new samples. Note that classification error rate should be minimised and it is as below.

$$CER = \frac{\sum_{i=1}^{|T'|} 1 - \delta\big(F(\mathbb{X}_i) - C_{T'}(\mathbb{X}_i)\big)}{|T'|} \quad (1)$$

$\delta(x)$ is Dirac delta function. Although CER is the main cost function, it is not used in the learning process, but rather, it would be the final check for the performance of the learning algorithms.

To find a suitable function F, one needs to define a model for F, with some adaptive parameters (this is a regular process in machine learning). The model, that is used in “Learning Vector Quantization” (LVQ, introduced by T. Kohonen), is as follows. A set of prototypes is defined.

$$W = \{\mathbb{W}_i^j \in \mathbb{R}^n \,|\, j \in L, i = 1,2,...,k_j, k_j \in \mathbb{N}\}$$

k_j prototypes are dedicated to a class j. Having W, the function F can be defined as below.

$$F(\mathbb{X}, W) = arg_j(min_{i,j}||\mathbb{X} - \mathbb{W}_i^j||) \ (2)$$

where $||.||$ is the Euclidean norm. Note that F takes two arguments, in the relation above. However, it would only depend on a single argument, when the training phase is finished and W becomes fixed.

Having a set of prototypes $W = \{\mathbb{W}_i | i = 1,...,k\}$in $\mathbb{R}^n$, the Voronoi cell of a prototype $\mathbb{W}_i \in W$, with respect to W, written as $V(\mathbb{W}_i | W)$ is define as below.

$$V(\mathbb{W}_i | W) = \{\mathbb{X} \in \mathbb{R}^n | \ ||\mathbb{X} - \mathbb{W}_i|| \leq ||\mathbb{X} - \mathbb{W}_j|| \ \forall j \neq i\}$$

The model, described so far, is called crisp classification. In contrast, fuzzy classifiers assign to each object $\mathbb{X}$ a probability vector $P(\mathbb{X})$. The j-th element of the vector $P_j(\mathbb{X})$ gives the probability that the object $\mathbb{X}$ belongs to the class j. At the end of a fuzzy learning process, one has the option of classifying an object $\mathbb{X}$ into the class $j \in L$, for which $P_j(\mathbb{X})$ is the largest. However, in some applications, if two or more largest elements of $P(\mathbb{X})$ are close to each other, the data point $\mathbb{X}$ is preferred not be classified, as there is a notable level of uncertainty. According to the definition of a fuzzy classifier, “Soft Learning Vector Quantisation”
(SLVQ) can be thought as a fuzzy classifier, as it implicitly uses probabilities to adapt prototypes.

To achieve a desirable function F, parameters W may be changed, according to an online learning scheme. W is initialised randomly (or using a more intelligent method). At time t, a sample $(\mathbb{X}, y) \in T$ is chosen randomly, with all the members having an equal probability of being selected. Then, the update happens according to certain rules, in such a way that $F(\mathbb{X}, W)$ gives a better classification.

Gaussian distribution is a key element in SLVQ and RSLVQ. Therefore, its notation is mentioned here. $N(\mu, \sigma)$ denotes a Gaussian distribution, with mean μ and standard deviation σ. A variation of the same notation is used to address a point (vector) in $\mathbb{R}^n$, that is produced, using a Gaussian distributed sample producer, with parameters μ and σ. Note that $\mu \in \mathbb{R}^n$ and $\sigma \in \mathbb{R}$. The notation is $N_{\mathbb{R}^n}(\mu, \sigma)$. Therefore, $N_{\mathbb{R}^n}(\mu, \sigma)$

denotes a random vector $\mathbb{X}$in $\mathbb{R}^n$, whose production probability is $p(\mathbb{X} \mid \mu, \sigma)$ (refer to equation (4)).

When the probability of a point $\mathbb{X} \in \mathbb{R}^n$ is to be measured, given the distribution is $N(\mathbb{W}, \sigma)$, the following formula is used.

$$p(\mathbb{X} \mid \mathbb{W}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{d^2(\mathbb{X}, \mathbb{W})}{2\sigma^2}) \ (3)$$

In the above formula, $\mathbb{W}$ and $\mathbb{X}$ are vectors in $\mathbb{R}^n$ and $d(\,.\,,\,.\,)$ is a general distance measure.

Having a matrix A, the element, that is located at row i and column j, is denoted by

$$A_{ij} \ or \ \{A\}_{ij}$$

Every now and then, it is required to extract a certain row or column of a matrix A. For that purpose, we would like to use notations, that are used in Matlab programming. To indicate the i-th row and j-th column of matrix A, notations $A_{i,:}$ and $A_{:,j}$ are used, respectively.

Finally, the vectorisation operation, on a $m \times n$ matrix A, is denoted by $vec(A)$ and it is defined as below. $vec(A)$ is a column vector, with $n \times m$ elements.

$$\{vec(A)\}_i = A_{a,b}$$

Where $a = rem(i-1,n) + 1$ and $b = \lfloor \frac{i-1}{n} \rfloor + 1$. Note that $rem(p, q)$ is the remainder of dividing p by q and $\lfloor\,.\,\rfloor$ is the floor function.

3- An introduction and discussions on SLVQ and RSLVQ

Section 3-1 gives a general framework for stochastic gradient learning. In sections 3-2 and 3-4, Soft Learning Vector Quantisation (SLVQ) and Robust Soft Learning Vector Quantisation (RSLVQ) are introduced. Not only the key concepts are mentioned in order to integrate the new modifications into the original concepts more meaningfully, but also a few comments, about the original SLVQ and RSLVQ are made.

Section 3-3 explains a new way of looking at SLVQ. A fruit of the new approach is Attraction SLVQ (ASLVQ).

3-1- Stochastic Gradient Ascent learning

We introduce a general framework, which is going to be used throughout the paper. A Likelihood function $L_r(T, W)$ is defined, where T is a training set and W is a set of adaptable parameters. Moreover $L_r(T, W)$ is defined as

$$L_r(T, W) = \prod_{\mathbb{X} \in T} L_p(\mathbb{X}, W)$$

$L_p(\mathbb{X}, W)$ is the partial likelihood function. In words, $L_r(T, W)$ can be written as the product of partial likelihood function $L_p(\mathbb{X}, W)$, each depending on a single training sample from T.

Given a random $\mathbb{X} \in T$, at time t, the Stochastic Gradient Ascent Learning (SGAL), with respect to $\mathbb{W} \in W$, is defined as

$$SGAL(L_p, \mathbb{X}, \mathbb{W}) = \frac{\partial}{\partial \mathbb{W}} L_p\left(\mathbb{X}, W\right)\Big|_{W(t)} \ (4a)$$

The updating rule of the adaptable parameters $\mathbb{W} \in W$ would always be

$$\mathbb{W}(t+1) = \mathbb{W}(t) + \alpha \, . \, SGAL(L_p, \mathbb{X}) \ (4b).$$

α is the updating rate, which is a constant.

For the rest of the thesis, the updating rules will not be included. Only the corresponding SGALs would be included, assuming the reader knows how to apply SGALs to update parameters.

3-2- Soft learning vector quantisation (SLVQ)

S. Seo and K. Obermayer [1] proposed a method of learning vector quantisation, called “Soft Learning Vector Quantisation” (SLVQ). A flaw in traditional LVQ methods is that the same updating rate α is applied to all the prototypes, at a certain time t. So, no matter which prototype gets updated, at time t, the rate would be $\alpha(t)$. Equations (9a) and (9b) will make it clear that the updating rule of SLVQ depends on more parameters, to pinpoint the right updating rate. In this section, we are going to introduce the philosophy of SLVQ and then argue that, although there are minor issues with SLVQ, it may serve as a nice framework to learn classification problems.

A goal of the paper, published by S. Seo and K. Obermayer [1], is to define a learning model that is easy to treat mathematically. In a key step, they assume that the data points are produced by a Homogeneous Gaussian Mixture model. To describe it, in a simple way, each prototype is assumed to be responsible for a Gaussian distribution $N(\mathbb{W}_i^j, \sigma_i^j)$, in $\mathbb{R}^n$. The standard deviation σ_i^j is unknown (it would be initialised and adapted, in the learning process). As explained in the same paper, homogeneity addresses the fact that a prototype $\mathbb{W}_i^j$ would only produce data points of class j. With a class j and a prototype $\mathbb{W}_i^j$, probabilities $p^j$and p_i^j are associated, respectively. p^j is the probability that an arbitrary prototype, from class j, is given the chance to produce a point, in $\mathbb{R}^n$, and p_i^j is the probability that the prototype $\mathbb{W}_i^j$ is the prototype to produce a data point, in $\mathbb{R}^n$. According to what is described so far, to have a rough idea about the distribution of data in $\mathbb{R}^n$, one may choose a prototype $\mathbb{W}_i^j$, with probability p_i^j, and a point $\mathbb{X}$, is randomly generated, according to $N(\mathbb{W}_i^j, \sigma_i^j)$. To emphasise, this is the model that is assumed to be responsible for producing the training set T.
Assuming a Gaussian mixture model, being responsible for data production, S. Seo and K. Obermayer [1] proposed a learning method to adapt the prototypes. The method is based on the following cost function.

$$L_r(T, W) = \prod_{i=1}^{|T|} \frac{p(\mathbb{X}_i, y_i \,|\, W)}{p(\mathbb{X}_i, \bar{y}_i \,|\, W)} \quad (5)$$

In the above expression, $p(\mathbb{X}_i, y_i \,|\, W)$ is the probability that $\mathbb{X}_i$ is produced by a prototype, responsible for class y_i, given that the prototypes are distributed as given by W. Mathematically, it can be written as

$$p(\mathbb{X}_i, y_i \,|\, W) = \sum_t p(\mathbb{X}_i \,|\, \mathbb{W}_t^{y_i}) \,.\, p_t^{y_i} \ (6).$$

In a rather similar manner, $p(\mathbb{X}_i, \bar{y}_i \,|\, W)$ is the probability that $\mathbb{X}_i$ is produced by a prototype, responsible for a class other than y_i, given that the prototypes are distributed as given by W. It can be written as

$$p(\mathbb{X}_i, \bar{y}_i \,|\, W) = \sum_{k \neq y_i} \sum_t p(\mathbb{X}_i \,|\, \mathbb{W}_t^k) \,.\, p_t^k \ (7).$$

The conditional probability $p(\mathbb{X} \,|\, \mathbb{W}_i^j)$ is the probability that $\mathbb{X}$ is produced, when prototype $\mathbb{W}_i^j$ is the producer. Its value is readily given by a Gaussian component of the mixture, using formula (4).

Note that σ is assumed to be fixed for all the prototypes. Therefore, it would not be mentioned in the argument of the conditional probability. In a more general treatment, one can have different standard deviations for different prototypes. We are not going to investigate the general form in this thesis.

The logarithmic version of cost function (5) is

$$logL_r(T, W) = \sum_{i=1}^{|T|} log \frac{p(\mathbb{X}_i, y_i | W)}{p(\mathbb{X}_i, \bar{y}_i | W)} \quad (8a)$$

The partial likelihood function of (8) is

$$logL_p\big((\mathbb{X}, y), W\big) = log \frac{p(\mathbb{X}, y | W)}{p(\mathbb{X}, \bar{y} | W)} \quad (8b)$$

According to equation (8b)

$$SGAL\big(L_p, (\mathbb{X}, y), \mathbb{W}_i^j\big) = \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y | W) - \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} | W) \quad (9a)$$

Using chain rule, the following is obtained. Note that $d(\mathbb{X}, \mathbb{Y})$ is a general dissimilarity measurement.

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y | W) = \frac{1}{p(\mathbb{X}, y | W)} \cdot \frac{\partial p(\mathbb{X}, y | W)}{\partial p(\mathbb{X} | \mathbb{W}_i^j)} \cdot \frac{\partial p(\mathbb{X} | \mathbb{W}_i^j)}{\partial d(\mathbb{X}, \mathbb{W}_i^j)} \cdot \frac{\partial d(\mathbb{X}, \mathbb{W}_i^j)}{\partial \mathbb{W}_i^j} \quad (9b)$$

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} | W) = \frac{1}{p(\mathbb{X}, \bar{y} | W)} \cdot \frac{\partial p(\mathbb{X}, \bar{y} | W)}{\partial p(\mathbb{X} | \mathbb{W}_i^j)} \cdot \frac{\partial p(\mathbb{X} | \mathbb{W}_i^j)}{\partial d(\mathbb{X}, \mathbb{W}_i^j)} \cdot \frac{\partial d(\mathbb{X}, \mathbb{W}_i^j)}{\partial \mathbb{W}_i^j} \quad (9c)$$

In case the Euclidean distance is chosen for $d(\mathbb{X}, \mathbb{W}_i^j)$, $p(\mathbb{X} | \mathbb{W}_i^j)$ follows the Gaussian distribution, as stated in equation (3), and $p(\mathbb{X}, y | W)$ and $p(\mathbb{X}, \bar{y} | W)$ are as described in formulas (6) and (7), equation (9a) becomes

$$\frac{\partial}{\partial \mathbb{W}_i^j} logL_r(T,W) = 2.\left[\delta(j-y)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},y\,|\,W)} - (1-\delta(j-y))\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{X}_i^j)}{\sigma^2 p(\mathbb{X},\bar{y}\,|\,W)}\right](\mathbb{X}-\mathbb{W}_i^j)\ (9d)$$

Function $\delta(x)$ is Dirac delta function.

Another way to calculate the gradient is as the following. It can be done by dividing the summation, over all $\mathbb{X} \in T$, in two terms. If the derivative is taken with respect to $\mathbb{W}_i^j$, the first term is all $\mathbb{X}$, such that $C_T(\mathbb{X}) = j$. For the second term, $C_T(\mathbb{X}) \neq j$ holds.

$$logL_r(T,W) = \sum_{i=1}^{|T|} logp(\mathbb{X}_i, y_i\,|\,W) - \sum_{i=1}^{|T|} logp(\mathbb{X}_i, \bar{y}_i\,|\,W)$$

The gradient of the expression above is

$$\frac{\partial}{\partial \mathbb{W}_i^j} logL_r(T,W) = \sum_{\mathbb{X}:C_T(\mathbb{X})=j} \frac{p_i^j}{p(\mathbb{X},y\,|\,W)} \times \frac{\partial p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\partial \mathbb{W}_i^j} - \sum_{\mathbb{X}:C_T(\mathbb{X})\neq j} \frac{p_i^j}{p(\mathbb{X},\bar{y}\,|\,W)} \times \frac{\partial p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\partial \mathbb{W}_i^j}$$

The gradient can be further simplified to get

$$\frac{\partial}{\partial \mathbb{W}_i^j} logL_r(T,W) = \frac{2p_i^j}{\sigma^2} \sum_{\mathbb{X}:C_T(\mathbb{X})=j} \frac{1}{p(\mathbb{X},y\,|\,W)} p(\mathbb{X}\,|\,\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j) - \frac{2p_i^j}{\sigma^2} \sum_{\mathbb{X}:C_T(\mathbb{X})\neq j} \frac{1}{p(\mathbb{X},\bar{y}\,|\,W)} p(\mathbb{X}\,|\,\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)$$

In case of online learning, we have the above relation, simplified to

$$\frac{\partial}{\partial \mathbb{W}_i^j} logL_r(T,W) = \frac{2p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2}\left(\frac{\delta(C_T(\mathbb{X})-j)}{p(\mathbb{X},y\,|\,W)} - \frac{1-\delta(C_T(\mathbb{X})-j)}{p(\mathbb{X},\bar{y}\,|\,W)}\right)(\mathbb{X}-\mathbb{W}_i^j)\ (10)$$

The gradient, given by (9d) and (10), is similar to Learning Vector Quantisation (LVQ) gradient, introduced by T. Kohonen [13], except the updating rates are different for different prototypes. As one can observe, the coefficient of the vector $\mathbb{X} - \mathbb{W}_i^j$, depends on the probability that $\mathbb{X}$ is produced by prototype $\mathbb{W}_i^j$, given we know that $\mathbb{X}$ is produced by a prototype of class j.

A Matlab program for SLVQ function is included in Matlab programs part, under the name program1.

S. Seo and K. Obermayer [1] claimed, in the paper, that the original LVQ [13] methods are based on heuristics. We would like to introduce a cost function that its gradient gives the gradient descent rule that is used in LVQ, introduced by Kohonen. However, this cost function is valid, only if the classification problem is a binary classification. To find the cost function, we would like to be inspired by a similar cost function for unsupervised LVQ that is included in a book, written by H. Ritter et al. [4], in which the cost function, minimised by unsupervised LVQ method, is

$$E = \int ||\mathbb{X} - \mathbb{W}_{S(\mathbb{X})}||^2 p(\mathbb{X}) d\mathbb{X} \text{ (11)}$$

where $\mathbb{W}_{S(\mathbb{X})}$ is the nearest prototype, in W, to $\mathbb{X}$. $p(\mathbb{X})$ is the probability density of the input data space. If one calculates the gradient of equation (11), with respect to W, the averaged updating rule is derived as the following.

$$\mathbb{W}_i^j(t+1) = \mathbb{W}_i^j(t) + 2\epsilon \int_{V(\mathbb{W}_i^j|W)} (\mathbb{X} - \mathbb{W}_{s(\mathbb{X})}) p(\mathbb{X}) d\mathbb{X} \text{ (12)}$$

$V(\mathbb{W}_i^j|W)$ is the Voronoi cell of prototype $\mathbb{W}_i^j$, with respect to W, and $0 < \epsilon \ll 1$ is the learning rate.

Assuming that we would like to do online stochastic learning, equation (12) would be reduced to

$$\mathbb{W}_i^j(t+1) = \mathbb{W}_i^j(t) + 2\epsilon(\mathbb{X} - \mathbb{W}_{S(\mathbb{X})}) = \mathbb{W}_i^j(t) + \epsilon'(\mathbb{X} - \mathbb{W}_{S(\mathbb{X})}) \text{ (13)}$$

which is the simple unsupervised VQ learning method.

For a binary supervised LVQ, with labels $L = \{1, -1\}$, the cost function can be defined as below.

$$E = \int \left(||\mathbb{X} - \mathbb{W}^1_{S(\mathbb{X})}||^2 - ||\mathbb{X} - \mathbb{W}^{-1}_{S(\mathbb{X})}||^2 \right) C_T(\mathbb{X}) p(\mathbb{X}) d\mathbb{X} \text{ (14)}$$

Where $\mathbb{W}^1_{S(\mathbb{X})}$ is the winner prototype, when only the prototypes of class 1 are considered. Similarly, $\mathbb{W}^{-1}_{S(\mathbb{X})}$ is the winner prototype, when prototypes of class -1 are taken into account. $C_T(\mathbb{X})$ is the training function, with respect to T (refer to definitions in section 2). We would like to define it as $C_T : \mathbb{R}^n \rightarrow \{1, -1\}$ such that if $(\mathbb{X}, y) \in T$, then $C_T(\mathbb{X}) = y$. There is a family of functions that are eligible to be $C_T(\mathbb{X})$. As an example, we may consider the Voronoi cells of all the prototypes $\mathbb{W}$ and map a data $\mathbb{X}$ to the label of the closest prototype. In case the data point $\mathbb{X}$ is on the boundary Voronoi cells, then the mapping would be done randomly to 1 or -1. If we make the assumption that $p(\mathbb{X})$ is bounded and countably discontinuous, then the boundaries oft he Voronoi cells have measure zero and the integral, in the cost function is well-defined. Under the mentioned conditions the gradient of the cost function would be

$$\frac{\partial E}{\partial \mathbb{W}_i^1} = 2\int_{V(\mathbb{W}_i^1|W^1)} (\mathbb{X} - \mathbb{W}_i^1)C_T(\mathbb{X})p(\mathbb{X})d\mathbb{X}$$

$$\frac{\partial E}{\partial \mathbb{W}_i^{-1}} = -2\int_{V(\mathbb{W}_i^{-1}|W^{-1})} (\mathbb{X} - \mathbb{W}_i^{-1})C_T(\mathbb{X})p(\mathbb{X})d\mathbb{X}$$

W^1 and W^{-1} contain prototypes $\mathbb{W}_i^1\ \forall i$ and $\mathbb{W}_i^{-1}\ \forall i$, respectively. As in the unsupervised case, the online version of the updating rules, when $(\mathbb{X}, \mathbb{y})$ is given randomly, is as below.

$$\mathbb{W}_{S^+(\mathbb{X})}(t+1) = \mathbb{W}_{S^+(\mathbb{X})}(t) + \mu(\mathbb{X} - \mathbb{W}_{S^+(\mathbb{X})})\ (15a)$$

$$\mathbb{W}_{S^-(\mathbb{X})}(t+1) = \mathbb{W}_{S^-(\mathbb{X})}(t) - \mu(\mathbb{X} - \mathbb{W}_{S^-(\mathbb{X})})\ (15b)$$

$\mathbb{W}_{S^+(\mathbb{X})}$ is the winner prototype, among the prototypes that have the same label as the provided data point $\mathbb{X}$. $\mathbb{W}_{S^-(\mathbb{X})}$ is the winner prototype, among the prototypes that have the opposite label, compared to the provided data point $\mathbb{X}$.

Unfortunately, the author of this article do not know if a similar cost function can be found for a classification problem, with the number of classes greater than 2. In case such function exists, for an arbitrary number of classes, then the argument that LVQ is based on heuristics is not valid.

