ELSEVIER

Artificial Intelligence 141 (2002) 123–135

Artificial Intelligence

www.elsevier.com/locate/artint

Unifying metric approach to the triple parity

Tony Y.T. Chan

The University of Aizu, Aizu-Wakamatsu City, Fukushima Prefecture, 965-8580, Japan

Received 26 June 2001; received in revised form 22 February 2002

Abstract

The even-odd parity problem is a tough one for neural networks to handle because they assume a finite dimensional vector space. Typically, the size of the neural network increases as the size of the problem increases. The triple parity problem is even tougher. In this paper, a method is proposed for supervised and unsupervised learning to classify bit strings of arbitrary length in terms of their triple parity. The learner is modeled by two formal concepts, transformation system and stability optimization. Even though a small set of short examples were used in the training stage, all bit strings of any length were classified correctly in the online recognition stage. The proposed learner has successfully learned to devise a way by means of metric calculations to classify bit strings of any length according to their triple parity. The system was able to acquire the concept of counting, dividing, and then taking the remainder, by autonomously evolving a set of string-editing rules along with their appropriate weights to solve the difficult problem.

Keywords: Parity problems; Artificial neural networks; Supervised learning; Unsupervised learning; Unifying metric approach; Transformation systems

1. Introduction

Among the most practical and popular vector space approaches to pattern recognition and machine learning (or pattern learning) are statistical discriminant analysis and neural network. These approaches in their current forms assume a finite dimensional, normed vector space. As such, they are unable to deal with the parity problem properly. Minsky and Papert [10, p. 254–255] pointed out some of the difficulties that a multilayer neural network would run into, even for solving a supervised parity problem of length 4. In Chapter 8 of [15], the writing that popularized the back-propagation algorithm, Rumelhart,

E-mail address: t-chan@u-aizu.ac.jp (T.Y.T. Chan).

0004-3702/02/$ – see front matter
PII: S0004-3702(02)00248-5

Hinton, and Williams mentioned that this is a "very difficult problem" [15, p. 334]. In [1], a general metric approach was proposed for inductive learning in two classes. It discovered class structures by maximizing the distances between objects in different classes while simultaneously minimizing the distances between objects in the same classes. It solved the particularly difficult unsupervised parity problem of unbounded length with 100% accuracy.

The notion of distance is extremely versatile. Grenander [7, Part IV] defined metric patterns by introducing a probability density on the objects. Besides versatility, another beauty of the metric approach is that distance can be defined on symbolic (structural) representations as well as on numeric representations because the normed vector space is a special case of the metric space. I used the metric approach for the supervised and unsupervised classification of chromosomes represented by strings [2]. Unlike the case for the usual parity problem, the median and telocentric chromosomes can be distinguished rather easily. This is not surprising since the structures of even and odd parities are actually more complicated than the structures of median and telocentric chromosomes in their string representations. Given the structures of the classes, this approach will attempt to find a metric function to sort out the classes in the same domain by exploiting their different structures. I also tackled the supervised feature subset selection problems in the Euclidean metric settings [3]. Wong, Shen, and Wong [19] used it for texture classification and did experiments on hundreds of training vectors at a time. Liang and Clarson used it on brainwaves [9], represented by vectors. The general unifying metric idea is akin to Fisher's discriminant [5, p. 115], which maximizes the distance between class centers while at the same time minimizing the within-class scatters. Fisher assumed a fixed Euclidean vector space throughout the process. Here we use dynamic families of metric spaces during the learning process. The preparation of an introductory book on this subject of metric approaches to pattern learning is underway [4].

This paper extends the method in [1] to three classes and applies the extension to the triple parity problem. This is even more difficult to solve than the usual even-odd parity because the decision surfaces required for triple parity are even more complex. For the supervised case, we are given three sets of correct examples. The first set contains some examples of remainder-0 parity strings, the second set contains some examples of remainder-1 parity strings, and the third set contains some examples of remainder-2 parity strings. Of course, for learning to be genuine, we do not tell the computer the meanings of the classes which these examples represent. It only knows that it has some examples of bit strings from some classes. The problem is for it to learn the generalizations to classify any bit string according to these examples.