Regarding the convergence of SLVQ, we would like to point out that, in case of convergence, the prototypes will not converge to the prototypes that are assumed to be responsible for producing the training set T, according to the Gaussian mixture model. To explain it, we may consider an easy problem. Assume a classification problem with two classes + and –, on the real line $\mathbb{R}$. To each class, there is a prototype. $\mathbb{W}^+$, for class + and $\mathbb{W}^-$, for class –. $\mathbb{W}^+$ is placed at $\mathbb{X} = 1$ and $\mathbb{W}^-$ is placed at $\mathbb{X} = -1$. Each prototype has a Gaussian distribution, with $\sigma = 1$. So far, we have described the Gaussian mixture model. Now, we would like to write the dynamical behaviour of $\mathbb{W}^+$. One can easily see that, on average, the change in $\mathbb{W}^+$, as a function of its position, is as the following.

$$\Delta\mathbb{W}^+ = \int_{-\infty}^{+\infty} (\mathbb{X} - \mathbb{W}^+)p^+(\mathbb{X})d\mathbb{X} - \int_{-\infty}^{+\infty} (\mathbb{X} - \mathbb{W}^+)p^-(\mathbb{X})d\mathbb{X}$$

For $\mathbb{W}^-$, it would be

$$\Delta\mathbb{W}^- = \int_{-\infty}^{+\infty} (\mathbb{X} - \mathbb{W}^-)p^+(\mathbb{X})d\mathbb{X} - \int_{-\infty}^{+\infty} (\mathbb{X} - \mathbb{W}^-)p^-(\mathbb{X})d\mathbb{X}$$

$p^+(\mathbb{X})$ is $p(\mathbb{X} \mid 1,1)$, as explained in equation (4), and $p^-(\mathbb{X})$ is $p(\mathbb{X} \mid -1,1)$.

Having a look at the equation above, one can see that the first term of right hand side has a unique fixed point at $\mathbb{W}^+ = 1$ and the second term of has a unique fixed point at $\mathbb{W}^+ = -1$. It simply means that $\mathbb{W}^+ = 1$ cannot be a fixed point of both of them together. Consequently, it is proven that prototypes would not settle at the assumed positions, as the Gaussian mixture model. However, one may argue that SLVQ is a Bayesian classifier, in which the probability of misclassification is minimised. Therefore, we do not care if the prototypes approach the positions, in $\mathbb{R}^n$, where they are supposed to be, as suggested by a Gaussian mixture model.

Regardless of the issues mentioned in previous paragraphs, SLVQ has a nice mathematical framework that makes a nice platform to conduct further investigation on. As an example, when studying the dynamics of traditional LVQ, Voronoi cells of prototypes are defined and Voronoi cell of a prototype $\mathbb{W}_i^j$, created by a set of prototypes W, depends on the position of other prototypes. This makes the dynamics of LVQ to be a hard problem to treat. The problem would vanish in SLVQ, as it follows a different learning scheme.

3-3- SLVQ variant with attraction (no repulsion)

In this section, a new way of looking at the original SLVQ method is introduced. The new variant results in an updating rule that applies only attraction. If instead of $p(\mathbb{X}_i, y_i \mid W)$ and $p(\mathbb{X}_i, \bar{y}_i \mid W)$ their normalised quantities are used, then some interesting results may be achieved. The material, provided in this section, can be found in an article Written by M. Mohannazadeh and T. Villmann [12].
We would like to define the quantities below.

$$p(\mathbb{W}^y \mid \mathbb{X}) = \frac{1}{p(\mathbb{X})} \sum_t p(\mathbb{X} \mid \mathbb{W}_t^y) . p_t^y \ (16a)$$

$$p(\mathbb{W}^{\bar{y}} \mid \mathbb{X}) = \frac{1}{p(\mathbb{X})} \sum_{j \neq y} \sum_t p(\mathbb{X} \mid \mathbb{W}_t^j) . p_t^j \ (16b)$$

Given that $\mathbb{X}$ has happened, $p(\mathbb{W}^y \mid \mathbb{X})$ is the probability that data point $\mathbb{X}$ is produced by a prototype of class y. In a similar way, given that $\mathbb{X}$ has happened, $p(\mathbb{W}^{\bar{y}} \mid \mathbb{X})$ is the probability that data point $\mathbb{X}$ is produced by a prototype of a class different from y. Note that $p(\mathbb{X}_i, y_i \mid W)$ becomes equal to $p(\mathbb{W}^y \mid \mathbb{X})$, when it is divided by $p(\mathbb{X})$. The same is true for $p(\mathbb{X}_i, \bar{y}_i \mid W)$and $p(\mathbb{W}^{\bar{y}} \mid \mathbb{X})$.

The terms, defined in equations (16a) and (16b), are used to define a new maximum likelihood ratio function.

$$L_r(T, W) = \prod_{i=1}^{|T|} \frac{p(\mathbb{W}^{y_i} \mid \mathbb{X}_i)}{p(\mathbb{W}^{\bar{y}_i} \mid \mathbb{X}_i)} = \prod_{i=1}^{|T|} \frac{p(\mathbb{W}^{y_i} \mid \mathbb{X}_i)}{1 - p(\mathbb{W}^{y_i} \mid \mathbb{X}_i)} \ (17a)$$

Note that there is no difference between the function in (17) and the one given in (5), as the factor $1/p(\mathbb{X})$ appears in both the nominator and the denominator of (17).

Despite the fact that cost functions (5) and (17) are the same, they result in different updating rules. The difference arises from the quantities that are used in the cost functions. $P\left(\mathbb{W}^{y_i} \mid \mathbb{X}_i\right)$ and $P\left(\mathbb{W}^{\bar{y}_i} \mid \mathbb{X}_i\right)$ add up to one and, consequently, they make full probabilistic model. However, we cannot make the same conclusion for quantities $p(\mathbb{X}_i, y_i \mid W)$ and $p(\mathbb{X}_i, \bar{y}_i \mid W)$, as they do not add up to 1. In other words, with the probabilities, defined in equation (5), we may lose information as the learning phase is performed.

The partial likelihood function, based on (17),is

$$L_p((\mathbb{X}, y), W) = \frac{p(\mathbb{W}^y | \mathbb{X})}{1 - p(\mathbb{W}^y | \mathbb{X})} \ (17b)$$

In case of online learning (sample $(\mathbb{X}, y)$), when the logarithm of the function $L_p((\mathbb{X}, y), W)$ is used, the gradient is

$$SGAL(L_p, (\mathbb{X}, y), \mathbb{W}_i^y) = \frac{\partial log L_p(\mathbb{X}, W)}{\partial p(\mathbb{X} | \mathbb{W}_i^y)} \cdot \frac{\partial p(\mathbb{X} | \mathbb{W}_i^y)}{\partial d(\mathbb{X}, \mathbb{W}_i^y)} \cdot \frac{\partial d(\mathbb{X}, \mathbb{W}_i^y)}{\partial \mathbb{W}_i^y} \ (18a)$$

In (18a), the partial derivatives are as below.

If (16a) and (16b) are used

$$\frac{\partial log L_p(T, W)}{\partial p(\mathbb{X} | \mathbb{W}_i^y)} = \frac{p_i^y}{p(\mathbb{X})} \cdot \left(\frac{1}{p(\mathbb{W}^y | \mathbb{X})} + \frac{1}{1 - p(\mathbb{W}^y | \mathbb{X})}\right) \ (18b)$$

If the distribution is Gaussian, as defined in (3)

$$\frac{\partial p(\mathbb{X} | \mathbb{W}_i^y)}{\partial d(\mathbb{X}, \mathbb{W}_i^y)} = -\frac{p(\mathbb{X} | \mathbb{W}_i^y)}{\sigma^2} \cdot d(\mathbb{X}, \mathbb{W}_i^j) \ (18c).$$

If the distance measurement is Euclidean

$$\frac{\partial d(\mathbb{X}, \mathbb{W}_i^y)}{\partial \mathbb{W}_i^y} = -\frac{1}{d(\mathbb{X}, \mathbb{W}_i^y)} \cdot (\mathbb{X} - \mathbb{W}_i^y) \ (18d).$$

$p(\mathbb{X})$ is assumed to be constant and known.

In the new version, only the prototypes of the class y (same class as the presented data point $\mathbb{X}$) are updated.
A Matlab program for ASLVQ function is included in Matlab programs part, under the name program2.

Another approach, to deal with the problem of the dependence of prototypes in SLVQ model, is to use Lagrange multipliers optimisation. One can see that, in likelihood function (5), some prototypes do not have a full degree of freedom. Those prototypes are the prototypes that are responsible for a different class, with respect to a data point $\mathbb{X}$, that is given at time t. The equation that makes the prototypes dependent is

$$p(\mathbb{X}, y | W) + p(\mathbb{X}, \bar{y} | W) = \sum_t p_t^y p(\mathbb{X} | \mathbb{W}_t^y) + \sum_{j \neq y} \sum_t p_t^j P(\mathbb{X} | \mathbb{W}_t^j) = p(\mathbb{X}) \ (20)$$

In (20), it is assumed that $p(\mathbb{X})$ is known, at least for data points $\mathbb{X}$ that are in the training set T. In (16), on the contrary, $p(\mathbb{X})$ is not required to be known and it is estimated using formula (19).

Now, to apply gradient ascent learning, there are two options. One way is to calculate the gradient, with respect to the independent variables (prototypes of class y, when $(\mathbb{X}, y)$ is provided). This is exactly ASLVQ, which is explained earlier. In the second way, we can take all the prototypes into account. The authors of the article [12] are under the impression that if one intends to keep all the prototypes in the likelihood functions, then the optimisation problem is a constraint optimisation problem and, therefore, the cost function should be modified to reflect the fact. Therefore, a new function, using the concept of Lagrange multipliers, is defined as the following. Based on function (5)

$$F(\mathbb{X}, W) = log\frac{p(\mathbb{X}, y \mid W)}{p(\mathbb{X}, \bar{y} \mid W)} + \lambda\big(p(\mathbb{X}, y \mid W) + p(\mathbb{X}, \bar{y} \mid W) - p(\mathbb{X})\big) \quad (21)$$

The gradient, with respect to W, is

$$\frac{\partial F(\mathbb{X}, W)}{\partial \mathbb{W}_i^j} = \frac{p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{\sigma^2}\left(\frac{\delta(C(\mathbb{X}) - j)}{p(\mathbb{X}, y \mid W)} - \frac{1 - \delta(C(\mathbb{X}) - j)}{p(\mathbb{X}, \bar{y} \mid W)} + \lambda\right)(\mathbb{X} - \mathbb{W}_i^j)$$

The gradient, with respect to λ, is simply the constraint

$$p(\mathbb{X}, y \mid W) + p(\mathbb{X}, \bar{y} \mid W) = p(\mathbb{X})$$

which can be plugged in the previous formula to achieve the updating rule below.

$$\frac{\partial F(\mathbb{X}, W)}{\partial \mathbb{W}_i^j} = \frac{p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{\sigma^2}\left(\frac{\delta(y - j)}{p(\mathbb{X}, y \mid W)} - \frac{1 - \delta(y - j)}{p(\mathbb{X}) - p(\mathbb{X}, y \mid W)} + \lambda\right)(\mathbb{X} - \mathbb{W}_i^j) \quad (22)$$

Finally, two remarks are made. Firstly, the constant $p(\mathbb{X})$, which should be found in each learning step, is not the same as relation (19), for if the relation (19) is used, the penalty function in relation (21) would always be equal to zero. Therefore, as a drawback of this method, we are required to have an estimation of the distribution of the data space, that is to be classified.

Secondly, note that the new set of adaptive parameters is $W \cup \lambda$ and λ would be initiated randomly.

The updating model (22) becomes more accurate if a special λ is dedicated to each class j, because, according to the class y of data point $\mathbb{X}$, the function $F(\mathbb{X}, W)$ is different. Hence, we may define λ^j for a class j. Consequently, the λ, in gradient (22), is substituted by λ^y to have the new gradient.

$$\frac{\partial F(\mathbb{X}, W)}{\partial \mathbb{W}_i^j} = \frac{p_i^j p(\mathbb{X} | \mathbb{W}_i^j)}{\sigma^2} \left(\frac{\delta(y-j)}{p(\mathbb{X}, y | W)} - \frac{1-\delta(y-j)}{p(\mathbb{X}) - p(\mathbb{X}, y | W)} + \lambda^y \right) (\mathbb{X} - \mathbb{W}_i^j) \quad (23)$$

Relation (23) suggests to have multiple online functions to maximise, as the main function itself varies, when data points $\mathbb{X}$, with different classes, are provided to the system.

The last unanswered question, in this section, is "how to estimate the distribution $p(\mathbb{X})$, for LSLVQ". If we do not have a prior knowledge of the distribution $p(\mathbb{X})$, then a solution might be to perform an unsupervised density estimation (for example unsupervised LVQ or Neural Gas), parallel to the main classification problem, in order to find an estimation of $p(\mathbb{X})$. Note that the estimation improves as the number of learning steps increases.

The rest of this section gives experimental results on ASLVQ and LSLVQ. A binary classification problem, which is called the butterfly problem, is given to two programs, one using SLVQ and the other one using ASLVQ algorithm. The butterfly problem is as follows.
On $\mathbb{R}^2$ plane, a point $x = (x_1, x_2)$ is of class 1, iff $x_1 \times x_2 \geq 0$ and it is of class 2 otherwise.

The Matlab program, to produce a training set, a test set and initiate prototypes for the butterfly problem, is include in "Matlab Program" part of this article, under the name "program 3".

The reason why the butterfly program is chosen is that, with Euclidian distance measurement, we cannot actually achieve an appropriate classification, for reasonable number of prototypes.

For instance, the problem of classifying hand written characters was given to both SLVQ and ASLVQ and no improvements in the classification was achieved. Therefore, the butterfly problem serves as a nice problem to compare SLVQ and ASLVQ.

After one million learning steps, having 20 prototypes per class, the misclassification error rates are 0.178 and 0.057, for SLVQ and ASLVQ, respectively. The initial error rate (before learning) was 0.605. Also, the standard deviation was chosen to be 1.

Note that the error rates highly depend on the initialisation on prototypes, which is done totally randomly. Therefore, the error rates are compared relatively. In this thesis, the author tries to avoid initiating prototypes in desirable regions, as done by some researchers, for we believe that in real world problems it is rather impractical.

It is emphasised that ASLVQ successfully achieved a less error rate, compared to SLVQ. The practical result backs up the mathematical theory of ASLVQ.

For LSLVQ, although the theoretical ground was laid, the author of this thesis was unable to find a way to make the theory work in practice. The instability of the method was also mentioned by a few colleagues.

3-4- Robust soft learning vector quantisation (RSLVQ)

It is explicitly mentioned by S. Seo and K. Obermayer [1] that RSLVQ has been proposed to deal with the stability problem of SLVQ. In other words, RSLVQ is the stable version of SLVQ. For RSLVQ, a new cost function is defined as below.

$$L_r(T, W) = \prod_{i=1}^{|T|} \frac{p(\mathbb{X}_i, y_i \,|\, W)}{p(\mathbb{X}_i, y_i \,|\, W) + p(\mathbb{X}_i, \bar{y}_i \,|\, W)} \quad (24a)$$

The cost function can be explained as the following. We need to look at partial cost functions that are defined for each training data point $(\mathbb{X}_i, y_i)$, i.e.

$$\frac{p(\mathbb{X}_i, y_i \,|\, W)}{p(\mathbb{X}_i, y_i \,|\, W) + p(\mathbb{X}_i, \bar{y}_i \,|\, W)} \quad (24b)$$

The maximum that it can achieve is limited, whole it is not the case for

$$\frac{p(\mathbb{X}_i, y_i \,|\, W)}{p(\mathbb{X}_i, \bar{y}_i \,|\, W)}$$

as it can achieve infinity. Now, one needs to imagine that the cost function of SLVQ has as many as $|T|$ points in W space, for which the value of the cost function is ∞. In case W is initialised near any of these points, it would automatically be sucked into a wrong region and the regions contain prototypes W that cause the scenario that all the prototypes, with a certain class, are located at a single point, while the rest are pushed to infinity. This effect can be reduced by limiting the maximum, each partial cost function can achieve (as in RSLVQ). Consequently, even if the prototypes are initialised in a wrong region, there is still a chance for them to get back in the right region, as the cost function is rather smoother. The updating rule of RSLVQ, with respect to the new cost function (24), is as the following.

$$\mathbb{W}_i^j(t+1) = \mathbb{W}_i^j(t) + \mu \frac{\partial}{\partial \mathbb{W}_i^j} log L_r(T, W) \ (25a)$$

With $L_r(T, W)$ defined as in (24b). The elaboration of the partial derivative of (25a)is as below.

$$\frac{\partial}{\partial \mathbb{W}_i^j} log L_r(T, W) = \frac{\partial}{\partial \mathbb{W}_i^j} log p(\mathbb{X}, y \,|\, W) - \frac{\partial}{\partial \mathbb{W}_i^j} log \left(p(\mathbb{X}, \bar{y} \,|\, W) + p(\mathbb{X}, y \,|\, W) \right) \ (25b)$$

The first and the second terms of the right hand side of (25b) are as below, respectively.

$$\frac{\partial}{\partial \mathbb{W}_i^j} log p(\mathbb{X}, y \,|\, W) = \frac{1}{p(\mathbb{X}, y \,|\, W)} \cdot \frac{\partial}{\partial \mathbb{W}_i^j} p(\mathbb{X}, y \,|\, W) \ (25c)$$

$$\frac{\partial}{\partial \mathbb{W}_i^j} log \left(p(\mathbb{X}, \bar{y} \,|\, W) + p(\mathbb{X}, y \,|\, W) \right) = \frac{1}{p(\mathbb{X}, \bar{y} \,|\, W) + p(\mathbb{X}, y \,|\, W)} \cdot \frac{\partial}{\partial \mathbb{W}_i^j} \left(p(\mathbb{X}, \bar{y} \,|\, W) + p(\mathbb{X}, y \,|\, W) \right) \ (25d)$$

Using chain rule, the following is obtained. Again, $d(\mathbb{X}, \mathbb{Y})$ is a general dissimilarity measurement.

$$\frac{\partial}{\partial \mathbb{W}_i^j} p(\mathbb{X}, y \mid W) = \frac{\partial}{\partial d(\mathbb{X}, \mathbb{W}_i^j)} p(\mathbb{X}, y \mid W) \, . \frac{\partial}{\partial \mathbb{W}_i^j} d(\mathbb{X}, \mathbb{W}_i^j) \quad (25e)$$

$$\frac{\partial}{\partial \mathbb{W}_i^j} \Big(p(\mathbb{X}, \bar{y} \mid W) + p(\mathbb{X}, y \mid W) \Big) = \frac{\partial}{\partial d(\mathbb{X}, \mathbb{W}_i^j)} \Big(p(\mathbb{X}, \bar{y} \mid W) + p(\mathbb{X}, y \mid W) \Big) \, . \frac{\partial}{\partial \mathbb{W}_i^j} d(\mathbb{X}, \mathbb{W}_i^j) \quad (25f)$$

In case the Euclidean distance is chosen for $d(\mathbb{X}, \mathbb{W}_i^j)$ and $p(\mathbb{X}, y \mid W)$ and $p(\mathbb{X}, \bar{y} \mid W)$ are as described in formulas (6) and (7), equation (9c) becomes

$$\frac{\partial}{\partial \mathbb{W}_i^j} log \frac{p(\mathbb{X}, y \mid W)}{p(\mathbb{X}, \bar{y} \mid W)} = \frac{p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{\sigma^2} \left(\frac{\delta(j-y)}{p(\mathbb{X}, y \mid W)} - \frac{1}{p(\mathbb{X}, y \mid W) + p(\mathbb{X}, \bar{y} \mid W)} \right) (\mathbb{X} - \mathbb{W}_i^j) \quad (25g)$$

The corresponding Matlab program is included in the programs section. One can notice that, using butterfly problem, RSLVQ works notably better than SLVQ. However, its performance is still worse than ASLVQ. This conclusion does not necessarily hold for other classification problems.

4- Tangent distances

It is often the case, in application, that Euclidean distance (denoted by d_E) of two points, in $\mathbb{R}^n$ space, does not give an appropriate dissimilarity measurement, according to a certain classification task. A common example is the classification of optical characters. A certain character and its rotation are supposed to be classified in the same class. It means that if we are given a dissimilarity function, then it should return a negligible dissimilarity, when a character and its rotation are passed to the function. However, mostly this is not the case, when using Euclidean distance, as a dissimilarity function, for Euclidean distance compares vectors component-wise. As a solution, the idea of tangent distance was offered by P. Simard et.al [2] and elaborated by S. Saralajew and T. Villmann [3], in recent years. Tangent distance and tangent learning are computationally efficient simplifications of a more general idea, that is briefly introduced here.

We are given a space $\mathbb{R}^n$. A point (object) $\mathbb{X} \in \mathbb{R}^n$ is assumed to be a representative of a $\mathbb{R}^m$ $(m < n)$ dimensional manifold. It is assumed that the manifold of $\mathbb{X} \in \mathbb{R}^n$ is produced according to a function $M : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n, M(\mathbb{X}, \theta)$, where $\theta \in \mathbb{R}^m$ is a parameter (can be varied like a knob). Also, we would like to define $M(\mathbb{X}, 0) = \mathbb{X}$.

If $\mathbb{Y} \in \mathbb{R}^n$, to explain $\mathbb{Y} = M(\mathbb{X}, \theta)$ in a simple way, the function takes an object $\mathbb{X}$and a parameter θ and outputs another object $\mathbb{Y}$, that is basically the same object as the first one, from the classification problem point of view ($\mathbb{Y}$and $\mathbb{X}$ are not necessarily equal in $\mathbb{R}^n$). We may assume that the manifold makes equivalence classes on $\mathbb{R}^n$, which means $\mathbb{X} \sim \mathbb{Y}$if, and only if, $\exists \theta : \mathbb{Y} = M(\mathbb{X}, \theta)$. If there is a dissimilarity measure $d_M(\mathbb{X}, \mathbb{Y})$ is defined, according to the new model, then, ideally, $d_M(\mathbb{X}, \mathbb{Y}) = 0$ iff $\mathbb{X} \sim \mathbb{Y}$. This may not be always possible in practice. For instance, as mentioned by P. Simard et.al [4], optical characters "6" and "9" are not to be classified in the same category. However, if the rotation of an object is the same as the object, then, the two characters would be categorised in the same class.

With the equivalence classes of data in $\mathbb{R}^n$, one can use Hausdorff distance [5]$d_H(\mathbb{X}, \mathbb{Y})$ to find the level of dissimilarity between two data points $\mathbb{X}$ and $\mathbb{Y}$.

It might be possible to make some simplifying assumptions on the function $M(\mathbb{X}, \theta)$. It can be thought of a composition of functions $M_i^\theta(\mathbb{X}), i \in \{1,2,...,t\}$. $t \in \mathbb{N}$ and $\theta \in \mathbb{R}$. Functions $M_i^\theta : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ would be called basic functions. Each of the basic functions corresponds to a particular transformation that preserves the class of an object. For instance, rotation and thickening in optical characters. The mathematical expression for combining the basic functions is as below.

$$M(\mathbb{X}, \theta) = M_{i_t}^{\theta_t} \circ \ldots \circ M_{i_2}^{\theta_2} \circ M_{i_1}^{\theta_1}(\mathbb{X}) \ (26)$$

i_j, for $j = \{1,2,...,t\}$, is a reordering of numbers 1 to t. θ_i denotes the i-th elements of the vector $\theta \in \mathbb{R}^m$. The notation i_j is used, in the above relation, to emphasise on the assumption that the order of composing the basic functions do not matter, in a sense that the output of $M(\mathbb{X}, \theta)$ always satisfies $M(\mathbb{X}, \theta) \sim \mathbb{X}$. With different orders one may get different outputs (different functions $M(\mathbb{X}, \theta)$). Also notice that basic functions are not necessarily commutative. Nonetheless, it is assumed that different functions, achieved by composing the basic functions in different orders, give the same manifold, as long as their inputs are from the same class. Therefore, the basic functions are composed, in an arbitrary order, and would be kept fixed for the rest of the analysis and application.

Although the general theory, explained in previous paragraphs, is mathematically accurate, it is impractical to be implemented in machine learning systems, as there is no analytical expression for the distance, in general. A solution is to find the tangent hyper-plane, induced by the function $M(\mathbb{X}, \theta)$, around the point $\mathbb{X}_0$. If $M(\mathbb{X}, \theta)$ is differentiable, with respect to θ, then the affine is the partial derivative of $M(\mathbb{X}, \theta)$, with respect to θ, evaluated at $\mathbb{X}_0$ and $\theta = 0$, which is a $n \times m$ matrix $A_{\mathbb{X}_0}$.

$$A_{\mathbb{X}_0} = \left.\frac{\partial M(\mathbb{X}, \theta)}{\partial \theta}\right|_{\theta=0, \mathbb{X}=\mathbb{X}_0} \quad (27)$$

$\mathbb{X}_1 = \mathbb{X}_0 + A_{\mathbb{X}_0}\theta$, when $||\theta|| \approx 0$, gives an object $\mathbb{X}_1$ that is supposed to be in the same class as $\mathbb{X}_0$.

If $M(\mathbb{X}_0, \theta)$ is written as a composition of basic functions, then the k-th column of $A_{\mathbb{X}_0}$, denoted by $(A_{\mathbb{X}_0})_k$, can be expressed as below.

$$(A_{\mathbb{X}_0})_k = \left.\left[\left(\prod_{d=k+1}^{m} \frac{\partial M_{i_d}^{\theta}}{\partial \mathbb{X}}\right) \times \frac{\partial M_{i_k}^{\theta}}{\partial \theta_k}\right]\right|_{\theta=0, \mathbb{X}=\mathbb{X}_0} \quad (28)$$

There might be a need to take the differential of $A_{\mathbb{X}}$, with respect to $\mathbb{X}$. As $A_{\mathbb{X}}$ is a matrix function, we may put it into vector form, in order to calculate its differential.

$$\frac{\partial vec(A_{\mathbb{X}})}{\partial \mathbb{X}}$$

At this point, there are two possibilities to define a distance, using the tangent space approximation of a manifold. The first, used by P. Simard et.al[4], is the double-sided tangent distance. If one needs to find $d(\mathbb{X}, \mathbb{Y})$, the tangent spaces of $\mathbb{X}$ and $\mathbb{Y}$, that are $\mathbb{X} + A_{\mathbb{X}}\theta$ and $\mathbb{Y} + A_{\mathbb{Y}}\gamma$ respectively, are found. Then, the distance is defined as below.

$$d_d(\mathbb{X}, \mathbb{Y}) = inf_{\theta,\gamma}\{d_E(\mathbb{X} + A_{\mathbb{X}}\theta, \mathbb{Y} + A_{\mathbb{Y}}\gamma)\} \quad (29)$$

The parameters θ and γ, for which the Euclidian distance is minimised, can be calculated and, hence, $d(\mathbb{X}, \mathbb{Y})$ is computable. This version of tangent distance gives a fair measure of dissimilarity. As a remark, double-sided distance is not a metric, for it violates triangular inequality property of a metric on a metric space.

The second variation of tangent distance is called single-sided tangent distance. Although one may assume, theoretically, that single-sided tangent distance is less accurate than the double-sided version, it has been shown by D.M. Keysers, in his PhD thesis [11], that single-sided tangent works as good as the double sided version, in practice. Consequently single-sided tangent distance is preferred over the double-sided variation, since it gives us an analytical solution for the distance between a point $\mathbb{X} \in \mathbb{R}^n$ and a tangent space in $\mathbb{R}^n$. In order to define the single-sided distance $d_s(\mathbb{X}, \mathbb{Y})$, the tangent plane $\mathbb{X} + A_{\mathbb{X}}\theta$ of the first argument is considered. Having that, one can define the single-sided distance as below.

$$d_s(\mathbb{X}, \mathbb{Y}) = inf_\theta\{d_E(\mathbb{X} + A_{\mathbb{X}}\theta, \mathbb{Y})\} \ (30)$$

As mentioned by S. Saralajew and T. Villmann [3], θ, for which $d_E(\mathbb{X} + A_{\mathbb{X}}\theta, \mathbb{Y})$ is minimised, is as below.

$$\theta* = A_{\mathbb{X}}^T(\mathbb{Y} - \mathbb{X}) \ (31)$$

An obvious consequence of the transition, from double-sided to single-sided, is that we lose symmetry of the distance, i.e. $\exists \mathbb{X}, \mathbb{Y} : d_s(\mathbb{X}, \mathbb{Y}) \neq d_s(\mathbb{Y}, \mathbb{X})$.

A rather obvious property of both double-sided and single-sided tangent distance is

$$d_s(\mathbb{X}, \mathbb{Y}) \leq d_E(\mathbb{X}, \mathbb{Y})$$

$$d_d(\mathbb{X}, \mathbb{Y}) \leq d_E(\mathbb{X}, \mathbb{Y})$$

5- Integrating tangent distances into SLVQ and RSLVQ

In this section, tangent distances would be used as the dissimilarity measurement and they are supposed to substitute the Euclidean distance. The integration of tangent distances into SLVQ and RSLVQ is not an straightforward problem, however. We are going to have to make assumptions for different scenarios and, also, to make necessary simplifications, so there would be a learning rule after all. A key assumption, throughout this section, is that we assume the manifolds, defined in the data space of the learning problem, are always differentiable.

5-1- SLVQ, using tangent distances (Simard Assumption)

Let's assume a specific classification problem in $\mathbb{R}^n$, for which Euclidean distance does not offer an appropriate dissimilarity measurement, when comparing two objects $\mathbb{X}, \mathbb{Y} \in \mathbb{R}^n$. We would like to use a version of tangent distance instead. To do so, one needs to introduce a set of transformations, denoted by $M_i^\theta(\mathbb{X})$ in the previous section, that describe variations of an object, in data space. Note that, assuming that we are given the class preserving transformations, we are adapting the idea that was developed by P. Simard et.al [2]. This variant is explained in this section. In the next section, it would be assumed that the class preserving transforms are not known. Therefore, a learning method should locally approximate the transformations. This is exactly what is assumed by S. Saralajew and T. Villmann [3],when applying tangent distances to classification problems.

The transformations are composed to have $M(\mathbb{X}, \theta)$. Then, whenever $d_d(\mathbb{X}, \mathbb{Y})$ or $d_s(\mathbb{X}, \mathbb{Y})$ is required, the local tangent spaces (gradient of $M(\mathbb{X}, \theta)$, with respect to θ) are found and the formulas (27) and (28) are used. Now, in order to learn the classification problem, according to the training set T, SLVQ is utilised. However, this time, we would like to use tangent distances instead of the Euclidean distance. The procedure is explained in more detail, in the next paragraphs.

Remark: Note that $M(\mathbb{X}, \theta)$ is, to a high extent, what it needed to clarify the very first sentence of the previous paragraph, which contains "a specific classification problem"). Also, having $M(\mathbb{X}, \theta)$, a considerable amount of information is added to the original training set T. This is the reason for the result of this approach to be much more reliable (compared to normal SLVQ), as the amount of information, contained in the training set, is rather not comparable to the case where we only have the training set T.

Having a look at the introduction on SLVQ in this thesis, one may start to imagine how the Gaussian mixture model produces data samples, when the distance is tangent distance. Let's imagine a component $\mathbb{W}$ of the Gaussian mixture. If the tangent space at $\mathbb{W}$, given by $M(\mathbb{W}, \theta)$, is shifted by $-\mathbb{W}$, we get the linear space $A_\mathbb{W}\theta$. $A_\mathbb{W}$ is a $n \times r$ matrix, where r is the dimensions of the parameter space (recall $M : \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^n$). The complement space of $A_\mathbb{W}\theta$ can be created by a $n \times (n - r)$ matrix $A^c_\mathbb{W}$, such that column vectors of $A_\mathbb{W}$ and $A^c_\mathbb{W}$ are mutually orthogonal. Roughly speaking, we would like to have a Gaussian distribution on each of the spaces (tangent space and the complement space), centred at $\mathbb{W}$, while they have different standard deviations. The standard deviation of the complement space should be much smaller, in comparison. Therefore, component $\mathbb{W}$ produces points, on $\mathbb{R}^n$, according to the following formula.

$$\mathbb{W} + A_{\mathbb{W}} N_{\mathbb{R}^r}(0,\sigma_1) + A^c_{\mathbb{W}} N_{\mathbb{R}^{n-r}}(0,\sigma_2) \ , \ \sigma_1 \gg \sigma_2 > 0 \ (31)$$

$N_{\mathbb{R}^r}(0,\sigma_1)$ is a random point in $\mathbb{R}^r$ (refer to section 2), that follows a Gaussian distribution.

As the first attempt, we would like to combine SLVQ and single-sided tangent distance, by substituting the Euclidean distance with single-sided tangent distance and its orthogonal complement component. This approach, in which the distance on the tangent space is also taken into account, is called the strong approach. As before, the function, to be maximised, is equation (5).

$$L_r(T, W) = \prod_{i=1}^{|T|} \frac{p(\mathbb{X}_i, y_i \mid W)}{p(\mathbb{X}_i, \bar{y}_i \mid W)}$$

with the terms in the nominator and the denominator defined in relations (6) and (7).

So far there has been no differences, regarding the symbols. The difference becomes clear as soon as one writes $p(\mathbb{X} \mid \mathbb{W}^j_i)$, considering single-sided tangent distance.

$$p(\mathbb{X} \mid \mathbb{W}^j_i) = \frac{1}{\sqrt{2\pi\sigma_1^2}} exp\left(- \frac{||A(A^TA)^{-1}A^T(\mathbb{X} - \mathbb{W}^j_i)||^2}{2\sigma_1^2} \right)$$

$$\times \frac{1}{\sqrt{2\pi\sigma_2^2}} exp\left(- \frac{||A^c((A^c)^TA^c)^{-1}(A^c)^T(\mathbb{X} - \mathbb{W}^j_i)||^2}{2\sigma_2^2} \right) \ (32a)$$

The relation (32a) can be read as the following. The probability that the data point $\mathbb{X}$ is produced, given the producer prototype is $\mathbb{W}^j_i$, is the probability that the component of the random vector on the tangent space is $A(A^TA)^{-1}A^T(\mathbb{X} - \mathbb{W}^j_i)$ and the component of the random vector on the complement space is$A^c((A^c)^TA^c)^{-1}(A^c)^T(\mathbb{X} - \mathbb{W}^j_i)$.

After simplifications, we achieve

$$p(\mathbb{X}\,|\,\mathbb{W}_i^j) = \frac{1}{2\pi\sigma_1\sigma_2} exp\left(-\frac{||AA^T(\mathbb{X} - \mathbb{W}_i^j)||^2}{2\sigma_1^2} - \frac{||A^c(A^c)^T(\mathbb{X} - \mathbb{W}_i^j)||^2}{2\sigma_2^2} \right) \quad (32b)$$

In the relations (32a) and (32b), for brevity, A is used to indicate $A_{\mathbb{W}_i^j}$ and A^c to indicate $A^c_{\mathbb{W}_i^j}$. In order to understand relation (32), firstly, we would like to mention the assumption that choosing a random point on $\mathbb{R}^r$ (r is the dimensions of the tangent space), with distribution $N_{\mathbb{R}^r}(0,\sigma_1)$ on the tangent space and choosing another random point on $\mathbb{R}^{n-r}$, with distribution $N_{\mathbb{R}^{n-r}}(0,\sigma_2)$ on the complement space, are independence events. Therefore, one can see the product of Gaussian terms in equation (32a). Note that $p(\mathbb{X}, \mathbb{W}_i^j)$ is written as the product of two Gaussian random variables, for the spaces A and A^c are orthogonal spaces and they make a basis for $\mathbb{R}^n$. Secondly, the arguments of the exponential functions in relations (32a) and (32b) are explained as the following.

$||A(A^TA)^{-1}A^T(\mathbb{X} - \mathbb{W}_i^j)||$ is basically the projection of the vector $(\mathbb{X} - \mathbb{W}_i^j)$ on the tangent space $A_{\mathbb{W}_i^j}$. $||A^c((A^c)^TA^c)^{-1}(A^c)^T(\mathbb{X} - \mathbb{W}_i^j)||$ is the projection of $(\mathbb{X} - \mathbb{W}_i^j)$ on the complement space $A^c_{\mathbb{W}_i^j}$. For a more information, the reader is referred to [3].

Another point is that in case $\sigma_1 = \sigma_2$, we get the same result as when we use the Euclidean distance (original SLVQ). Therefore, σ_1 and σ_2 put weights on the terms (in the argument of the exponential function) in (32b) (one in the tangent space, the other one in the complement space), when they add up.

If the logarithmic version of the maximum likelihood function is used and the learning scheme is online, given a random input $\mathbb{X}$, then the gradient would be

$$\frac{\partial}{\partial \mathbb{W}_i^j} log \frac{p(\mathbb{X}, y \mid W)}{p(\mathbb{X}, \bar{y} \mid W)} = \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W) - \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W) \ (33a)$$

where

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W) = \frac{\delta(j-y) p_i^j}{p(\mathbb{X}, y \mid W)} \times \frac{\partial p(\mathbb{X} \mid \mathbb{W}_i^j)}{\partial \mathbb{W}_i^j} \ (33b)$$

and

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W) = \frac{(1 - \delta(j-y)) p_i^j}{p(\mathbb{X}, \bar{y} \mid W)} \times \frac{\partial p(\mathbb{X} \mid \mathbb{W}_i^j)}{\partial \mathbb{W}_i^j} \ (33c)$$

Finally, the derivative of $p(\mathbb{X} \mid \mathbb{W}_i^j)$, with respect to $\mathbb{W}_i^j$ is required.

$$\frac{\partial p(\mathbb{X} \mid \mathbb{W}_i^j)}{\partial \mathbb{W}_i^j} = p(\mathbb{X} \mid \mathbb{W}_i^j) \left(-\frac{AA^T(\mathbb{X} - \mathbb{W}_i^j) U_1}{\sigma_1^2} - \frac{A^c (A^c)^T (\mathbb{X} - \mathbb{W}_i^j) U_2}{\sigma_2^2} \right) \ (33d)$$

with

$$U_1 = \frac{\partial A}{\partial \mathbb{W}_i^j} A^T (\mathbb{X} - \mathbb{W}_i^j) + A \frac{\partial A^T}{\partial \mathbb{W}_i^j} (\mathbb{X} - \mathbb{W}_i^j) - AA^T \ (33e)$$

$$U_2 = \frac{\partial A^c}{\partial \mathbb{W}_i^j} (A^c)^T (\mathbb{X} - \mathbb{W}_i^j) + A^c \frac{\partial (A^c)^T}{\partial \mathbb{W}_i^j} (\mathbb{X} - \mathbb{W}_i^j) - (A^c)(A^c)^T \ (33f)$$

At this point, an argument would be made to show the reason that the author thinks the strong approach is theoretically more valid than the weak approach, in which only the tangent distance mistaken into account (the distance on the complement space is ignored).

In basic SLVQ, a Gaussian mixture model is assumed to be responsible for data distribution on the data space. So each Gaussian component, which is a vector, produces other vectors, on the same space, according to the Gaussian distribution. With the same reasoning, one may attempt to define a space $\mathbb{W} + A_{\mathbb{W}}\theta$ to be a prototype. So, a prototype is a space now, rather than a vector. Then, the prototype $\mathbb{W} + A_{\mathbb{W}}\theta$ is supposed to produce a data point $\mathbb{X} \in \mathbb{R}^n$ of the same class, according to a Gaussian distribution. We know that $\mathbb{X}$ should be located on one of the affine spaces, that are parallel to $\mathbb{W} + A_{\mathbb{W}}\theta$. The probability of any of the affine spaces to be the one, on which $\mathbb{X}$ is located is also known to be

$$\frac{1}{\sqrt{2\pi\sigma_2^2}} exp\left(\frac{||A^c(A^c)^T(\mathbb{X} - \mathbb{W}_i^j)||^2}{2\sigma_2^2} \right)$$

However, which of the points, on the affine space should be taken to be $\mathbb{X}$? A solution might be to produce a random vector in the space $A_{\mathbb{W}}$, with equal probability of having any of them. If this solution is accepted, then we have used the weak approach. This approach ignores the fact that manifolds of data, with the same class, can be fairly complicated. Therefore, the author offers to consider a Gaussian distribution on the space $A_{\mathbb{W}}$, which makes the process of data production more local.

After explaining the accuracy of the strong approach, the weak approach is investigated in the next paragraphs.

Although the expressions (33a) to (33f) are mathematically accurate, they become long and rather hard to handle. Note that A and A^c are functions of prototypes $\mathbb{W}_i^j$ and shall not be treated as constant matrices. In case there is not much information on $\frac{\partial A^c}{\partial \mathbb{W}_i^j}$, it is not possible to continue with the strong approach, as numerical values are required to update the position of the prototypes. To overcome this problem, we may consider the tangent space of the random data point $\mathbb{X}$, that is presented in an online scheme. This approach deviates from the assumption that prototypes produce data points, according to

$$\mathbb{W} + A_{\mathbb{W}} N_{\mathbb{R}^r}(0,\sigma_1) + A_{\mathbb{W}}^c N_{\mathbb{R}^{n-r}}(0,\sigma_2) \ , \ \sigma_1 \gg \sigma_2 > 0$$

and because single-sided tangent distance is non-symmetrical, considering the tangent space of $\mathbb{X}$, may not optimise the same problem. The only hope here is to get a more convenient updating rule. The gradient, when the weak approach is adopted, is as in equation (33a), using

$$p(\mathbb{X} | \mathbb{W}_i^j) = \frac{1}{\sqrt{2\pi\sigma_2}} exp\left(-\frac{d_s^2(\mathbb{W}_i^j, \mathbb{X})}{2\sigma_2^2} \right) \text{ (34)}$$

where

$$d_s(\mathbb{W}_i^j, \mathbb{X}) = || (I - A_{\mathbb{X}} A_{\mathbb{X}}^T)(\mathbb{W}_i^j - \mathbb{X}) ||$$

Note that one can find that $A^c = (I - AA^T)$.
Having (34), we compute the required terms in (33a), as below.

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y | W) = \frac{\delta(j - y)p_i^j}{\sigma_2^2 p(\mathbb{X}, y | W)} p(\mathbb{X} | \mathbb{W}_i^j)(I - A_{\mathbb{X}} A_{\mathbb{X}}^T)^2(\mathbb{X} - \mathbb{W}_i^j) \text{ } (35a)$$

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} | W) = \frac{-(1 - \delta(j - y))p_i^j}{\sigma_2^2 p(\mathbb{X}, \bar{y} | W)} p(\mathbb{X} | \mathbb{W}_i^j)(I - A_{\mathbb{X}} A_{\mathbb{X}}^T)^2(\mathbb{X} - \mathbb{W}_i^j) \text{ } (35b)$$

In the rest of the paper, the first approach, resulting in the updating rules described in (33a) to (33f), is called the strong approach and the other approach, described in equations (34) to (35b), is called the weak approach.