Fundamentally, the goal of pattern learning is for the machine to adapt a working program for classification and prediction based on some chosen example data that it has been exposed to. The problem setting involves a pattern language describing a set P of objects. The object set is divided into exactly three mutually-exclusive subsets, $Q_1, Q_2, Q_3 \subset P$. Each of these subsets is assumed to be a class of objects. The teacher supplies three finite *training groups* of objects to the learning agent. The union of the training groups $\breve{Q}_1 \cup \breve{Q}_2 \cup \breve{Q}_3$ is called the *training set*. The agent then autonomously devises a way to distinguish the three sets Q_1, Q_2, and Q_3 by means of metric calculations. Hence, the next time an unknown object $p \in P$ is presented to the agent, it will be able

to classify it as belonging to Q_1, Q_2, or Q_3. The learner accomplishes this by deriving a stable metric space to separate the training groups. A stable metric space is one containing well-separated, compact clusters.

2. The learning model

The approach is modeled by two formal concepts, transformation system and stability optimization. Let $T = (P, S)$ be a transformation system, where P is an underlying set of structural objects described by a pattern language.

Let $s = x \leftrightarrow y$ be a substitution operation where x and y are subobjects. An object X can be transformed to the object Y via rule s by matching a subobject x from X and replacing it with y. Substitution rules are bidirectional in that the substitution of x by y implies that the substitution of y by x is also possible. When $x = \theta$ is empty, the operation is called insertion. When y is empty, the operation is called deletion. $S = (s_1, s_2, \ldots, s_m)$ is a list of m bidirectional substitution operations. These operations are the building blocks by which any object from P can be transformed into any other object from P.

Now we introduce weights to the substitution rules. With each substitution s_i, we associate a weight w_i, $w_i \geqslant 0$, so that it costs w_i to operate s_i. Let $W = (w_1, w_2, \ldots, w_m)$; and $\Delta_W(p_1, p_2) =$ the smallest total cost to transform p_1 into p_2 using costs W. The function Δ takes three arguments of two types: one type appears as a subscript and the other type appears inside the parentheses. The subscript W is a global (sometimes implicit) input argument compared with p_1 and p_2, which are local in scope. When it is clear from the context which W is meant, we may drop the subscript and write simply $\Delta(p_1, p_2)$. We interpret the *cheapest* cost as the distance between two objects. The more they relate, the closer their distance together, while the more they differ, the greater their distance apart. The definition of the cheapest cost is obviously dependent on the specific distance algorithm implementation. Under this model, a fast albeit non-optimal distance calculation is sometimes necessary as long as it can learn interesting concepts. The weights would attempt to distinguish the more important building blocks from the lesser ones in terms of relatedness. When a rule has weight 0, it means that this particular building block has no discriminant power in separating the classes of objects. When all the weights are 1, Δ simply counts the minimum number of operations required to transform one object into another.

Definition 1. The *average intra-group distance* for a training group k is

$$\rho_k(\Delta_W) = \frac{2}{n(n-1)} \sum_{i=2}^{n} \sum_{j=1}^{i-1} \Delta_W(\check{q}_i, \check{q}_j)$$

where $\check{q}_i, \check{q}_j \in \check{Q}_k$ and the size of $\check{Q}_k$ is n.

Given a specific training group and a specific distance function, ρ returns the average distance within a group of training objects. Imagine an $n \times n$ distance matrix. It is symmetric with 0's on the main diagonal. So, ρ returns the average over the distances

on the lower triangular part of this matrix. Note that for this formula to work, there must be at least two training objects. When n is equal to 1, we trivially define $\rho = 0$. We may drop the subscript or the input argument for ρ when it is clear from the context which subscript or argument is meant.

Definition 2. The *average inter-group distance* between groups k and h is

$$\upsilon_{k,h}(\Delta_W) = \frac{1}{nn'} \sum_{i=1}^{n} \sum_{j=1}^{n'} \Delta_W(\check{q}_i, \check{r}_j)$$

where $\check{r}_j \in \check{Q}_h$ and the size of $\check{Q}_h$ is n'.

Here n' can equal 1. In essence, these two rather standard definitions capture the idea of the average of a distance table where distances are listed between pairs of objects. Distance matrix (or distance table), in a sense, is the opposite of the idea of a confusion matrix where the diagonal elements are large and off-diagonal elements are small.

Definition 3. The *stability quotient* for three groups, 1, 2, and 3, is

$$Z_{1,2,3}(\Delta_W) = \frac{\rho_1 + \rho_2 + \rho_3}{\upsilon_{1,2}\upsilon_{1,3}\upsilon_{2,3}}.$$

The stability quotient serves as the criterion function of an optimization procedure so that we can simultaneously minimize the within group distances while maximizing the between group distances. Obviously, we would like to configure the topology in such a way that, within a group, objects are close to each other, while at the same time they are far from objects of other groups. The goal is to try to keep all intra-group distances equal to zero and none of the inter-group distances equal to zero. This is the central idea that gives the unifying metric approach a sense of intelligence and being able to learn.