An example is given to show the way one should think about the Simard approach in learning a classification problem. An artificial classification problem, in $\mathbb{R}^2$, is generated the following. We would like to define a family of curves $M(\mathbb{X}, \theta)$, on which data points are equivalent. In other words, if $\mathbb{X}_1 \in \mathbb{R}^2$ and $\mathbb{X}_2 \in \mathbb{R}^2$ are on the same curve, which means there are θ_1 and θ_2 such that $\mathbb{X}_2 = M(\mathbb{X}_1, \theta_1)$ and $\mathbb{X}_1 = M(\mathbb{X}_2, \theta_2)$, then $d_M(\mathbb{X}_1, \mathbb{X}_2) = 0$. For a data point $\mathbb{X} = [x_1, x_2]^T \in \mathbb{R}^2$, the corresponding curve is defined as

$$M(\mathbb{X}, \theta) = g(f(\mathbb{X}), \theta)$$

where

$$f(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ tan^{-1}(\frac{x_2}{x_1}) \end{bmatrix}$$

and

$$g(\begin{bmatrix} \alpha \\ \theta_0 \end{bmatrix}, \theta) = \alpha \times \begin{bmatrix} cos(\theta + \theta_0) \\ 0.5sin(\theta + \theta_0) \end{bmatrix} = \alpha \times \begin{bmatrix} cos\theta cos\theta_0 - sin\theta sin\theta_0 \\ 0.5sin\theta cos\theta_0 - 0.5cos\theta sin\theta_0 \end{bmatrix}$$

We can calculate $tan\theta_0 = \dfrac{2x_2}{x_1}$ and , consequently, $cos\theta_0$ and $sin\theta_0$. The constant α,

which indicates a family of curves, can also be found as below.

$$\alpha = \frac{x_1}{cos\theta_0} = \sqrt{x_1^2 + 4x_2^2}$$

One can easily check if $M(\mathbb{X}, \theta)$, which gives a one-dimensional manifold in $\mathbb{R}^2$, satisfies the required property $M(\mathbb{X}, \theta)\Big|_{\theta=0} = \mathbb{X}$.

The tangent space, induced by $M(\mathbb{X}, \theta)$, is calculated here.

$$\frac{\partial M(\mathbb{X}, \theta)}{\partial \theta} = \alpha \times \begin{bmatrix} -sin\theta cos\theta_0 - cos\theta sin\theta_0 \\ 0.5cos\theta cos\theta_0 + 0.5sin\theta sin\theta_0 \end{bmatrix}$$

Consequently, we get

$$A_{\mathbb{X}} = \frac{\partial M(\mathbb{X},\theta)}{\partial \theta}\bigg|_{\theta=0} = \begin{bmatrix} -\alpha sin\theta_0 \\ 0.5\alpha cos\theta_0 \end{bmatrix} = \begin{bmatrix} -2x_2 \\ 0.5x_1 \end{bmatrix}$$

5-2- SLVQ, using tangent distances (Saralajew-Villmann Assumption)

As mentioned in section 5-1, the class preserving transformations are usually not known in advance. As a more general approach, S. Saralajew and T. Villmann[3] estimate the local tangent spaces, according to the training data set T, as the corresponding learning process is being performed. To be precise, the tangent space, corresponding to each prototype $\mathbb{W}_i^j$, is adapted using the gradient descent learning. Hence, the tangent spaces of prototypes would be added to the set of adaptable parameters.

In the previous section, two learning approaches (weak and strong) were described. The question is that whether we can use any of them in this section. Before continuing with the discussion, it is pointed out that, in the previous section, the tangent spaces were always made on the data side (not the prototype side), when the single-sided tangent distance was to be calculated. therefore, the discussion, made in this paragraph and the next one, is based on the mentioned assumption. With the weak approach, we have assumed that the tangent space, required to calculate single-sided tangent distance, is estimated on the presented data point $\mathbb{X}$, at a certain time. With Simard assumption, we readily have the tangent space $A_{\mathbb{X}}$, as the transformation $M(\mathbb{X},\theta)$ is given. However, with Saralajew-Villmann assumption, it is not practically possible to have $A_{\mathbb{X}}$, for all $\mathbb{X}$. The reason is that there is a huge number of data points in a training set T and if a tangent space is considered for each, we would be left with an even greater number of adapting parameters. Even if it is done, we would not have enough information to meaningfully train the model. Therefore the weak approach is not an option in this section. The solution would be to modify the weak approach by having the tangent spaces defined on the prototype side.

The strong approach is closer to what is required in this section, in a sense that the tangent space is created for each prototype $\mathbb{W}_i^j$. However, one can see, by examining the updating rule of the strong approach, that the gradient of A, with respect to $\mathbb{X}$, still should be known, in order to calculate the updates.

The mentioned difficulties make it hard to directly use either of weak or strong approaches in this section (with Saralajew-Villmann assumption). Therefore, a simplifying assumption is made: “The tangent space $A_{\mathbb{W}_i^j}$ of the prototype $\mathbb{W}_i^j$ is independent of $\mathbb{W}_i^j$”. The assumption suggests that $A_{\mathbb{W}_i^j}$ can be treated as a constant matrix, when the partial derivatives of the likelihood function is taken, with respect to $\mathbb{W}_i^j$. This assumption is generally not true. However, it reduces the complexity of the algorithm, with the cost of having a model that deviates from the original model. The learning algorithm, in the new scenario is as the following. The set of adaptive parameters is $W \cup A_r(W)$, where $A_r(W) = \left\{A_r(\mathbb{W}_i^j) \mid j \in L, i = \{1,2,\ldots,k_j\}\right\}$and $A_r(\mathbb{W}_i^j)$ is a $n \times r$ orthogonal matrix, associated with the prototype $\mathbb{W}_i^j$. It is assumed that the dimensions r of the manifold $M(\mathbb{X}, \theta)$ is known. Members of $A_r(W)$ are initialised randomly, similar to what is done to prototypes. If the learning scheme is online, the log likelihood function is defined as

$$L_r(\mathbb{X}, W, A_r(W)) = log \frac{p\left(\mathbb{X}, y \mid \mathbb{W}, A_r(W)\right)}{p\left(\mathbb{X}, \bar{y} \mid \mathbb{X}, A_r(\mathbb{X})\right)} \quad (36)$$

The gradient for the prototypes is

$$\frac{\partial}{\partial \mathbb{W}_i^j} log \frac{p\left(\mathbb{X}, y \mid W, A_r(W)\right)}{p\left(\mathbb{X}, \bar{y} \mid W, A_r(W)\right)} = \frac{\partial}{\partial \mathbb{W}_i^j} logp\left(\mathbb{X}, y \mid W, A_r(W)\right) - \frac{\partial}{\partial \mathbb{W}_i^j} logp\left(\mathbb{X}, \bar{y} \mid W, A_r(W)\right) \quad (37a)$$

At this point, we would like to mention that both weak and strong approaches, with the tangent space defined on the prototype side, would be treated in this section. It can be shown practically that the strong approach has a better performance, as expected, since it acts more locally. First, we assume the distance measurement is based on the weak approach. The updating rules would be as below and the method is called TSLVQ. Later in this section, we would do the same, with strong approach, and the method would be called TSLVQ_S (the “S”, at the end, stands for strong).

The first term of the right hand side of (37a), using the weak approach, is

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\frac{\delta(j-y) p_i^j}{\sigma_2^2 p(\mathbb{X}, y \mid W, A_r(W))} p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j)) \left(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)^2 (\mathbb{X} - \mathbb{W}_i^j) \quad (37b)$$

The second term of the right hand side of (37a) is

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W, A_r(W)) =$$

$$\frac{(1-\delta(j-y))p_i^j}{\sigma_2^2 p(\mathbb{X},\bar{y}\,|\,W,A_r(W))}p(\mathbb{X}\,|\,\mathbb{W}_i^j,A_r(\mathbb{W}_i^j))\big(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)^2(\mathbb{X}-\mathbb{W}_i^j)\ (37c)$$

With

$$p(\mathbb{X}\,|\,\mathbb{W}_i^j,A_r(\mathbb{W}_i^j))=\frac{1}{\sqrt{2\pi\sigma_2^2}}exp\left(-\frac{d_s^2(\mathbb{X},\mathbb{W}_i^j)}{2\sigma_2^2}\right)$$

where d_s is the single-sided tangent distance.
The gradient of the tangent spaces $A_r(\mathbb{W}_i^j)$ is

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}log\frac{p(\mathbb{X},y\,|\,W,A_r(W))}{p(\mathbb{X},\bar{y}\,|\,W,A_r(W))}=$$
$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}logp(\mathbb{X},y\,|\,W,A_r(W))-\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}logp(\mathbb{X},\bar{y}\,|\,W,A_r(W))\ (38a)$$

with

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}logp(\mathbb{X},y\,|\,W,A_r(W))=$$
$$\delta(j-y)\left(\frac{2p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j,A_r(\mathbb{W}_i^j))}{\sigma_2^2 p(\mathbb{X},y\,|\,W,A_r(W))}\times(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\big(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)\right)A_r(\mathbb{W}_i^j)\ (38b)$$

and

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}logp(\mathbb{X},\bar{y}\,|\,W,A_r(W))=$$
$$(1-\delta(j-y))\left(\frac{2p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j,A_r(\mathbb{W}_i^j))}{\sigma_2^2 p(\mathbb{X},\bar{y}\,|\,W,A_r(W))}\times(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\big(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)\right)A_r(\mathbb{W}_i^j)\ (38c)$$

In Appendix A, a few steps, that was taken to achieve (38b) and (38c), are included. Simply, if the label y of the presented $\mathbb{X}$ is the same as the label of a prototype $\mathbb{W}_i^j$, the derivative of the first term (38b) is used, while for the opposite case, the derivative of the second term (38c) is used.
Till this point, we have only derived the derivatives, according to the gradient descent learning scheme. The updated tangent spaces, however, matrices, associated with the tangent spaces, are not necessarily in the orthogonal form. Orthonormal form being that $(A_r(\mathbb{W}_i^j))^T A_r(\mathbb{W}_i^j)=I$, for all prototypes $\mathbb{W}_i^j$. Therefore, after each adapting step,

tangent spaces are put in the orthonormal form. This process is necessary as in each learning step the distance between a presented data point $\mathbb{X}$ and prototypes are calculated, using a formula that assumes the tangent spaces to be orthonormal.
At this point, two fundamental questions are addressed, Firstly, one needs to find an appropriate dimensions r of the manifolds $M(\mathbb{X}, \theta)$, embedded in the original space. Secondly, it is not known whether the tangent spaces preserve their rank after being updated, at each learning step.

The second problem is rather easier to deal with. An adapted tangent space $A_r(\mathbb{W})$, corresponding to a prototype $\mathbb{W}$, would be of the following form(according to (38b) and (38c)).

$$A \pm c(\mathbb{X} - \mathbb{W})(\mathbb{X} - \mathbb{W})^T(I - AA^T)A$$

where

$$c = \frac{2p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{\sigma_2^2 p(\mathbb{X}, y \mid W, A_r(W))} \neq 0$$

or

$$c = \frac{2p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{\sigma_2^2 p(\mathbb{X}, \bar{y} \mid W, A_r(W))} \neq 0$$

Note that the unnecessary notation is dropped, in the above expressions, for brevity.

Then, we may factor out A and rewrite the expression as below.

$$\left(I \pm c(\mathbb{X} - \mathbb{W})(\mathbb{X} - \mathbb{W})^T(I - AA^T)\right)A$$

If the intension is to preserve the rank of the matrix A, we would better have the matrix

$$I \pm c(\mathbb{X} - \mathbb{W})(\mathbb{X} - \mathbb{W})^T(I - AA^T)$$

to be a full rank matrix. Because, if it is the case, the rank of the product would be exactly r. In order to make sure that the above matrix is full rank, we may use a special case of the "Matrix Determinant Lemma". It says that if $u, v \in \mathbb{R}^n$ are column vectors, then

$$det(I + uv^T) = 1 + u^T v$$

We may take $u = c(\mathbb{X} - \mathbb{W})$ and $v^T = (\mathbb{X} - \mathbb{W})^T(I - AA^T)$. Hence, we get

$$det\big(I \pm c(\mathbb{X} - \mathbb{W})(\mathbb{X} - \mathbb{W})^T(I - AA^T)\big) = 1 \pm c(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W})$$

Consequently, if

$$c(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W}) \notin \{-1,1\}$$

then, we have got a full rank matrix. This an easy task to prove that the above relation always holds, for the case

$$c(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W}) \notin \{-1\}$$

as long as A is an orthogonal matrix. One needs to know that $(I - AA^T)$ is a positive semidefinite matrix, meaning that

$$x^T(I - AA^T)x \geq 0 \ \forall x$$

To show the truth of the expression above, it is expanded.

$$x^T(I - AA^T)x = x^T x - (A^T x)^T(A^T x) = ||x|| - ||Ax||$$

Since A is orthogonal,

$$||Ax|| \leq ||A||.||x|| = ||x||$$

Therefore, it is proven that the first case always holds and consequently, we do not need to worry about the rank of the tangent spaces A to decrease.

However, we may fail to prove the following

$$c(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W}) \notin \{1\}$$

It means that the repulsion mechanism, in TSLVQ is troublesome. Although the probability of

$$c(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W}) \in \{1\}$$

Is small, but there is still a small chance we have to reduce the rank of a tangent space. Hopefully, we may prevent such a problem, by taking a small updating rate, which makes the factor c to be small. More formally

$$(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W}) \leq ||\mathbb{X} - \mathbb{W}||^2$$

Which is valid, when $(I - AA^T)$is an orthonormal matrix. Therefore

$$c(\mathbb{X} - \mathbb{W})^T(I - AA^T)(\mathbb{X} - \mathbb{W}) \leq c||\mathbb{X} - \mathbb{W}||^2$$

Now, if we have

$$c < \frac{1}{||\mathbb{X} - \mathbb{W}||^2}$$

we can assume that we are safe. Note that we need to have some estimation on the maximum value of $||\mathbb{X} - \mathbb{W}||^2$ and it definitely depends on the classification problem.

TSLVQ, with Saralajew-Villmann assumption, was used to learn the butterfly problem (described in section 3-2), with the dimensions of tangent spaces set to be 1, i.e., lines in $\mathbb{R}^2$ would be prototypes. The result reveals that the algorithm reduces the initial misclassification error rate, significantly. Approximately, with an initial CER of 0.5, the final CER would become 0.1, when 10^5 learning steps are taken. However, TSLVQ is not as effective as ASLVQ (although it works better than SLVQ). This is not an expected result, for the structure of the classification problem is more compatible with the version of SLVQ, in which lines are taken as prototypes. The suspicion is on the unstable behaviours of SLVQ. In the next section, TRSLVQ would be used as an alternative to see if a better result is achieved.

The Matlab program of TSLVQ is included in the programs section, as program 4.

As the second version of TSLVQ (TSLVQ_S), we would like to use the strong approach. In theory, one should observe improvement by using the strong approach. We are going to derive the updating rules first and, then, it would be practically examined.

The likelihood function and the gradient, with respect to a prototype $\mathbb{W}_i^j$, are the same as in (36) and (37a), respectively. However, the probability of producing $\mathbb{X}$, when prototype $\mathbb{W}_i^j$ (with $A_r(\mathbb{W}_i^j)$) is chosen to produce, differs a bit, just to take the strong approach into account. It would be

$$p(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j)) =$$

$$\frac{1}{2\pi\sigma_1\sigma_2} exp\left(-\frac{||\left(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)(\mathbb{X} - \mathbb{W}_i^j)||^2}{2\sigma_2^2} - \frac{||A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T(\mathbb{X} - \mathbb{W}_i^j)||^2}{2\sigma_1^2} \right) \quad (39)$$

Having (39), the first and the second term of the right hand side of (37a), would be as below, respectively.

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y\,|\,W, A_r(W)) = \frac{\delta(j-y)p_i^j}{p(\mathbb{X}, y\,|\,W, A_r(W))} p(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j))$$

$$\times \left(\frac{(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T)^2(\mathbb{X} - \mathbb{W}_i^j)}{\sigma_2^2} + \frac{\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)^2(\mathbb{X} - \mathbb{W}_i^j)}{\sigma_1^2} \right) \quad (40a)$$

and

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W, A_r(W)) = \frac{\big(1-\delta(j-y)\big)p_i^j}{p(\mathbb{X}, \bar{y} \mid W, A_r(W))} p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))$$

$$\times \left(\frac{(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T)^2(\mathbb{X} - \mathbb{W}_i^j)}{\sigma_2^2} + \frac{\big(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)^2(\mathbb{X} - \mathbb{W}_i^j)}{\sigma_1^2} \right) \ (40b)$$

In order to update the tangent spaces, the gradient of the tangent spaces would be as (38a), with the first and the second terms of the right hand side of (38a) being as the following, respectively.

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logP(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\delta(j-y)\left(\frac{2p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{p(\mathbb{X}, y \mid W, A_r(W))} \right)\left(-\frac{(\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\big(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)}{\sigma_2^2} + \frac{(\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\big(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)}{\sigma_1^2} \right) A_r(\mathbb{W}_i^j) \ (41a)$$

and

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logP(\mathbb{X}, \bar{y} \mid W, A_r(W)) =$$

$$\big(1-\delta(j-y)\big)\left(\frac{2p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{p(\mathbb{X}, \bar{y} \mid W, A_r(W))} \right)\left(-\frac{(\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\big(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)}{\sigma_2^2} + \frac{(\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\big(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)}{\sigma_1^2} \right) A_r(\mathbb{W}_i^j) \ (41b)$$

A single computing step, regarding the derivation of (41a), is included in appendix B.

We would like to investigate the rank of the updated versions of the tangent spaces. The procedure would be the same as what was done for the weak approach. The updated version of the tangent space matrix would be of form

$$A \pm c\left(-\frac{(\mathbb{X}-\mathbb{W})(\mathbb{X}-\mathbb{W})^T(I-AA^T)}{\sigma_2^2} + \frac{(\mathbb{X}-\mathbb{W})(\mathbb{X}-\mathbb{W})^T(AA^T)}{\sigma_1^2} \right)A =$$
$$\left(I \pm c\left(-\frac{(\mathbb{X}-\mathbb{W})(\mathbb{X}-\mathbb{W})^T(I-AA^T)}{\sigma_2^2} + \frac{(\mathbb{X}-\mathbb{W})(\mathbb{X}-\mathbb{W})^T(AA^T)}{\sigma_1^2} \right)\right)A$$

Where

$$c = \frac{2p_i^j p(\mathbb{X} \,|\, \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{p(\mathbb{X}, \bar{y} \,|\, W, A_r(W))}$$

Then, the matrix, in the big parenthesis should be full rank. We may use the "Matrix Determinant Lemma"

$$det(I + uv^T) = 1 + u^T v$$

to have the determinant of the matrix as below.

$$1 \pm \frac{c}{\sigma_1^2\sigma_2^2}(\sigma_2^2 - \sigma_1^2)(\mathbb{X}-\mathbb{W})^T(\mathbb{X}-\mathbb{W}) = 1 \pm \frac{c}{\sigma_1^2\sigma_2^2}(\sigma_2^2 - \sigma_1^2)||\mathbb{X}-\mathbb{W}||^2$$

It is required to have

$$\frac{c}{\sigma_1^2\sigma_2^2}(\sigma_2^2 - \sigma_1^2)||\mathbb{X}-\mathbb{W}||^2 \notin \{-1,1\}$$

Since

$$\frac{c}{\sigma_1^2\sigma_2^2}(\sigma_2^2 - \sigma_1^2)||\mathbb{X} - \mathbb{W}||^2 \leq 0$$

We only need to check the inequality below, to be safe.

$$\frac{c}{\sigma_1^2\sigma_2^2}(\sigma_2^2 - \sigma_1^2)||\mathbb{X} - \mathbb{W}||^2 \notin \{-1\}$$

We may choose parameters such that

$$c < \frac{\sigma_1^2\sigma_2^2}{(\sigma_1^2 - \sigma_2^2)||\mathbb{X} - \mathbb{W}||^2}$$

and for that, an estimation on the maximum value of $||\mathbb{X} - \mathbb{W}||^2$ is required.

Lastly, a piece of practical advice is given. When applying TSLVQ_S to problems with high dimensions, it is necessary to increase σ_1 and σ_2 relatively. Otherwise the sums $p(\mathbb{X}, y \mid W, A_r(W))$ and $p(\mathbb{X}, \bar{y} \mid W, A_r(W))$ tend to become too small, so that the updating expressions, that have the sums in their denominators, go to infinity and, consequently, Matlab would update the parameters to be "Not a Number" (NaN). If it happens, then we cannot continue with the rest of the algorithm, computationally. TSLVQ_S was examined practically and the result was much better and more stable than TSLVQ. The Matlab program of TSLVQ_S can be found under the name Program 8, in Matlab programs section.

5-3- RSLVQ, using tangent distances (TRSLVQ)

Robust soft learning vector quantisation (RSLVQ), introduced earlier in section 3-3, is a more stable version of SLVQ, in which prototypes do not diverge. In this section, tangent distances would be used, as distance measurements, in RSLVQ and the corresponding updating rule would be derived. Assuming that the main philosophy is developed in previous sections (while developing TSLVQ), we are going include the main issues here. Also, only Saralajew-Villmann assumption would be treated in this section and the set of prototypes and their corresponding tangent spaces would be assumed to be independent, as in the previous section.

First, we start with the weak approach. The log likelihood function would be as the following.

$$L_r(\mathbb{X}, W, A_r(W)) = log\frac{p(\mathbb{X}, y \mid W, A_r(W))}{p(\mathbb{X}, y \mid W, A_r(W)) + p(\mathbb{X}, \bar{y} \mid W, A_r(W))} = log\frac{p(\mathbb{X}, y \mid W, A_r(W))}{p(\mathbb{X} \mid W, A_r(W))} \quad (42)$$

The gradient, with respect to the prototypes is

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W, A_r(W)) - \frac{\partial}{\partial \mathbb{W}_i^j} log\big(p(\mathbb{X}, y \mid W, A_r(W)) + p(\mathbb{X}, \bar{y} \mid W, A_r(W))\big) \quad (43a)$$

With the first and the second terms, in (43a) as

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\frac{\delta(j-y)p_i^j}{\sigma_2^2 p(\mathbb{X}, y \mid W, A_r(W))} p(\mathbb{X} \mid \mathbb{W}_i^j)\left(I - A_r(\mathbb{W}_i^j)\big(A_r(\mathbb{W}_i^j)\big)^T\right)^2 (\mathbb{X} - \mathbb{W}_i^j) \quad (43b)$$

and

$$\frac{\partial}{\partial \mathbb{W}_i^j} log\left(p(\mathbb{X}, y \mid W, A_r(W)) + p(\mathbb{X}, \bar{y} \mid W, A_r(W))\right) = \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}) =$$

$$\frac{-p_i^j}{\sigma_2^2 p(\mathbb{X})} p(\mathbb{X} \mid \mathbb{W}_i^j)\left(I - A_r(\mathbb{W}_i^j)\big(A_r(\mathbb{W}_i^j)\big)^T\right)^2 (\mathbb{X} - \mathbb{W}_i^j) \quad (43c)$$

The gradient, with respect to the tangent spaces is as below.

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logp(\mathbb{X}, y \mid W, A_r(W)) - \frac{\partial}{\partial A_r(\mathbb{W}_i^j)} log\big(p(\mathbb{X}, y \mid W, A_r(W)) + p(\mathbb{X}, \bar{y} \mid W, A_r(W))\big) \quad (44a)$$

with

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logp(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\delta(j - y)\left(\frac{2p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{\sigma_2^2 p(\mathbb{X}, y \mid W, A_r(W))}(\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\big(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)\right)A_r(\mathbb{W}_i^j) \quad (44b)$$

and

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} log\big(p(\mathbb{X}, y \mid W, A_r(W)) + p(\mathbb{X}, \bar{y} \mid W, A_r(W))\big) =$$

$$\left(\frac{2p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{\sigma_2^2 p(\mathbb{X})}(\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\big(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)\right)A_r(\mathbb{W}_i^j) \quad (44c)$$

We would like to check if the updated tangent spaces are of the same rank as the initial tangent spaces. The form of the updates would be similar to the case of TSLVQ, in the previous section. The only difference is that the coefficient c is different now.

$$c = \frac{2p_i^j p(\mathbb{X} | \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))}{\sigma_2^2} \left(\frac{1}{p(\mathbb{X}, y | W, A_r(W))} - \frac{1}{p(\mathbb{X})} \right)$$

Therefore, we need to have the following constraint on c.

$$c < \frac{1}{||\mathbb{X} - \mathbb{W}||^2}$$

The maximum of $||\mathbb{X} - \mathbb{W}||^2$, in practice, should be found to find a suitable c.

As promised, in the previous section, TRSLVQ is used to learn the butterfly problem and the result is much better than TSLVQ. With an initial CER of 0.5, the final CER, after 10^5 learning steps, is around 0.05. In a similar situation, ASLVQ gives a final CER of 0.01, which is still better. However, one important issue is that ASLVQ achieves the CER quick, but not much of a progress is made, when the number of steps are increased. On the other hand, TRSLVQ would continue to get better as the number of learning iterations is increased.

The Matlab program of TRSLVQ is included in the programs section, under the name program-5.

As the second version of TRSLVQ, TRSLVQ_S, equipped with the strong approach, is presented here. The gradient would be as in (43a), with the first and the second terms of the right hand side to be

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y | W, A_r(W)) =$$

$$\frac{\delta(j-y)p_i^j}{p(\mathbb{X}, y | W, A_r(W))} p\big(\mathbb{X} | \mathbb{W}_i^j, A_r(\mathbb{W}_i^j)\big)$$

$$\times \left(\frac{\big(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\big)^2}{\sigma_2^2} + \frac{(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T)^2}{\sigma_1^2} \right)(\mathbb{X} - \mathbb{W}_i^j) \ (45a)$$

and

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}) =$$

$$\frac{\left(1-\delta(j-y)\right)p_i^j}{p(\mathbb{X})}p(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j))\left(\frac{\left(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)^2}{\sigma_2^2}+\frac{(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T)^2}{\sigma_1^2}\right)(\mathbb{X}-\mathbb{W}_i^j)+$$

$$\frac{\delta(j-y)p_i^j}{p(\mathbb{X})}p(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j))\left(\frac{\left(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)^2}{\sigma_2^2}+\frac{(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T)^2}{\sigma_1^2}\right)(\mathbb{X}-\mathbb{W}_i^j) \quad (45b)$$

Note that

$$p(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j)) =$$

$$\frac{1}{2\pi\sigma_1\sigma_2}exp\left(-\frac{\left|\left|\left(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)(\mathbb{X}-\mathbb{W}_i^j)\right|\right|^2}{2\sigma_2^2}-\frac{\left|\left|A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T(\mathbb{X}-\mathbb{W}_i^j)\right|\right|^2}{2\sigma_1^2}\right)$$

The gradient of the tangent spaces is as in (44a), with

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}logp(\mathbb{X},y\,|\,W,A_r(W)) = \delta(j-y)\left(\frac{2p_i^j p\left(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j)\right)}{p(\mathbb{X},y\,|\,W,A_r(W))}\right)$$

$$\times\left(-\frac{(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\left(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_2^2}+\frac{(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_1^2}\right)A_r(\mathbb{W}_i^j) \quad (46a)$$

and

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)}logp(\mathbb{X}\,|\,W,A_r(W)) = \left(1-\delta(j-y)\right)\left(\frac{2p_i^j p\left(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j)\right)}{p(\mathbb{X}\,|\,W,A_r(W))}\right)$$

$$\times\left(-\frac{(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\left(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_2^2}+\frac{(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_1^2}\right)A_r(\mathbb{W}_i^j)$$

$$+\delta(j-y)\left(\frac{2p_i^j p\left(\mathbb{X}\,|\,\mathbb{W}_i^j, A_r(\mathbb{W}_i^j)\right)}{p(\mathbb{X}\,|\,W,A_r(W))}\right)$$

$$\times\left(-\frac{(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\left(I-A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_2^2}+\frac{(\mathbb{X}-\mathbb{W}_i^j)(\mathbb{X}-\mathbb{W}_i^j)^T\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_1^2}\right)A_r(\mathbb{W}_i^j) \quad (46b)$$

TRSLVQ_S is the best method of the family of tangent distance SLVQ (RSLVQ), when attacking a classification problem. Its excellence (with respect to similar alternatives) has been expected, theoretically, as we tried to make the assumptions stronger and use the more stable version, i.e. RSLVQ. Also, in practice, it was observed that TRSLVQ_S, together with TSLVQ_S, are the only algorithms, provided in this paper (excluding Grassmannian method that comes later) that can actually deal with a real world problem, with high dimensions. The rest may show nice results, when applied to toy problems.

The Matlab program of TRSLVQ_S is included in the programs part, under the name program-9.

5-4- PCA, tangent space for local data, TDSLVQ and TDRSLVQ

Although TSLVQ (TSLVQ_S)and TRSLVQ (TRSLVQ_S) are rather successful, in this section, a new approach is provided. We stick to Saralajew-Villmann assumption and, also, we would like to use the strong approach here. More precisely, we do not know the class-preserving transforms, but we would like to have Gaussian distribution on both the tangent and the complement space. To pave the way for this approach, we would like to use local data points, given by the training set, to approximate the tangent space at a certain data point $\mathbb{X}$. Having the tangent space (call it A), the distances of a prototype $\mathbb{W}$ to $\mathbb{X}$ can be calculated as below.

$$d_2(\mathbb{X}, \mathbb{W}) = ||(I - AA^T)(\mathbb{X} - \mathbb{W})||$$

$$d_1(\mathbb{X}, \mathbb{W}) = ||AA^T(\mathbb{X} - \mathbb{W})||$$

The approximation of tangent space, at a data point $\mathbb{X}$, is done as the following. Data point $(\mathbb{X}, y)$ is presented to the system, by the online scheme. An open ball around $\mathbb{X}$ is considered and all the data points, in the training set T that lie in the open ball and have label y, are used to approximate the r-affine (r is the dimensions of the tangent space) that best fits the collection of data. In order to approximate the r-affine, we may use PCA. The final output of this step is a matrix $A(\mathbb{X})$, containing the tangent space of $\mathbb{X}$. Note that, with this method, the cost function, to be minimised by the best r-affine, is as below.

$$f(V, A) = \sum_{i=1}^{m} ||(I - AA^T)(\mathbb{X}_i - V)|| \quad (47)$$

In (47), the data point collection is $\{\mathbb{X}_i\}_{i=1}^{m}$ and the r-affine is $V + A\theta$, where θ is the free variable, $V \in \mathbb{R}^n$ is an offset vector, and A has r columns.
The gradients would be as below.

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W) - \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W)$$

with

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W) = \frac{\delta(j-y) p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{2p(\mathbb{X}, y \mid W)} \left(\frac{AA^T}{\sigma_1^2} + \frac{I - AA^T}{\sigma_2^2} \right) \left(\mathbb{X} - \mathbb{W}_i^j \right) \quad (48a)$$

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W) = \frac{\left(1 - \delta(j-y)\right) p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{2p(\mathbb{X}, y \mid W)} \left(\frac{AA^T}{\sigma_1^2} + \frac{I - AA^T}{\sigma_2^2} \right) \left(\mathbb{X} - \mathbb{W}_i^j \right) \quad (48b)$$

Note that there would be no tangent spaces associated with the prototypes, which is an advantage, since we have reduced the number of adapting parameters.

This approach would be named TDSLVQ, which stands for “Tangent Data SLVQ”. The corresponding program would be found under the name program6 in Matlab program section.

Program 6 uses a function called “r-affine”, that is also included in programs part, under the name program 7.

One can see that TDSLVQ uses less parameters. However, it is notably slower than TSLVQ and TRSLVQ. The reason is the process of finding a r-affine for a collection of data points. Also, the performance appears to be lower.

The last thing, to be investigated in this section, is TDRSLVQ, which is the same as TDSLVQ, but it uses RSLVQ cost function instead. As before, we are seeking a more stable version of TDSLVQ.

With the very same philosophy as in the case of TDSLVQ, the gradient of TDRSLVQ is as below.

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W) - \frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, \bar{y} \mid W)$$

with

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}, y \mid W) =$$

$$\frac{\delta(j-y)p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{2}\left(\frac{1}{p(\mathbb{X}, y \mid W)} - \frac{1}{p(\mathbb{X})}\right)\left(\frac{AA^T}{\sigma_1^2} + \frac{I - AA^T}{\sigma_2^2}\right)\left(\mathbb{X} - \mathbb{W}_i^j\right) \ (49a)$$

and

$$\frac{\partial}{\partial \mathbb{W}_i^j} logp(\mathbb{X}) =$$

$$\frac{\left(1-\delta(j-y)\right)p_i^j p(\mathbb{X} \mid \mathbb{W}_i^j)}{2p(\mathbb{X})}\left(\frac{AA^T}{\sigma_1^2} + \frac{I - AA^T}{\sigma_2^2}\right)\left(\mathbb{X} - \mathbb{W}_i^j\right) \ (49b)$$

The advantage of TDSLVQ and TDRSLVQ is that we can have the strong approach and still we are able to derive the updating rules, without much trouble (refer to 5-1, where the strong approach, with Simard assumption, was left incomplete). It was not the case for TSLVQ and TRSLVQ, where we defined the tangent spaces on the prototype side and treated data points as simple points. The disadvantage of TDSLVQ and TDRSLVQ is an issue that makes the learning methods practically useless for real-time system. Because, after the training phase, if a new data point is given to the system, to be classified, the system is requires to approximate the tangent space at the point and not only it would be time consuming, but also it requires a considerable number of data points of the same class. Note that, in online learning, it is assumed that no memory is dedicated to store the previously observed input data.

After providing the updating rules of TDSLVQ and TDRSLVQ, we would like to explain the reason that it is thought to work better, using a simple example. Assume the half-circle, defined as below in $\mathbb{R}^2$, serves as a one-dimensional manifold in $\mathbb{R}^2$. The manifold only contains data of the same class.

$$\alpha : (0,\pi) \to \mathbb{R}^2, \alpha(\theta) = \begin{bmatrix} cos\theta \\ Sin\theta \end{bmatrix}$$

Also, we assume that we have a prototype $\mathbb{W} \in \mathbb{R}^2$ to be adapted. The probability of all points $\mathbb{X}$ on the manifold, to be given to the online system, are equal. Now, we would like to consider 3 scenarios and find the fixed-points of each case. In the first case, we use updating rule of LVQ (we have dropped coefficients for simplicity), in the second case, we use the updating rule of the weak approach and, finally, we would assume the last case to use the strong approach, as described in this section.

For the last case, one notices that, given $\mathbb{X}$ as the random point, vector $\mathbb{X} - \mathbb{W}$ is decomposed to component $\frac{AA^T}{\sigma_1^2}(\mathbb{X} - \mathbb{W})$, which moves parallel to the tangent at $\mathbb{X}$, and component $\frac{I - AA^T}{\sigma_2^2}(\mathbb{X} - \mathbb{W})$, which moves in the direction that is perpendicular to the tangent space at $\mathbb{X}$. Therefore, not only $\mathbb{W}$ gets closer to the tangent spaces, but also to the cluster of the points on the manifold, with respect to Euclidean distance. To be more accurate, we would like to show that the strong approach (the third case) is a compromise between SLVQ and TSLVQ and it takes both Euclidean and tangent distances into account.

The fixed-point for $\mathbb{W}$, when the three methods are applied to the problem, can be found as the following. Note that the mathematical knowledge, that is used in the rest of this section, can be found on textbook “Analysis on Manifolds”, written by J.R. Munkres [10].

Firstly, a prototype $\mathbb{W}$ is a fixed point, when it would not change position, on average, when an updating rule is applied to it, in a certain learning problem. Mathematically, for the first case, it can be expressed as below.

$$\Delta\mathbb{W} = \int (\mathbb{X} - \mathbb{W}) p(\mathbb{X}) d\mathbb{X}$$

Since the manifold is a subset of $\mathbb{R}^2$, we may need to go through some steps. First, $\mathbb{X} - \mathbb{W}$, that appears in the integral, is a vector. Therefore, we may consider each of its components separately. Hence the problem is reduced to

$$\Delta\mathbb{W}_i = \int (\mathbb{X}_i - \mathbb{W}_i) p(\mathbb{X}) d\mathbb{X} \ , \ i = 1,2$$

Note that $\mathbb{X}_i - \mathbb{W}_i$ can be thought as a function on the space $\mathbb{R}^2$. If we call the function $f : \mathbb{R}^2 \to \mathbb{R}$, which is basically a shift by $\mathbb{W}_i$, we can write the integral as below.

$$\Delta\mathbb{W} = \frac{1}{\pi}\int f \circ \alpha(\theta) \times V\big(D(\alpha)\big)d\theta$$

$V\big(D(\alpha)\big)$ is the volume of the parallelepiped that is made by the vectors in $D(\alpha)$ and the formula for it is as below.

$$V\big(D(\alpha)\big) = \Big(det\big(D^T(\alpha)D(\alpha)\big)\Big)^{\frac{1}{2}}$$

In the problem in hand, $V\big(D(\alpha)\big) = 1$. Therefore, the integral becomes

$$\Delta\mathbb{W}_1 = \frac{1}{\pi}\int_0^\pi \big(cos\theta - \mathbb{W}_1\big)d\theta = -\mathbb{W}_1$$
$$\Delta\mathbb{W}_2 = \frac{1}{\pi}\int_0^\pi \big(sin\theta - \mathbb{W}_2\big)d\theta = \frac{2}{\pi} - \mathbb{W}_2$$

Putting them equal to zero, we achieve the fixed-points $\mathbb{W}_1 = 0$ and $\mathbb{W}_2 = \frac{2}{\pi}$, for the first scenario.

Going through similar processes, for the second and the third cases (the integration part is more difficult), we find the fixed-points to be (respectively)

$$\mathbb{W}_1 = 0 \ , \ \mathbb{W}_2 = \frac{4}{\pi} - \frac{8}{3\pi}$$
$$\mathbb{W}_1 = 0 \ , \ \mathbb{W}_2 = \frac{\frac{4}{\pi} - \frac{8}{3\pi}}{\frac{\sigma_2^2}{\sigma_1^2} + 1}$$

Note that the tangent spaces, required for the second and the third cases, can be found, theoretically, by calculating the following partial derivative.

$$D(\alpha) = \begin{bmatrix} -sin\theta \\ cos\theta \end{bmatrix}$$

Having a close look at the fixed-points of the second and the third case, one sees that the fixed-points of the weak approach does not depend σ_2, while in the third case (strong approach), the learning scheme is more flexible to adapt to the nature of the classification problem, since it depends on the ratio $\frac{\sigma_2}{\sigma_1}$. Furthermore, the third case would act like the second case, if $\sigma_2 \ll \sigma_1$, which can be interpreted that we have chosen the dimensions of the manifolds correctly and we do not expect much movement in the direction of the perpendicular space to the tangent space. However, it is assumed that it is practically hard to predict the correct dimensions of the class preserving manifolds. Therefore, we may choose σ_2 not to be ignorable (although it should be less that σ_1).
As the last point we would like to mention the general equations to derive the weak and strong approaches fixed-points. In a similar manner to the half-circle example, that was discussed earlier, we would like to assume a general case, in which the class-preserving manifold is described as $M : \mathbb{R}^r \to \mathbb{R}^n$, where r is the dimensions of the manifold and $\mathbb{R}^n$ is the space containing the data space. Assuming that tangent spaces are denoted by $A(\mathbb{X})$, the fixed-points of the weak approach should be derived from the equation below.

$$\int_\theta p(\mathbb{X})AA^T\big(M(\theta) - \mathbb{W}\big)V\big(D(M)\big)d\theta = \int_\theta p(\mathbb{X})\big(M(\theta) - \mathbb{W}\big)V\big(D(M)\big)d\theta$$

For the strong approach, we need to solve the following equation, which is a little different, since the integrals have different coefficients.

$$\left(1 - \frac{\sigma_2^2}{\sigma_1^2}\right)\int_\theta p(\mathbb{X})AA^T\big(M(\theta) - \mathbb{W}\big)V\big(D(M)\big)d\theta = \left(1 - \frac{\sigma_2^2}{\sigma_1^2}\right)\int_\theta p(\mathbb{X})\big(M(\theta) - \mathbb{W}\big)V\big(D(M)\big)d\theta$$

In order to investigate the stability of fixed points, one can find the Jacobean of the map that is used for the learning method. The Jacobean would be as below.

$$\left(1 - \frac{\mu}{\sigma_2^2}\int_\theta V\big(D(M)\big)d\theta\right)I + \mu\left(\frac{1}{\sigma_2^2} - \frac{1}{\sigma_1^2}\right)\int_\theta AA^TV\big(D(M)\big)d\theta$$

One can notice that the Jacobean does not depend on $\mathbb{W}$ and the stability of the fixed points only depends on the manifold (which affects A) and the parameters σ_1, σ_2, and μ. However, this is true to deduce that, for a certain problem, all the fixed points would be either stable or non-stable. The reason is that all the analysis was based on a single prototype $\mathbb{W}$. The way we would like to think about it is to have each prototype to be responsible for a region in $\mathbb{R}^n$. Therefore, the integral

$$\int_{\theta} AA^T V\big(D(M)\big)d\theta$$

may give different outputs, on different regions and, consequently, the stability of the fixed points changes accordingly. As a result, there might be regions, for which there are no stable fixed points.
The practical experience on real problems shows that TDSLVQ and TDRSLVQ converge quickly to local optima and changing σ_1 and σ_2 may help to avoid early convergence.

5-5- Performance of SLVQ and RSLVQ, based on tangent distances

It is a difficult task to investigate the performance of SLVQ and RSLVQ, with the Euclidean distance, analytically and the versions, in which tangent distances are utilised, are even more complicated to deal with. In order to have an idea about how successful the new versions of SLVQ and RSLVQ are, we may assume that we know how to measure the performance of simple SLVQ and RSLVQ and, then, try to convert the data space of a classification problem to one that can be dealt with, using simple SLVQ and RSLVQ.

Let's assume we are given a classification problem, in which class-preservative manifolds exist and these manifolds capture the structure of the data space, with respect to the classification problem. Also, we assume that the strong assumption is applicable, which means that if $\mathbb{W}$ is a prototype and $A(\mathbb{W})$ is the tangent space at $\mathbb{W}$, data points $\mathbb{Y}$, around $\mathbb{W}$, are produced according the following formula.

$$\mathbb{Y} = \mathbb{W} + A_{\mathbb{W}} N_{\mathbb{R}^r}(0,\sigma_1^2) + A_{\mathbb{W}}^c N_{\mathbb{R}^{n-r}}(0,\sigma_2^2)\ ,\ \sigma_1 \gg \sigma_2 > 0$$

The set of prototypes would be W and $\mathbb{W}_i, \mathbb{W}_j \in W$. Then, we may map the data points as the following. If $\mathbb{X}$ satisfies

$$\sigma_2^2 ||A_{\mathbb{W}_i} A_{\mathbb{W}_i}^T (\mathbb{X} - \mathbb{W}_i)||^2 + \sigma_1^2 ||(I - A_{\mathbb{W}_i} A_{\mathbb{W}_i}^T)(\mathbb{X} - \mathbb{W}_i)||^2 \leq$$

$$\sigma_2^2 ||A_{\mathbb{W}_j} A_{\mathbb{W}_j}^T (\mathbb{X} - \mathbb{W}_j)||^2 + \sigma_1^2 ||(I - A_{\mathbb{W}_j} A_{\mathbb{W}_j}^T)(\mathbb{X} - \mathbb{W}_j)||^2$$

then,

$$\mathbb{X} \to \mathbb{W}_i + (I - A_{\mathbb{W}_i} A_{\mathbb{W}_i}^T)(\mathbb{X} - \mathbb{W}_i) + \frac{\sigma_2^2}{\sigma_1^2} A_{\mathbb{W}_i} A_{\mathbb{W}_i}^T (\mathbb{X} - \mathbb{W}_i)$$

Or equivalently

$$\mathbb{X} \to \mathbb{X} + (\frac{\sigma_2^2}{\sigma_1^2} - 1) A_{\mathbb{W}_i} A_{\mathbb{W}_i}^T (\mathbb{X} - \mathbb{W}_i)$$

What this map basically does is to arrange points in a way that Euclidean distance becomes valid, in a sense that it gives a correct dissimilarity measurement, with respect to the classification problem in hand.