It is not intelligent just to minimize the numerator instead of the entire quotient. Let s_j be a useless operation. There are plenty of these around, particularly when we consider the evolving nature of this system later on in this section. Let $w_j = 1$; then the rest of the useful and useless operations will all have a weight of 0. With this weighting scheme, the distance between any two objects in the pattern language is always 0, so that intra-group distances are automatically 0. Obviously, this weighting scheme has no discriminant power at all. Yet, minimizing only the numerator will quickly produce this meaningless weighting scheme. The denominator ensures that this scenario will not occur.

Definition 4. The *stability optimization* is to minimize

$$Z(\Delta_W),$$

subject to the constraint that

$$\sum_{i=1}^{m} w_i = 1.$$

In other words, we want Z to get as close to zero as possible since negative costs are not allowed.

Why bother calculating Z at all? Why not just calculate the misclassification rate on the training set, and directly minimize it subject to the same constraint? In fact, the latter is not a bad idea, particularly for the supervised tasks. The two minimizations work differently. The advantage of minimizing the misclassification rate is to save CPU time. But how can the learning agent calculate the misclassification rates when it does not even know the correct class labels for the training objects? The advantage of minimizing Z is found in the ability of the agent to detect erroneous labels for the unsupervised tasks by observing an asymptotic phenomenon during the optimization process, as we shall see later in Section 4.

What happens when the optimal Z-value is so high that it is unacceptable, i.e., as evidenced by a high misclassification rate? This occurs when the given set of building blocks cannot discriminate the classes. The classes intermingle with one another in the current metric space. Now we are ready to consider the evolving nature of this system. We begin with the given P, an underlying set of structural objects; S_0, the initial given set of substitution operations; and $\mathcal{G}$, a generator or constructor. The generator systematically constructs new substitution operations, that is, new building blocks from previous substitution operations. These new substitutions are called macros. (Compare with Arthur Samuel's use of products of preexisting predicates to invent new ones in his checker learning machine [10, p. x].) At each macro generation step $t > 0$, we have $S_t = S_{t-1} \cup \mathcal{G}(S_{t-1})$ so that $S_0 \subset S_1 \subset S_2, \ldots$. At each step $t \geqslant 0$, we associate with the current transformation system (P, S_t) its own stability optimization with its own Z_t.

There is a simple way to see whether or not the current Z_t is satisfactory after arriving at a certain step t. We suspend the optimization loop at that step and try out the current optimum solution on the training set (or on a separate validation test set). If the resulting classifications are satisfactory, we go to the on-line recognition stage. If not, we continue until another stable step is suspected. The definition of *satisfactory* could depend on the best, average, and variance of the misclassification rates thus far in the training process, and how much training time has already been spent on the current problem. We keep repeating this cycle of searching and validating on the training set. Convergence can be verified experimentally. In this way, we are attempting to minimize Z while at the same time getting some feedback from minimizing the misclassification rate on the training set.

Throughout the adaptation process, P and $\mathcal{G}$ are fixed. From an implementational time-complexity point of view, we have, in effect, added another loop (the t-loop in Fig. 1) on top of the optimization loop. The optimization loop tries out different cost vectors while the macro generation t-loop tries out larger and larger sets of macro substitution operations. Each stability optimization corresponds to a family of metric spaces. The number of times through the t-loop is the number of families being examined by the agent. In the unsupervised case, when training examples receive wrong labels or no labels, the training set will be repartitioned either randomly or according to the best metric obtained so far. This constitutes the outermost loop. The stopping criterion has always been the same: loop until it finds a stable metric space. Alternatively, the human designer can preset a time limit, after which the training process will be halted and new training objects can be introduced into the training set or some old training objects can be removed. In the perfect case, stable

```
Repartition the training groups accordingly
  Loop on set of substitution operations (t-loop)
    Optimization on weight vector until space stable
      Distance calculation of two objects, Δ_W(p_1, p_2)
      ⋮
End loops
```

Fig. 1. Control structure for the unsupervised inductive learning model.

means $Z = 0$. Otherwise, stable means that the groups in the metric space form well-defined clusters to the satisfaction of the learning agent, judging by the misclassification rate on the training set. In these cases, the training process stops and the agent is ready to go on-line to predict the class labels of any objects in the specified environment.