According to what is said so far, we would like to (roughly) claim that the performance of SLVQ or RSLVQ, with tangent distances, is in direct relation to the performance of simple SLVQ or RSLVQ, when they are applied to the version of the data point, created using the map above.

6- Grassmannian Manifolds, an introduction and its application in SLVQ

In the last few years, papers on machine learning, in which the concept of Grassmannian Manifolds are used, have been published. Grassmannian Manifolds provide a way of making new topologies, from the given input space, such that the new topology provides us a better chance of classifying the original objects. In this section, first a basic introduction on Grassmannian manifolds is provided. Then, the concept would be used to obtain a dissimilarity measurement, that is going to be a substitution for Euclidean distance, used in basic SLVQ by S. Seo and K. Obermayer [1].

6-1- Introduction to Grassmannian Manifolds

Classification methods, introduced in section 5, were based on tangent distances. In order to calculate tangent distances, one needs to have tangent spaces of the manifolds, that exist in the data space. Therefore, it is necessary to have manifolds that are differentiable and all the methods, in section 5, are valid as long as we know the manifolds are differentiable. It would be good if did not have to make such assumption, for there might be spaces such that either they have manifold that are not differentiable (at some points) or it is not easy to prove the manifolds are differentiable everywhere in the data space. Luckily, Grassmannian manifolds, would give us the opportunity to drop the assumption that the class-preserving manifolds, in the data space, are differentiable and, consequently, learning methods, that are based on Grassmannian manifolds, are applicable to a greater class of problems (including the differentiable problems).

First, we would like to formally define a Grassmannian and its corresponding quantities. Part of the information is taken from a paper by S. Chepushtanova and M. Kirby [5].

Definition: $G(m, \mathbb{R}^n)$ is the space of all k-dimensional subspaces in $\mathbb{R}^n$ and it is generally called the Grassmannian manifold.

Definition: A k-dimensional topological manifold is a topology $(\mathbb{X}, T_{\mathbb{X}})$, with $\mathbb{X}$ being a set and $T_{\mathbb{X}}$ being a topology defined on $\mathbb{X}$, such that for each $x \in \mathbb{X}$ there exists a neighbourhood of $N(\mathbb{X}) \in T_{\mathbb{X}}$ that is homeomorphic to an open set of $\mathbb{R}^k$.

Theorem: In general, $G(m, \mathbb{R}^n)$ is a k-dimensional topological manifold, with $k = m \times (n - m)$.

A proof is not provided here, as it is necessary to go into technical details. However, if we accept that $G(m, \mathbb{R}^n)$ is a manifold, then in order to find the dimensions, one needs to notice how one abstract point in $G(m, \mathbb{R}^n)$ can be represented as a $n \times m$ orthonormal matrix, which is basically a basis for the subspace. First, we define an equivalence relation between two matrices A and B, both with dimensions $n \times m$. $A \sim B$ if and only if there exists a full-rank matrix P, of dimensions $m \times m$, such that $AP = B$.

We claim that each point in $G(m, \mathbb{R}^n)$ is of form

$$F = \begin{bmatrix} I_m \\ R_{(n-m)\times m} \end{bmatrix}$$

where I_m is the identity matrix of dimension m and $R_{(n-m)\times m}$ is an arbitrary matrix of dimensions $(n-m)\times m$. The reason is that the rank of matrix F is exactly m, which gives a m-dimensional basis in $\mathbb{R}^n$. Therefore ,F is an m-dimensional subspace. On the other hand, if we are given a subspace A of dimensions $n\times m$, we may use Gauss-Jordan elimination method to turn A into a matrix of form F, without changing the column space of A. Finally, we need to show that there is no free variable in F. Note that all the variables are in $R_{(n-m)\times m}$. Let's assume that for R_1 and R_2 ($R_1 \neq R_2$), F_1 and F_2 are the same subspace in $\mathbb{R}^n$, therefore, $F_1 \sim F_2$ (see the previous paragraph for the definition of the equivalence). Hence, we may write

$$\exists P : F_1 P = F_2 \rightarrow \begin{cases} P = I_m \\ R_1 P = R_2 \end{cases}$$

which means that $R_1 = R_2$. Therefore, all the variables in R part of matrix F are independent and, consequently, the dimensions of $G(m,\mathbb{R}^n)$ is the dimensions of R, which is $(n-m)\times m$.

$G(m,\mathbb{R}^n)$ is a $m\times(n-m)$ dimensional manifold, which means that it locally resembles $\mathbb{R}^k$, when $k = m\times(n-m)$. In $\mathbb{R}^k$, there are different distances defined. The most famous of them all is the Euclidean distance measurement. Although we can take $G(m,\mathbb{R}^n)$ back to $\mathbb{R}^k$, according to a continuous map, in order to find the distance between two subspaces, it is more desirable to be able to define a distance, that can be calculated in $G(m,\mathbb{R}^n)$. We would like to use the vectors, defining each subspace, to define a distance. A well-known dissimilarity measure on $G(m,\mathbb{R}^n)$ is called "The Geodesic Distance". If $\mathbb{X},\mathbb{Y}\in G(m,\mathbb{R}^n)$ are two $n\times m$ orthonormal matrices, then the singular values of the matrix $\mathbb{X}^T\mathbb{Y}$ give what is required.
The geodesic distance is defined as the following.

$$d_g(\mathbb{X},\mathbb{Y}) = \left(\sum_{i=1}^{m}\theta_i^2\right)^{\frac{1}{2}} \quad (50)$$

θ_i , $(i = 1,2,...,k)$ are called principal angles, that may be defined according to a recursive relation. Some detail about the definition is given in a paper by Y. M. Lui et al. [6]. A reasonable property of principal angles is that they are invariant with respect to any orthogonal transformations of the whole space. We do not concern ourselves with the definition of principal angles here. Instead, we would like to mention that, according to a paper by A. Bjorck and G.H. Golub [7], principal angles can be efficiently computed, using Singular Value decomposition (SVD). The process of calculating principal angles, given orthonormal $\mathbb{X}, \mathbb{Y} \in G(m, \mathbb{R}^n)$, is as the following. Find singular values of $\mathbb{X}^T\mathbb{Y}$ and call them σ_i. Then, the i-th principal angle θ_i is given by the following relation.

$$cos\theta_i = \sigma_i \ (51)$$

Note that the reason we need $\mathbb{X}, \mathbb{Y} \in G(m, \mathbb{R}^n)$ to be orthonormal is to have the singular values $0 \leq \sigma_i < 1$. Consequently, equation (51) would have a solution for θ_i.

As a final remark on geodesic distance, we would like to point out that there are m parameters, used in the calculation of $d_g(\mathbb{X}, \mathbb{Y})$. Although one expects to see a distance of $k = m \times (n - m)$ parameters, as in ordinary Euclidean spaces, $d_g(\mathbb{X}, \mathbb{Y})$is based on less number of parameters. To explain the fallacy, one may argue that θ_i, $\forall i$, that are used in the definition of Geodesic distance, depend on parameters, found in the matrices $\mathbb{X}$ and $\mathbb{Y}$, for which the distance is calculated.

We may continue with a simple example to appreciate the idea of Grassmannian manifolds.
Let's assume all the linear subspaces of dimension 1 in $\mathbb{R}^2$. The symbol to indicate the space is $G(1,\mathbb{R}^2)$. Each subspace (or line) can be thought as an abstract object. Then, the distance between two lines is defined to be the smallest angle (in radian) between them. More formally, if $\mathbb{X}$ and $\mathbb{Y}$ are two lines in $\mathbb{R}^2$ and $x \in \mathbb{X}, y \in \mathbb{Y}$, then the distance between $\mathbb{X}$ and $\mathbb{Y}$ is as below.

$$d(\mathbb{X}, \mathbb{Y}) = cos^{-1}\frac{|\langle x, y\rangle|}{||x||.||y||} \ (52)$$

$\langle x, y\rangle$ is the inner product of the two vectors x and y. Also, it is required to have both x and y not to be equal to 0 vector. One can see that $G(1,\mathbb{R}^2)$, together with the defined distance, make a metric space. The distance is always non-negative. If $\mathbb{X} = \mathbb{Y}$, then

$$d(\mathbb{X}, \mathbb{Y}) = cos^{-1}\frac{|\langle x, x\rangle|}{||x||.||x||} = 0$$

If $d(\mathbb{X}, \mathbb{Y}) = 0$ and $\mathbb{X} \neq \mathbb{Y}$, then $x \neq y$ and

$$d(\mathbb{X}, \mathbb{Y}) = cos^{-1}\frac{|\langle x, y\rangle|}{||x||.||y||} = 0$$

Therefore,

$$|\langle x, y\rangle| = ||x||.||y||$$

According to Cauchy-Schwartz inequality and knowing $x \neq y$, we cannot have the above equality to be true.

The symmetry of the distance is trivial. For the triangle inequality, instead of using the defined distance, we may consider the acute angle between lines. This way it becomes rather intuitive that the triangle inequality also hold.

The metric $\left(G(1,\mathbb{R}^2), d(\mathbb{X}, \mathbb{Y})\right)$ ($d(\mathbb{X}, \mathbb{Y})$ defined in (52)) resembles a circle, from the topology point of view. That is why $G(1,\mathbb{R}^2)$ can be thought as a 1-dimensional manifold. To prove the claim, we need to find, for each open set $U \subset G(1,\mathbb{R}^2)$, an open subset $V \subset \mathbb{R}$ and a one-to-one map $\phi : U \rightarrow V$, such that ϕ is continuous, both ways, and it is differentiable. First, it is reminded that an open sphere of radius ϵ, centred at $\mathbb{X} \in G(1,\mathbb{R}^2)$, is

$$S_\epsilon(\mathbb{X}) = \{\mathbb{Y} \in G(1,\mathbb{R}^2) \,|\, d(\mathbb{X}, \mathbb{Y}) < \epsilon\}$$

Possible open sets on $G(1,\mathbb{R}^2)$ are of form $S_\epsilon(\mathbb{X})$ or the union of them. So, an arbitrary $S_\epsilon(\mathbb{X})$ is treated. If $x \in \mathbb{X}$, then we may take $(-\epsilon, \epsilon)$ to be the open set on R. $|\epsilon|$ is assumed to be less than $\frac{\pi}{2}$. The function ϕ is defined as

$$\phi(a) = \begin{bmatrix} cosa & -sina \\ sina & cosa \end{bmatrix} x \ , \ -\epsilon < a < \epsilon$$

$\phi(a)$ is a vector that represents its line space. Surely, the function $\phi(a)$ is one-to-one, as $-\frac{\pi}{2} < a < \frac{\pi}{2}$. It can also be proven to be continuous, both ways (ϕ and ϕ^{-1}).

As another example assume $G(1,\mathbb{R}^3)$. In a similar manner as $G(1,\mathbb{R}^2)$, one can deduce that $G(1,\mathbb{R}^3)$ is 2-dimensional manifold. The informal reason, why it cannot be a one-dimensional manifold, is that in $\mathbb{R}^3$ there is 2 degrees of freedom, when going from one line to another line. Also, one can go through a similar reasoning to show that $G(2,\mathbb{R}^3)$ is a Grassmannian manifold of dimension 2. $G(2,\mathbb{R}^3)$ and $G(1,\mathbb{R}^3)$ resemble a sphere, from a topological point of view.
Let's elaborate more on $G(1,\mathbb{R}^3)$. $\mathbb{X} \in G(1,\mathbb{R}^3)$ is a line in $\mathbb{R}^3$ and it is the abstract object to consider. We may use the previously defined metric (52) to serve as a metric in $G(1,\mathbb{R}^3)$.

Note that x is a member of the subspace $\mathbb{X}$ and y is a member of the subspace $\mathbb{Y}$ and, as before $x \neq 0$ and $y \neq 0$. It can be proven that $\big(G(1,\mathbb{R}^3), d(\mathbb{X},\mathbb{Y})\big)$ is a metric space. Any metric space has a topology and, as claimed earlier, the topology is a sphere, which is a 2-dimensional manifold. In order to prove, for each open set $V \subset G(1,\mathbb{R}^3)$, we need to find an open set $U \subset \mathbb{R}^2$ and a map $\phi : U \to V$, such that ϕ is continuous, both ways, and it is differentiable. Assume

$$V = S_\epsilon(\mathbb{X})$$

Then, the function is defined as below.

$$\phi\left(\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}\right) = R_y(\beta)R_z(\alpha)R_y(\theta_2)R_x(\theta_1)\begin{bmatrix}0\\0\\1\end{bmatrix}$$

where

$$R_x(\theta) = \begin{bmatrix}1 & 0 & 0\\ 0 & cos\theta & -sin\theta\\ 0 & sin\theta & cos\theta\end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

are rotation matrices in $\mathbb{R}^3$. α and β are the angles, for which a vector $[0,0,1]^T$ is rotated to vector $x \in \mathbb{X}$. The corresponding open set, on $\mathbb{R}^2$, is $S_\epsilon(0)$.

6-2- SLVQ, using Grassmannian manifolds and Geodesic distance

As in section 2, we are given a classification problem in $\mathbb{R}^n$. In section 5, tangent distances were used as distance measurement, because the Euclidean distance does not have enough flexibility to deal with most real world problems. However, dealing with tangent distances is a difficult task, for we usually do not know the class-preservative manifolds on $\mathbb{R}^n$. Reviewing recent papers on the application of Grassmannian manifolds in machine learning, gives us hope to apply the theory to SLVQ. The papers are written by S. Chepushtanova and M. Kirby [5][8] and Y. M. Lui, J. R. Beveridge and M. Kirby [6].

The idea is as follows. We are given a training set T. T is going to be partitioned in subsets T^j, for $j \in L$. The subsets are define as below.

$$T^j = \{(\mathbb{X}, y) \in T \mid y \in j\} \quad (53)$$

Consequently, we have the following facts.

$$\bigcup_j T^j = T$$
$$T^j \cap T^i = \phi \ \forall i, j : i \neq j$$

The vector set of a set T^j is defined to be the matrix produced by tacking all the first elements of pairs, that exist in T^j. Formally

$$\{V(T^j)\}_{:,i} = \mathbb{X}_i \ : \ (\mathbb{X}_i, j) \in T^j \ (54)$$

If $\mathbb{R}^n$ is a low-dimensional space or the number of sample points in $T^j \, \forall j$ is low, then we may take $V(T^j)$ to be a representative for the space spanned by $V(T^j)$. In this case, if we are given a new data point $x \in \mathbb{R}^n$, to be classified, we can find the distance of x to all the subspaces $V(T^j) \forall j$ and choose the closest as the correct class. This scenario, however, does not happen often, as real-world problems are in large-dimensional spaces and the number of data samples, given in a training set T, is relatively large. The solution is to further partition T^j subsets to smaller subsets. The new partitions would be denoted by a subscript. Therefore, T^j_i is the i-th partition of T^j. At this point, we do not know the dimensions m to be taken for a subspace of $\mathbb{R}^n$, which is going to be an abstract point in the Grassmannian manifold. However, we certainly know that we would like T^j_i to have the same dimensions, for all j and i. Because, we want all $T^j_i \, \forall i, j$ to be a point in $G(m, \mathbb{R}^n)$. This way, they are more meaningfully comparable.

The next step, would be to define prototypes in $G(m, \mathbb{R}^n)$. They would be similar to what was defined earlier, with the difference that they are $n \times m$ matrices now. The prototypes would be denoted as before, using the notation $\mathbb{W}^j_i$, for the i-th prototype of class j. $\mathbb{W}^j_i$ would be filled with vectors of $V(T^j)$.

In order to find the gradient, the partial derivatives of the distance between two objects $\mathbb{W} \in G(m, \mathbb{R}^n)$ and $\mathbb{X} \in G(m, \mathbb{R}^n)$, with respect to T, is calculated.

$$d_G(\mathbb{W}, \mathbb{X}) = \left(\sum_{i=1}^{m} \theta_i^2 \right)^{\frac{1}{2}}$$

$$\frac{\partial d_G(\mathbb{W}, \mathbb{X})}{\partial \{\mathbb{W}\}_{ab}} = \left(\sum_i \frac{\partial d_G(\mathbb{W}, \mathbb{X})}{\partial \theta_i} \times \frac{\partial \theta_i}{\partial \{\mathbb{W}\}_{ab}} \right) \ (55a)$$

To find (55a), firstly the following derivative is found.

$$\frac{\partial d_G(\mathbb{W}, \mathbb{X})}{\partial \theta_i} = \left(\sum_{i=1}^{m} \theta_j^2 \right) \theta_i \ (55b)$$

Secondly, we assume the SVD $\mathbb{W}^T\mathbb{X} = U\Sigma V^T$. Consequently

$$\Sigma = U^T\mathbb{W}^T\mathbb{X}V$$

σ_i, the i-th element on the diagonal of Σ, is

$$\sigma_i = \{\Sigma\}_{ii} = \{U^T\}_{i,:}\mathbb{W}^T\mathbb{X}\{V\}_{:,i}$$

therefore, the last piece to find (55a) is as below.

$$\frac{\partial\theta_i}{\partial\mathbb{W}} = \frac{\partial\theta_i}{\partial\sigma_i} \times \frac{\partial\sigma_i}{\partial\mathbb{W}} = -\frac{1}{\sqrt{1-\sigma_i^2}}\,.\,\mathbb{X}\,.\,\{V\}_{:,i}\,.\,\{U^T\}_{i,:}\ (55c)$$

Having (55a) to (55c), the gradient is

$$\frac{\partial d_G(\mathbb{W},\mathbb{X})}{\partial\mathbb{W}} = -\left(\sum_{j=1}^{m}\theta_j^2\right)\left(\sum_i \theta_i \frac{\mathbb{X}\,.\,\{V\}_{:,i}\,.\,\{U^T\}_{i,:}}{\sqrt{1-\sigma_i^2}}\right)\ (56a)$$

We may rewrite the above expression as

$$\frac{\partial d_G(\mathbb{W},\mathbb{X})}{\partial \mathbb{W}} = -\,d_G(\mathbb{W},\mathbb{X})\left(\sum_i arccos\sigma_i \frac{\mathbb{X}\,.\,\{V\}_{:,i}\,.\,\{U^T\}_{i,:}}{\sqrt{1-\sigma_i^2}}\right) \ (56b)$$

Finally, the most compact form is

$$\frac{\partial d_G(\mathbb{W},\mathbb{X})}{\partial \mathbb{W}} = -\,d_G(\mathbb{W},\mathbb{X})\mathbb{X}VS(\Sigma)U^T \ (56c)$$

$S(\Sigma)$ is obtained by applying the function $\frac{arccosx}{\sqrt{1-x^2}}$ to the diagonal elements of the matrix Σ.

We would like to define the probability that $\mathbb{X} \in G(m,\mathbb{R}^n)$ is produced, when prototype $\mathbb{W}_i^j \in G(m,\mathbb{R}^n)$ is given the chance to produce, as below.

$$p(\mathbb{X}\,|\,\mathbb{W}_i^j,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}exp\left(-\frac{d_G^2(\mathbb{W},\mathbb{X})}{2\sigma^2}\right)$$

Also, it is pointed out that the probability that a data point $\mathbb{X}$ is produced by prototypes of the same label and the probability that a data point $\mathbb{X}$ is produced by prototypes of different labels are defined as in relations (6) and (7). The only difference is the new geodesic distance, that has been substituted.

Now the gradient of Grassmannian Soft Learning Vector Quantisation, or in short GSLVQ, is as the following.

$$\frac{\partial}{\partial \mathbb{W}_i^j} log \frac{p(\mathbb{X},y\,|\,W)}{p(\mathbb{W},\bar{y}\,|\,W)} =$$

$$\left(\delta(j-y)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},y\,|\,W)} - \big(1-\delta(j-y)\big)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},\bar{y}\,|\,W)}\right)\left(d_G^2(\mathbb{W}_i^j,\mathbb{X})\mathbb{X}VS(\Sigma)U^T\right)$$

As the whole numerical computation is based on the SVD of $\mathbb{W}^T\mathbb{X}$, the factor $d_G^2(\mathbb{W}_i^j,\mathbb{X})$ would be written in terms of Σ.

$$d_G^2(\mathbb{W}_i^j,\mathbb{X}) = tr(arccos^2(\Sigma)) = R(\Sigma)$$

In the relation above, $arccos\mathbb{X}$ of a matrix $\mathbb{X}$, with all the diagonal element less than 1 and greater than or equal to 0, is obtained by applying $arccos(x)$ to all the diagonal elements of $\mathbb{X}$, while letting the off-diagonal elements to stay the same as before. $tr(\mathbb{X})$, denotes the trace of a matrix $\mathbb{X}$. With the new changes, the gradient is further simplified as below.

$$\frac{\partial}{\partial \mathbb{W}_i^j} log \frac{p(\mathbb{X},y\,|\,W)}{p(\mathbb{W},\bar{y}\,|\,W)} =$$

$$\left(\delta(j-y)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},y\,|\,W)} - \big(1-\delta(j-y)\big)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},\bar{y}\,|\,W)}\right)\left(R(\Sigma)\mathbb{X}VS(\Sigma)U^T\right) \quad (57)$$

In order to test if the updated prototype matrix is orthonormal, the following calculation is done.

$$\big(\mathbb{W}_i^j(t+1)\big)^T\big(\mathbb{W}_i^j(t+1)\big) = \left(\big(\mathbb{W}_i^j(t)\big)^T + \alpha US(\Sigma)V^T\mathbb{X}^T\right)\left(\mathbb{W}_i^j(t) + \alpha\mathbb{X}VS(\Sigma)U^T\right)$$
$$= I + \alpha U\Sigma S(\Sigma)U^T + \alpha US(\Sigma)\Sigma U^T + \alpha^2 US^2(\Sigma)U^T = I + \alpha U\big(2\Sigma S(\Sigma) + \alpha S^2(\Sigma)\big)U^T$$

In the above calculation, we have

$$\alpha = \left(\delta(j-y)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},y\,|\,W)} - \big(1-\delta(j-y)\big)\frac{p_i^j p(\mathbb{X}\,|\,\mathbb{W}_i^j)}{\sigma^2 p(\mathbb{X},\bar{y}\,|\,W)}\right)R(\Sigma)$$

It seem that the updated prototypes are not necessarily orthonormal, as $\alpha U\big(2\Sigma S(\Sigma) + \alpha S^2(\Sigma)\big)U^T \neq 0$. Consequently, after each learning step, the prototypes must be put in the orthonormal form, as it is necessary to have them orthonormal, when calculating the principal angles. A good news, however, is that the updated prototypes would be full rank. In order to draw such a conclusion, we may assume that

$$\left(\mathbb{W}_i^j(t+1)\right)^T\left(\mathbb{W}_i^j(t+1)\right) = I + \alpha U\left(2\Sigma S(\Sigma) + \alpha S^2(\Sigma)\right)U^T$$

Is not full rank. Then, there should be a non-zero vector $x \in \mathbb{R}^m$, such that

$$\left(I + \alpha U\left(2\Sigma S(\Sigma) + \alpha S^2(\Sigma)\right)U^T\right)x = 0$$

It can be rewritten as below

$$\left(\alpha U\left(2\Sigma S(\Sigma) + \alpha S^2(\Sigma)\right)U^T\right)x = -x$$

which means that x is an eigenvector of $\alpha U\left(2\Sigma S(\Sigma) + \alpha S^2(\Sigma)\right)U^T$, with its eigenvalue to be -1. However, the matrix is positive definite, as $2\Sigma S(\Sigma) + \alpha S^2(\Sigma)$ is a diagonal matrix, with all diagonal entries to be positive numbers, and U is an orthonormal matrix. Positive definite matrices do not have negative eigenvalues and, hence, the matrix is full rank, which implies that $\mathbb{W}_i^j(t+1)$ has a full column rank.
We would like to emphasise that the learning method can be implemented practically if the SVD of the matrix $\mathbb{W}^T\mathbb{X}$ is efficiently computable, for all members of $G(m, \mathbb{R}^n)$. The SVD would provide the matrices V, U and Σ, that are explicitly used in the gradient.

Assuming the learning process is over, one would like to see how a new data $x \in \mathbb{R}^n$ would be classified. A new data x, to be classified, would be a vector in $\mathbb{R}^n$, not a subspace. Hence, we may take two approaches, at this point. The first alternative is to calculate the norm of the image of x on every prototype $\mathbb{W}_i^j \forall i, j$ and the “winner takes all” rule would be applied to find the class of x. In other words, if all prototypes are orthonormalised, then class $F(x)$ of $x \in \mathbb{R}^n$ is

$$F(x) = arg_j\left(min_{i,j}||(\mathbb{W}_i^j)^T x||\right)$$

The second alternative for classifying a new data point $x \in \mathbb{R}^n$ is applicable to problems, for which a group of class preservative transformations is given. Having the transforms, one may produce $m - 1$ new samples of x, with the same class. Then, $x$and its transformed versions are put in a $n \times m$ matrix $\mathbb{X}$, which makes an abstract point in $G(m, \mathbb{R}^n)$. Consequently, geodesic distance can be used to compare $\mathbb{X}$ to all prototypes $\mathbb{W}_i^j$ and, as in the previous alternative, "winner takes all" would introduce the class of $\mathbb{X}$ (or x).

Intuitively, the power of Grassmannian manifolds lies in the fact that data points are taken to a higher dimensional manifold. This is the trick for many machine learning techniques, as higher dimensional spaces are harvested to achieve a better separation of data. Knowing that the dimensions of a Grassmannian manifold is $m \times (n - m)$, it is tempting to have $m = \frac{n}{2}$, to maximise the dimensions of the manifold. This intuition can be backed up by practical experiments, provided by S. Chepushtanova and M. Kirby [5], as the authors state that increasing the dimensions of Grassmannian manifold would make the classification accuracy to tent %100.

6-3- Validity of the result of GSLVQ

In this part, few issues, that may challenge the validity of the result of GSLVQ, are discussed.

A network, trained by GSLVQ, would classify any scalar multiplication of an input $x \in \mathbb{R}^n$, in the same class as x itself. This is an acceptable result in a certain type of problems. For instance, if the task is to classify hand-written grayscale characters, the average intensity of an image is not a parameter of interest, as we are supposed to find the structure of characters. Therefore, an image would be the same as itself, multiplied by a constant α. This type of problems can be called "scale invariant problems". One should always check if GSLVQ is applied to a scale invariant problem.

Another problem is to know how good a subspace, generated by a set of samples of the same class, can approximate or contain a class-preservative manifold. Obviously, if a character and its rotated version is given, in general, it is not possible to obtain another rotated version, by simply making a linear combination of the first two samples. Also, a combination of characters, with the same class, may invade the realm of characters of different classes.

In order to deal with the lastly mentioned problem, we need to avoid creating training data samples, in $G(m, \mathbb{R}^n)$, randomly. To create better data samples, we would like to make each abstract point out of samples, of the same class in T, that are relatively close to each other, if the measure of closeness is the angle between the samples. In other words, abstract point are made to be local. To achieve this goal, we may apply neural gas learning or unsupervised LVQ to data points of each class (T^j) separately. The process is described in the next paragraph.

A single $V(T^j)$ is taken at a time. All the members of $V(T^j)$ would be normalised. Assume that m, the dimension of the Grassmannian manifold, is known. Prototypes $W = \{\mathbb{W}_i\}_{i=1}^{d}$, $\mathbb{W}_i \in \mathbb{R}^n$, when $d = \dfrac{|V(T^j)|}{m}$, are randomly initiated. Note it was mentioned earlier that $|V(T^j)|$ would be set in such a way that it is a multiple of m. Also, we would like the prototypes in W to be orthonormal vectors. The distance between a vector $x \in \mathbb{R}^n$ and a prototype $\mathbb{W}_j$ would be the Euclidean distance.

According to an online learning scheme, given that a vector $v \in V(T^j)$ is provided to the network, neural gas learning is utilised to update the prototypes. Each prototype should be normalised, before passing to the next learning step. For a theoretical treatment of neural gas, the reader is referred to a paper by T. Martinetz et al. [9].

After the learning phase, the m closest vectors, in $V(T^j)$, to the prototype $\mathbb{W}_i$ are put in one set. The set is nothing but one abstract point in $G(m, \mathbb{R}^n)$ that trains the main GSLVQ problem.

7- Conclusion

In this article, tangent distances and Geodesic distances, defined in the framework of Grassmannian manifolds, were utilised as dissimilarity measurements to replace the Euclidean distance, that was used originally by S. Seo and K. Obermayer [1] in SLVQ and RSLVQ. When using tangent distances, we need to make sure that the class-preserving manifolds are differentiable. However, Grassmannian manifolds have the advantage of not being dependent on the differentiability of the manifolds.

As the first alternative, single-sided tangent distance were integrated in SLVQ and RSLVQ to make TSLVQ, TSLVQ_S, TRSLVQ, TRSLVQ_S, TDSLVQ, and TDRSLVQ. Each of them were established under different conditions and assumptions. Some of the key conditions would be as the following. The first condition is whether we would like to use strong or weak approach (strong and weak approaches are explained in section 5-1). The second condition is whether we use Saralajew-Villmann assumption or Simard assumption. The last condition is whether the tangent space (necessary to find single-sided tangent distance) is made on the data side, while prototypes would act as points, or we prefer to have tangent spaces on the prototype side and treat data points as points.

Each of the methods, based on tangent distances, have advantages and disadvantages. But, as expected theoretically and proven practically, TRSLVQ_S is the best methods among the all methods, that are established in this article. It is worth mentioning that TSLVQ_S and TRSLVQ_S are the only methods (between the rest of the methods, described in this paper) that show a stable behaviour in real problems, such as hand-written characters classification.

Regardless of the new dissimilarity measurements, new ways of looking at basic SLVQ is provided in this article, in section 3-2. ASLVQ is the fruit of the investigation, which comes with a better probabilistic model. However, it may have problems with a small class of classification problems, as it dismisses the repulsion mechanism.

Finally, it is mentioned that, although the theory of GSLVQ, including the corresponding learning rules, was provided in section 6, the corresponding Matlab programs were not written, due to lack of time and computational resources. Therefore, GSLVQ is to be tested practically.

8- Matlab Programs

Program 1 – SLVQ function

```
%%% SLVQ (Soft Learning Vector Quantization)

%% It is a function that gets a training set "T", prototypes (labelled) "prototype", the number of classes %%"c", maximum number of learning steps "max"and outputs the updated prototypes, after some steps

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as

%%training set matrix T

function [prototype,error_rate,error_rate0]=SLVQ(T,prototype,test,c,max,u)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T);
[p1,p2]=size(prototype);
[ts1,ts2]=size(test);
%%Define the Gaussian distribution standard deviation SD=1;

%%misclassification error before learning

error=0; for i=1:ts2

nearest_dist=sum((prototype(1:end-1,:)-repmat(test(1:end-1,i),1,p2)).^2);
[C1,I1]=min(nearest_dist);
if prototype(end,I1)~=test(end,i)

error=error+1;

end end
```

```
error_rate0=error/ts2;

%%the recursive adapting part

uu=u;
for t=1:max

%u=uu/t;

u=u*(0.999);

%%randomly choose a training data from T

x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prottoypes accordingly partial_sum=0;

partial_sum_w=0; for j=1:p2

if prototype(end,j)==x_label d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
partial_sum=partial_sum+normpdf(d,0,SD);

end
if prototype(end,j)~=x_label

d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));

partial_sum_w=partial_sum_w+normpdf(d,0,SD);

end end

for j=1:p2 d=sqrt(sum((prototype(1:end-1,j)-x_main).^2)); p=normpdf(d,0,SD);
if prototype(end,j) == x_label

prototype(1:end-1,j)=prototype(1:end-1,j)+u*(1/(SD)^2)*(p/partial_sum)*(x_main-
prototype(1:end-1,j));

end
if prototype(end,j) ~= x_label

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(1/(SD)^2)*(p/partial_sum_w)*(x_main-
prototype(1:end-1,j));

end end

end

%%misclassification rate

error=0; for i=1:ts2

nearest_dist=sum((prototype(1:end-1,:)-repmat(test(1:end-1,i),1,p2)).^2);
[C1,I1]=min(nearest_dist);
if prototype(end,I1)~=test(end,i)

error=error+1;
```

```
end end

error_rate=error/ts2;
```

Program 2 – ASLVQ function

```
%%ASLVQ (Attraction SLVQ)

%% It is a function that gets a training set "T", prototypes (labelled) "prototype", the number of classes %%"c", maximum number of learning steps "max"and outputs the updated prototypes, after some steps

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=ASLVQ(T,prototype,test,c,max,u)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T); [p1,p2]=size(prototype); [ts1,ts2]=size(test); RR=p2/T2;

%%Define the Gaussian distribution standard deviation

SD=1;

%%misclassification error before learning

error=0; for i=1:ts2

nearest_dist=sum((prototype(1:end-1,:)-repmat(test(1:end-1,i),1,p2)).^2);
[C1,I1]=min(nearest_dist);
if prototype(end,I1)~=test(end,i)

error=error+1;

end end

error_rate0=error/ts2;

%%the recursive adapting part

uu=u;
for t=1:max

%u=uu/t;

u=u*(0.999);

%%randomly choose a training data from T
```

```
x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prottoypes accordingly partial_sum=0;

for j=1:p2
if prototype(end,j)==x_label

d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));

partial_sum=partial_sum+normpdf(d,0,SD);

end end

partial_sum=RR*partial_sum; for j=1:p2

d=sqrt(sum((prototype(1:end-1,j)-x_main).^2)); p=RR*normpdf(d,0,SD);
if prototype(end,j) == x_label

prototype(1:end-1,j)=prototype(1:end-1,j)+u*(1/(SD)^2)*(p/(partial_sum-
partial_sum^2))*(x_main- prototype(1:end-1,j));

end end

end

%%misclassification rate

error=0; for i=1:ts2

nearest_dist=sum((prototype(1:end-1,:)-repmat(test(1:end-1,i),1,p2)).^2);
[C1,I1]=min(nearest_dist);
if prototype(end,I1)~=test(end,i)

error=error+1;

end end

error_rate=error/ts2;
```

Program 3- Butterfly problem

```
%%%butterfly data production

c=2;
PPC=20;
%%make prototypes prototype=random('unif',-1,1,2,PPC*c); labb=repmat(1:c,PPC,1);
lab=labb(:)'; prototype=vertcat(prototype,lab); %%make training data
TS=1000;

TT=random('unif',-2,2,2,TS); labb=sign(TT(1,:).*TT(2,:)); labb=(labb./2)+3/2;
T=vertcat(TT,labb);
%%make test data
TS=1000;
TT=random('unif',-2,2,2,TS); labb=sign(TT(1,:).*TT(2,:)); labb=(labb./2)+3/2;
```

```
test=vertcat(TT,labb);
%%other parameters for SLVQ function max=100000;

u=0.1;

%%Normal SLVQ

[prototype_out,errorS,errorS0]=SLVQ(T,prototype,test,c,max,u); errorS0
errorS

%%New ASLVQ

[prototype_out,errorA,errorA0]=ASLVQ(T,prototype,test,c,max,u); errorA0
errorA
```

Program3 – RSLVQ

```
%%% RSLVQ (Robust Soft Learning Vector Quantization)

%% It is a function that gets a training set "T", prototypes (labelled) "prototype", the number of
classes %%"c", maximum number of learning steps "max"and outputs the updated prototypes, after
some steps

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T
would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the
label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=RSLVQ(T,prototype,test,c,max,u)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T);
[p1,p2]=size(prototype);
[ts1,ts2]=size(test);
%%Define the Gaussian distribution standard deviation SD=1;

%misclassification error before learning version 1 % error=0;

    %  for i=1:ts2


    %  nearest_dist=sum((prototype(1:end-1,:)-repmat(test(1:end-1,i),1,p2)).^2);


    %  [C1,I1]=min(nearest_dist);


    %  if prototype(end,I1)~=test(end,i)
    % error=error+1;

    % end

    % end

    % %%misclassification error before learning version 2 error=0;
    for i=1:ts2
    x=test(:,i); x_main=x(1:end-1); x_label=x(end); partial_sum_w=0; partial_sum=0;
    for j=1:p2
    if prototype(end,j)==x_label
    d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
    partial_sum=partial_sum+normpdf(d,0,SD);
    end
    if prototype(end,j)~=x_label
    d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
    partial_sum_w=partial_sum_w+normpdf(d,0,SD);
    end end
    if partial_sum_w>partial_sum error=error+1;
    end end
    error_rate0=error/ts2;
    %%the recursive adapting part
    uu=u;
    for t=1:max
    %u=uu/t;
    u=u*(0.999);
    %%randomly choose a training data from T
    x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
    %%update prottoypes accordingly partial_sum=0;
    partial_sum_w=0; for j=1:p2
    if prototype(end,j)==x_label d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
    partial_sum=partial_sum+normpdf(d,0,SD);
    end
    if prototype(end,j)~=x_label
    d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
    partial_sum_w=partial_sum_w+normpdf(d,0,SD);
    end end


for j=1:p2 d=sqrt(sum((prototype(1:end-1,j)-x_main).^2)); p=normpdf(d,0,SD);
if prototype(end,j) == x_label

prototype(1:end-1,j)=prototype(1:end-1,j)+u*(1/(SD)^2)*p*(1/partial_sum- 1/
(partial_sum+partial_sum_w))*(x_main-prototype(1:end-1,j));

end
if prototype(end,j) ~= x_label

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(1/(SD)^2)*(p/
(partial_sum+partial_sum_w))*(x_main- prototype(1:end-1,j));
```

```
end end

end

%%misclassification rate version 1

        %  error=0;

        %  for i=1:ts2

        %  nearest_dist=sum((prototype(1:end-1,:)-repmat(test(1:end-1,i),1,p2)).^2);

        %  [C1,I1]=min(nearest_dist);

        %  if prototype(end,I1)~=test(end,i)

        %  error=error+1;

        %  end

        %  end

        %  %%misclassification error before learning version 2
        error=0; for i=1:ts2
        x=test(:,i); x_main=x(1:end-1); x_label=x(end); partial_sum_w=0; partial_sum=0;
        for j=1:p2
        if prototype(end,j)==x_label
        d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
        partial_sum=partial_sum+normpdf(d,0,SD);
        end
        if prototype(end,j)~=x_label
        d=sqrt(sum((prototype(1:end-1,j)-x_main).^2));
        partial_sum_w=partial_sum_w+normpdf(d,0,SD);
        end end
        if partial_sum_w>partial_sum error=error+1;
        end end
        error_rate=error/ts2;
```

Program 4 – TSLVQ

```
%%% TSLVQ (Tangent Soft Learning Vector Quantization)

%% It is a function that gets a training set, prototypes (labelled) and the number of classes %%and
outputs a the updated prototypes
```

```
%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T
would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the
label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=TSLVQ(T,prototype,test,c,max,u,dim)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T); [p1,p2]=size(prototype);

%%Define the Gaussian distribution standard deviation

SD=0.3;

%%define the dimensions of tangent planes

tan_dim=dim;

%%define tangent planes for each prototype

A=random('unif',-1,1,p1-1,tan_dim,p2); %%normalize the basis
for i=1:p2

A(:,:,i)=orth(A(:,:,i));

end

%%find the error rate, before training

[test1,test2]=size(test); error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); if
d<d0

d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate0=error/test2;

%%the recursive adapting part
```

```
uu=u;
for t=1:max

u=uu/t;

%%randomly choose a training data from T

x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prottoypes accordingly partial_sum=0;

partial_sum_w=0; for j=1:p2

%%find projection matrix PM

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; if prototype(end,j)==x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));

partial_sum=partial_sum+normpdf(d,0,SD);

end
if prototype(end,j)~=x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));

partial_sum_w=partial_sum_w+normpdf(d,0,SD);

end end

for j=1:p2
%%find projection matrix PM PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); p=normpdf(d,0,SD);
if prototype(end,j) == x_label

prototype(1:end-1,j)=prototype(1:end-1,j)+u*(1/(SD)^2)*(p/partial_sum)*PM*PM*(x_main-
prototype(1:end- 1,j));

A(:,:,j)=A(:,:,j)+2*u*(1/(SD)^2)*(p/partial_sum)*(x_main-prototype(1:end-1,j))*(x_main-
prototype(1:end- 1,j))'*PM*A(:,:,j);

end
if prototype(end,j) ~= x_label

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(1/(SD)^2)*(p/partial_sum_w)*PM*PM*(x_main-
prototype(1:end- 1,j));

A(:,:,j)=A(:,:,j)-2*u*(1/(SD)^2)*(p/partial_sum_w)*(x_main-prototype(1:end-1,j))*(x_main-
prototype(1:end- 1,j))'*PM*A(:,:,j);

end

A(:,:,j)=orth(A(:,:,j));

end end
```

```
%%find the error rate, after training

error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); if
d<d0

d0=d;

class=prototype(end,j);

end

end
if class~=test(end,i)

error=error+1;

end end

error_rate=error/test2;
```

Program 5 – TRSLVQ

```
%%% TRSLVQ (Tangent Robust Soft Learning Vector Quantization)

%% It is a function that gets a training set, prototypes (labelled) and the number of classes %%and
outputs a the updated prototypes

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T
would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the
label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=TRSLVQ(T,prototype,test,c,max,u,dim)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T); [p1,p2]=size(prototype);

%%Define the Gaussian distribution standard deviation

SD=0.3;

%%define the dimensions of tangent planes

tan_dim=dim;

%%define tangent planes for each prototype
```

```
A=random('unif',-1,1,p1-1,tan_dim,p2); %%normalize the basis
for i=1:p2

A(:,:,i)=orth(A(:,:,i));

end

%%find the error rate, before training

[test1,test2]=size(test); error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); if
d<d0

d0=d; class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate0=error/test2;

%%the recursive adapting part

uu=u;
for t=1:max

u=uu/t;

%%randomly choose a training data from T

x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prototypes accordingly partial_sum=0;

partial_sum_w=0; for j=1:p2

%%find projection matrix PM

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; if prototype(end,j)==x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));

partial_sum=partial_sum+normpdf(d,0,SD);

end
if prototype(end,j)~=x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));

partial_sum_w=partial_sum_w+normpdf(d,0,SD);
```

```
end end

for j=1:p2
%%find projection matrix PM PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); p=normpdf(d,0,SD);
if prototype(end,j) == x_label

prototype(1:end-1,j)=prototype(1:end-1,j)+u*(1/(SD)^2)*p*(1/partial_sum- 1/
(partial_sum+partial_sum_w))*PM*PM*(x_main-prototype(1:end-1,j));

A(:,:,j)=A(:,:,j)+2*u*(1/(SD)^2)*p*(1/partial_sum-1/(partial_sum+partial_sum_w))*(x_main-
prototype(1:end- 1,j))*(x_main-prototype(1:end-1,j))'*PM*A(:,:,j);

end
if prototype(end,j) ~= x_label

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(1/(SD)^2)*(p/
(partial_sum+partial_sum_w))*PM*PM*(x_main- prototype(1:end-1,j));

A(:,:,j)=A(:,:,j)-2*u*(1/(SD)^2)*(p/(partial_sum+partial_sum_w))*(x_main-
prototype(1:end-1,j))*(x_main- prototype(1:end-1,j))'*PM*A(:,:,j);

end

A(:,:,j)=orth(A(:,:,j));

end end

%%find the error rate, after training

error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); if
d<d0

d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate=error/test2;
```

Program 6 – TDSLVQ

```
%%% TDSLVQ (Tangent Data Soft Learning Vector Quantization)
```

```
%% It is a function that gets a training set, prototypes (labelled) and the number of classes %%and
outputs a the updated prototypes

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T
would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the
label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=TDSLVQ(T,prototype,test,c,max,u,dim)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T); [p1,p2]=size(prototype);

%%Define the Gaussian distribution standard deviation

SD=0.1;
SD2=1;
%%define the dimensions of tangent planes tan_dim=dim;
%%radius of the ball
BR=1;
%%find the error rate, before training [test1,test2]=size(test);

error=0;
for i=1:test2

d0=100000; dears=[];
for w=1:test2

ilds=sqrt(sum((test(1:end-1,i)-test(1:end-1,w)).^2)); if ilds <BR && test(end,i)==test(end,w)

dears=horzcat(dears,test(1:end-1,w));

end end

[A,off]=k_affine(dears,tan_dim); for j=1:p2

PM=eye(p1-1)-A*A'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); if d<d0

d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate0=error/test2;

%%the recursive adapting part
```

```
uu=u;
for t=1:max

u=uu/t;

%%randomly choose a training data from T

x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prototypes accordingly

dears=[];
for w=1:T2

ilds=sqrt(sum((x_main-T(1:end-1,w)).^2)); if ilds <BR && x_label==T(end,w)
dears=horzcat(dears,T(1:end-1,w));
end

end

[A,off]=k_affine(dears,tan_dim);

PM=eye(p1-1)-A*A'; PM2=A*A';

partial_sum=0; partial_sum_w=0;

for j=1:p2
if prototype(end,j)==x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); d2=sqrt(sum((PM2*(prototype(1:end-1,j)-
x_main)).^2)); partial_sum=partial_sum+normpdf(d,0,SD)*normpdf(d2,0,SD2);

end
if prototype(end,j)~=x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); d2=sqrt(sum((PM2*(prototype(1:end-1,j)-
x_main)).^2)); partial_sum_w=partial_sum_w+normpdf(d,0,SD)*normpdf(d2,0,SD2);

end end

for j=1:p2 d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));
d2=sqrt(sum((PM2*(prototype(1:end-1,j)-x_main)).^2)); p=normpdf(d,0,SD)*normpdf(d2,0,SD2);
if prototype(end,j) == x_label

prototype(1:end-1,j)=prototype(1:end-1,j)+u*(1/2)*(p/partial_sum)*((1/SD2)*PM2+(1/
SD)*PM)*(x_main- prototype(1:end-1,j));

end
if prototype(end,j) ~= x_label

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(1/2)*(p/partial_sum)*((1/SD2)*PM2+(1/
SD)*PM)*(x_main- prototype(1:end-1,j));

end end
```

```
end

%%find the error rate, after training

error=0;
for i=1:test2

d0=100000; dears=[];
for w=1:test2

ilds=sqrt(sum((test(1:end-1,i)-test(1:end-1,w)).^2)); if ilds <BR && test(end,i)==test(end,w)

dears=horzcat(dears,test(1:end-1,w));

end end

[A,off]=k_affine(dears,tan_dim); for j=1:p2

PM=eye(p1-1)-A*A'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); if d<d0

d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate=error/test2;
```

program 7 – *k*-affine estimation

```
%%function to use PCA to estimate best k-affine for a set of data points

%%this function receives the data points and k (the dimension of the %%affine) and returns the orthogonal vectors spanning the space, plus the %%offset vector
%%data points X should be given in a matrix, with each column dedicated to %%one data sample

function [space,mean]=k_affine(X,k)

[x1,x2]=size(X);
mean=sum(X');
mean=(1/x2)*(mean');
%%centered data Y=X-repmat(mean,1,x2);
sig=(1/x2)*Y*Y';
[vec,val]=eig(sig);
val_signed=sum(val); val_unsigned=abs(val_signed); [temp,ind]=sort(val_unsigned,'descend');
space=[];

for i=1:k space=horzcat(space,vec(:,ind(i)));

end
```

Program 8 – TSLVQ_S

```
%%% TSLVQ_S (Tangent Soft Learning Vector Quantization strong)

%% It is a function that gets a training set, prototypes (labelled) and the number of classes %%and
outputs a the updated prototypes

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T
would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the
label of the sample. Label is a number from 1 to c

%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=TSLVQ_S(T,prototype,test,c,max,u,dim)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T); [p1,p2]=size(prototype);

%%Define the Gaussian distribution standard deviation

SD=30; SD2=50;

%%define the dimensions of tangent planes

tan_dim=dim;

%%define tangent planes for each prototype

A=random('unif',-1,1,p1-1,tan_dim,p2); %%normalize the basis
for i=1:p2

A(:,:,i)=orth(A(:,:,i));

end

%%find the error rate, before training

[test1,test2]=size(test); error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
PM2=A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2));

d2=sqrt(sum((PM2*(prototype(1:end-1,j)-test(1:end-1,i))).^2));

d=sqrt(d^2+d2^2); if d<d0

d0=d;

class=prototype(end,j);
```

```
end end

if class~=test(end,i) error=error+1;

end end

error_rate0=error/test2;

%%the recursive adapting part

uu=u;
for t=1:max

u=uu/t;

%%randomly choose a training data from T

x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prototypes accordingly partial_sum=0;

partial_sum_w=0; for j=1:p2

%%find projection matrix PM

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; PM2=A(:,:,j)*A(:,:,j)';
if prototype(end,j)==x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); d2=sqrt(sum((PM2*(prototype(1:end-1,j)-
x_main)).^2)); partial_sum=partial_sum+normpdf(d,0,SD)*normpdf(d2,0,SD2);

end
if prototype(end,j)~=x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); d2=sqrt(sum((PM2*(prototype(1:end-1,j)-
x_main)).^2)); partial_sum_w=partial_sum_w+normpdf(d,0,SD)*normpdf(d2,0,SD2);

end

end
for j=1:p2

%%find projection matrix PM

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
PM2=A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));
d2=sqrt(sum((PM2*(prototype(1:end-1,j)-x_main)).^2)); p=normpdf(d,0,SD)*normpdf(d2,0,SD2);

if prototype(end,j) == x_label A(:,:,j)=A(:,:,j)+2.*u.*(p/partial_sum).*((-1/(SD)^2).*(x_main-
prototype(1:end-1,j))*(x_main-prototype(1:end-

1,j))'*PM+(1/(SD2)^2).*(x_main-prototype(1:end-1,j))*(x_main-
prototype(1:end-1,j))'*PM2)*A(:,:,j); prototype(1:end-1,j)=prototype(1:end-1,j)+u*(p/
partial_sum)*((1/(SD)^2)*PM*PM*(x_main-prototype(1:end-

1,j))+(1/(SD2)^2)*PM2*PM2*(x_main-prototype(1:end-1,j)));
```

```
end
if prototype(end,j) ~= x_label

A(:,:,j)=A(:,:,j)-2.*u.*(p/partial_sum_w).*((-1/(SD)^2).*(x_main-prototype(1:end-1,j))*(x_main-
prototype(1:end- 1,j))'*PM+(1/(SD2)^2).*(x_main-prototype(1:end-1,j))*(x_main-
prototype(1:end-1,j))'*PM2)*A(:,:,j);

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(p/partial_sum_w)*((1/(SD)^2)*PM*PM*(x_main-
prototype(1:end- 1,j))+(1/(SD2)^2)*PM2*PM2*(x_main-prototype(1:end-1,j)));

end

A(:,:,j)=orth(A(:,:,j));

end end

%%find the error rate, after training

error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
PM2=A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2));

d2=sqrt(sum((PM2*(prototype(1:end-1,j)-test(1:end-1,i))).^2));

d=sqrt(d^2+d2^2); if d<d0

d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate=error/test2;
```

Program 9 – TRSLVQ_S

```
%%% TRSLVQ_S (Tangent Robust Soft Learning Vector Quantization strong)

%% It is a function that gets a training set, prototypes (labelled) and the number of classes %%and
outputs a the updated prototypes

%%The training set matrix "T" should be in the following format
%%if there are "| T |" training samples and the problem is n-dimensional then %%The matrix T
would have n+1 rows and | T | columns
%%Considering a column, The first n rows are dedicated to a vector sample %%The last row is the
label of the sample. Label is a number from 1 to c
```

```
%%The prototype matrix "prototype" should be in the same format as %%training set matrix T

function [prototype,error_rate,error_rate0]=TRSLVQ_S(T,prototype,test,c,max,u,dim)

%%find the dimensions of training and prototype matrix

[T1,T2]=size(T); [p1,p2]=size(prototype);

%%Define the Gaussian distribution standard deviation

SD=5;
SD2=30;
%%define the dimensions of tangent planes tan_dim=dim;
%%define tangent planes for each prototype A=random('unif',-1,1,p1-1,tan_dim,p2); %%normalize
the basis
for i=1:p2

A(:,:,i)=orth(A(:,:,i));

end

%%find the error rate, before training

[test1,test2]=size(test); error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
PM2=A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2));

d2=sqrt(sum((PM2*(prototype(1:end-1,j)-test(1:end-1,i))).^2));

d=sqrt(d^2+d2^2); if d<d0

d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate0=error/test2;

%%the recursive adapting part

uu=u;
for t=1:max

u=uu/t;

%%randomly choose a training data from T
```

```
x=T(:,random('unid',T2)); x_main=x(1:end-1); x_label=x(end);
%%update prototypes accordingly partial_sum=0;

partial_sum_w=0; for j=1:p2

%%find projection matrix PM

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; PM2=A(:,:,j)*A(:,:,j)';
if prototype(end,j)==x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); d2=sqrt(sum((PM2*(prototype(1:end-1,j)-
x_main)).^2)); partial_sum=partial_sum+normpdf(d,0,SD)*normpdf(d2,0,SD2);

end
if prototype(end,j)~=x_label

d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2)); d2=sqrt(sum((PM2*(prototype(1:end-1,j)-
x_main)).^2)); partial_sum_w=partial_sum_w+normpdf(d,0,SD)*normpdf(d2,0,SD2);

end end

for j=1:p2
%%find projection matrix PM PM=eye(p1-1)-A(:,:,j)*A(:,:,j)';
PM2=A(:,:,j)*A(:,:,j)'; d=sqrt(sum((PM*(prototype(1:end-1,j)-x_main)).^2));
d2=sqrt(sum((PM2*(prototype(1:end-1,j)-x_main)).^2)); p=normpdf(d,0,SD)*normpdf(d2,0,SD2);
if prototype(end,j) == x_label

A(:,:,j)=A(:,:,j)+2.*u.*p*(1/partial_sum-1/(partial_sum+partial_sum_w)).*((-1/(SD)^2).*(x_main-
prototype(1:end- 1,j))*(x_main-prototype(1:end-1,j))'*PM+(1/(SD2)^2).*(x_main-
prototype(1:end-1,j))*(x_main-prototype(1:end- 1,j))'*PM2)*A(:,:,j);

prototype(1:end-1,j)=prototype(1:end-1,j)+u*p*(1/partial_sum- 1/
(partial_sum+partial_sum_w)).*((1/(SD)^2)*PM*PM*(x_main-prototype(1:end-1,j))+(1/
(SD2)^2)*PM2*PM2*(x_main- prototype(1:end-1,j)));

end
if prototype(end,j) ~= x_label

A(:,:,j)=A(:,:,j)-2.*u.*(p/(partial_sum_w+partial_sum)).*((-1/(SD)^2).*(x_main-
prototype(1:end-1,j))*(x_main- prototype(1:end-1,j))'*PM+(1/(SD2)^2).*(x_main-
prototype(1:end-1,j))*(x_main-prototype(1:end-1,j))'*PM2)*A(:,:,j);

prototype(1:end-1,j)=prototype(1:end-1,j)-u*(p/(partial_sum_w+partial_sum))*((1/
(SD)^2)*PM*PM*(x_main- prototype(1:end-1,j))+(1/(SD2)^2)*PM2*PM2*(x_main-
prototype(1:end-1,j)));

end

A(:,:,j)=orth(A(:,:,j));

end end

%%find the error rate, after training
```

```
error=0;
for i=1:test2

d0=100000; for j=1:p2

PM=eye(p1-1)-A(:,:,j)*A(:,:,j)'; PM2=A(:,:,j)*A(:,:,j)';

d=sqrt(sum((PM*(prototype(1:end-1,j)-test(1:end-1,i))).^2));
d2=sqrt(sum((PM2*(prototype(1:end-1,j)-test(1:end-1,i))).^2)); d=sqrt(d^2+d2^2);

if d<d0 d0=d;

class=prototype(end,j);

end end

if class~=test(end,i) error=error+1;

end end

error_rate=error/test2;
```

9- List of abbreviations

In this section the abbreviations, that are used in this article, are included, along with the full name they are abbreviating. For some of the abbreviations, references are mentioned, in parenthesis, to explain the underlying concept behind them.

LVQ : Learning Vector Quantisation

SLVQ: Soft Learning Vector Quantisation ([1])

ASLVQ: Attraction Soft Learning Vector Quantisation (explained in this article)

RSLVQ: Robust Soft Learning Vector Quantisation ([1])

TSLVQ: Tangent Soft Learning Vector Quantisation (explained in this article)

TRSLVQ: Tangent Robust Soft Learning Vector Quantisation (explained in this article)

TDSLVQ: Tangent Data Soft Learning Vector Quantisation (explained in this article)

TDRSLVQ: Tangent Data Robust Soft Learning Vector Quantisation (explained in this article)

PCA: Principle Component Analysis

GSLVQ: Grassmannian Soft Learning Vector Quantisation

10- Appendices

Appendix A

The following manipulations are done, in order to get (38b) and (38c). Only for (38b), the path is shown.

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logp(\mathbb{X}, y \,|\, W, A_r(W)) = \frac{p_i^j}{p(\mathbb{X}, y \,|\, W, A_r(W))}$$

$$\times \frac{-1}{\sigma_2^2\sqrt{2\pi\sigma_2^2}} exp\left(- \frac{||\left(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)(\mathbb{X} - \mathbb{W}_i^j)||^2}{2\sigma_2^2} \right)$$

$$\times (\mathbb{X} - \mathbb{W}_i^j)^T \left(I - A_r(\mathbb{W}_i^j)\left(A_r(\mathbb{W}_i^j)\right)^T \right)$$

$$\times \frac{-\partial\left(A_r(\mathbb{W}_i^j)\left(A_r(\mathbb{W}_i^j)\right)^T \right)(\mathbb{X} - \mathbb{W}_i^j)}{\partial A_r(\mathbb{W}_i^j)}$$

The above partial derivative, can be rewritten, using less symbols, as follows. Note that after calculating the result, the notation shall be returned to the original, for it is supposed to convey some agreed upon meanings.

$$\frac{\partial\{(AA^T)\left(\mathbb{X} - \mathbb{W}_i^j\right)\}_k}{\partial\{A\}_{m,n}} =$$

$$\begin{cases} 2\{(\mathbb{X} - \mathbb{W}_i^j)\}_k\{A\}_{kn} + \sum_{t\neq k}\{(\mathbb{X} - \mathbb{W}_i^j)\}_t\{A\}_{tn} & form = k \\ \{(\mathbb{X} - \mathbb{W}_i^j)\}_m\{A\}_{kn} & form \neq k \end{cases}$$

We would like to put the last result in neat form as below.

$$\frac{\partial(AA^T)(\mathbb{X} - \mathbb{W}_i^j)}{\partial\{A\}_{m,n}} = F_m\{(\mathbb{X} - \mathbb{W}_i^j)\}_m A_{:,n} + G_m A_{:,n}(\mathbb{X} - \mathbb{W}_i^j)^T e_m = 2\{(\mathbb{X} - \mathbb{W}_i^j)\}_m A_{:,n}$$

Where G_m is the identity matrix, with its m-th diagonal element to be 2, instead of 1. Similarly, F_m is the identity matrix, with its m-th diagonal element being 0, instead of 1. e_m is the m-th member of the standard basis of $\mathbb{R}^n$, with its m-th element being 1.

Then the following is done.

$$\frac{\partial}{\partial\{A_r(\mathbb{W}_i^j)\}_{m,n}} logp(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\frac{2p_i^j p\left(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j)\right)\{(\mathbb{X} - \mathbb{W}_i^j)\}_m}{\sigma_2^2 p(\mathbb{X}, y \mid W, A_r(W))}$$

$$\times\left((\mathbb{X} - \mathbb{W}_i^j)^T\left(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)\{A_r(\mathbb{W}_i^j)\}_{:,n}\right)$$

Finally, the partial derivative of the first term of the logarithmic likelihood function (36), with respect to $A_r(\mathbb{W}_i^j)$ is

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logp(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\delta(j-y)\left(\frac{2p_i^j p\left(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j)\right)}{\sigma_2^2 p\left(\mathbb{X}, y \mid W, A_r(W)\right)} \times (\mathbb{X} - \mathbb{W}_i^j)(\mathbb{X} - \mathbb{W}_i^j)^T\left(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)\right)A_r(\mathbb{W}_i^j)$$

Appendix B

Equation (41a) is the first term of the right hand side of (38a). An intermediate stage is included here.

$$\frac{\partial}{\partial A_r(\mathbb{W}_i^j)} logp(\mathbb{X}, y \mid W, A_r(W)) =$$

$$\frac{p_i^j}{p(\mathbb{X}, y \mid W, A_r(W))} \times \frac{-1}{2\pi\sigma_1\sigma_2} p(\mathbb{X} \mid \mathbb{W}_i^j, A_r(\mathbb{W}_i^j))$$

$$\times\left(\frac{(\mathbb{X} - \mathbb{W}_i^j)^T\left(I - A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_2^2} \times \frac{-\partial\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)(\mathbb{X} - \mathbb{W}_i^j)}{\partial A_r(\mathbb{W}_i^j)}\right.$$

$$\left. + \frac{(\mathbb{X} - \mathbb{W}_i^j)^T\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)}{\sigma_1^2} \times \frac{\partial\left(A_r(\mathbb{W}_i^j)(A_r(\mathbb{W}_i^j))^T\right)(\mathbb{X} - \mathbb{W}_i^j)}{\partial A_r(\mathbb{W}_i^j)}\right)$$

11- References

1. S. Seo and K. Obermayer – Soft Learning Vector Quantisation – Neural Computation - MIT Press Jul 2003 - 15(7):1589-604

2. P. Simard, Y.L. Cun and J. Denker – Efficient Pattern Recognition Using a New Transformation Distance – NIPS’92 proceedings the 5th international conference on neural information processing systems - pages 50-58 - 1992

3. S. Saralajew and Thomas Villmann – Adaptive Tangent Distances in Generalised Learning Vector Quantisation for Transformation and Distortion Invariant Classification Learning – Neural Network (IJCNN), IEEE Explore - 2016
4. H. Ritter, T. Martinetz and K. Schulten – Neural Computation and Self-Organizing Maps - Textbook - Addison-Wesley Longman publishing Co.1992

5. S. Chepushtanova and M. Kirby – Classification of Hyperspectral Imagery on Embedded Grassmannians, Hyperspectral image and signal processing: Evolution in remote sensing (WHISPERS) - 2014

6. Y. M. Lui, J. R. Beveridge and M. Kirby – Action Classification on Product Manifolds - Computer vision and pattern recognition, IEEE Conference - 2010

7. A. Bjorck and G.H. Golub – Numerical Method for Computing Angles between Linear Subspaces – Mathematics of Computation 27, 579-594 - 1973

8. S. Chepushtanova and M. Kirby – Sparse Grassmannian Embedding for Hyperspectral Data Representation and Classification - IEEE Geoscience and remote sensing letters - Vol 14, Issue 3 - 2017

9. T. Martinetz, S. Berkovich and K. Schulten – “Neural Gas” Network for Vector Quantization and its application to Time-Series Prediction – IEEE Transaction,Neural Networks, 4(4):558-69 - 1993

10. J.R. Munkres – Analysis on Manifolds - Textbook - 1991

11. D.M. Keysers – Modelling of Image Variability for Recognition- PhD Thesis - March 2006

12. M. Mohannazadeh and T. Villmann - Soft-LVQ and dependent prototypes - Machine Learning Reports - MiWoCI Workshop - 2017.

13. Teuvo Kohonen - Learning Vector Quantisation - Self-Organizing Maps. Springer Series in Information Sciences, vol 30. Springer, Berlin, Heidelberg - 1995