After the training phase is over, one can prepare for the recognition stage. We store the shortest object from each training group as the representative of the group. Also, we store the most stable substitution cost vector W^* at the end of the training process. When an unknown object is presented to the system, its distances to the representatives will be calculated using the weights W^*. It can then be classified according to the nearest neighbor rule. In the case when the best Z is nonzero, more than one representative can be chosen from each group and the k-nearest neighbor rule can be used for classification.

3. Supervised case

Now we apply the model to the triple parity problem of bit strings. The idea of triple parity is a generalization of the usual parity problem. The usual parity problem is the classification of strings as either even or odd parity. In the triple parity case, we first count the number of **1**'s in a bit string. Then we divide the count by 3 and obtain the remainder. If the remainder is 0, then the string belongs to class 1; if 1, then class 2; if 2, then class 3. The underlying set of structural objects P is the set of all finite strings of **0**'s and **1**'s. The initial set of substitution operations is

$$S_0 = \{\mathbf{0} \leftrightarrow \theta, \mathbf{1} \leftrightarrow \theta\},$$

where θ is the null string; i.e., we have two rules: insertion (or deletion) of a **0** and insertion (or deletion) of a **1**. The generator is

$$\mathcal{G}(S) = \left\{ab \leftrightarrow \theta \mid a \in \{\mathbf{0}, \mathbf{1}\}, b \leftrightarrow \theta \in S\right\}.$$

Consider the training set in Fig. 2. There are three training groups for three classes of objects. Each class is represented by only two or three relatively short examples. The size of P is infinite and the longest example from P can be arbitrarily large. We shall see how the learner proceeds step by step to discover the idea of triple parity.

At step $t = 0$, the learner can find no satisfactory weight vector for classification. Given the three training groups and given the two insertion operations, no matter what W it tries, Z is a relatively large value. There are too many misclassifications in the training set itself. If we had specified a test set, there would be many misclassifications in the test set as well.

	Group 1
1	**00**
2	**111**
3	**10101**

	Group 2
1	**01010101**
2	**1011001**

	Group 3
1	**101**
2	**11011001**

Fig. 2. Training groups with no mislabels.

The learner begins to evolve to the next step $t = 1$ by generating more operations. Applying $\mathcal{G}$ to S_0, it obtains 4 more substitution operations as follows:

$$S_1 = \left\{ \begin{array}{c} \mathbf{0} \leftrightarrow \theta \\ \mathbf{1} \leftrightarrow \theta \\ \mathbf{00} \leftrightarrow \theta \\ \mathbf{01} \leftrightarrow \theta \\ \mathbf{10} \leftrightarrow \theta \\ \mathbf{11} \leftrightarrow \theta \end{array} \right\}.$$

Applying stability optimization under the transformation system (P, S_1), it again finds no satisfactory separation of classes. Even with six rules, no matter how the weights are spread out, no clear clusters are formed. The training objects still intermingle with one another in this family of metric spaces.

However, when $t = 2$,

$$S_2 = \left\{ \begin{array}{c} \mathbf{0} \leftrightarrow \theta \\ \mathbf{1} \leftrightarrow \theta \\ \mathbf{00} \leftrightarrow \theta \\ \mathbf{01} \leftrightarrow \theta \\ \mathbf{10} \leftrightarrow \theta \\ \mathbf{11} \leftrightarrow \theta \\ \mathbf{000} \leftrightarrow \theta \\ \mathbf{001} \leftrightarrow \theta \\ \mathbf{010} \leftrightarrow \theta \\ \mathbf{011} \leftrightarrow \theta \\ \mathbf{100} \leftrightarrow \theta \\ \mathbf{101} \leftrightarrow \theta \\ \mathbf{110} \leftrightarrow \theta \\ \mathbf{111} \leftrightarrow \theta \end{array} \right\}.$$

Here, it finds the perfect stability $Z = 0$. Specifically, all average intra-group distances are zero, i.e., $\rho_1 = \rho_2 = \rho_3 = 0$; and all average inter-group distances are $\frac{1}{10}$, i.e., $\upsilon_{1,2} = \upsilon_{1,3} = \upsilon_{2,3} = \frac{1}{10}$. Fig. 3 shows the perfect weight vector as $W^* = (0, \frac{1}{10}, 0, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, 0, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, 0)$. It then chooses the shortest object from each group to represent the group, i.e., **00** from group 1, **1011001** from group 2 and **101** from group 3. The stable metric space contains three point-clusters because there is perfect stability. Each cluster is represented by its chosen prototype. The distance between any two clusters or prototypes is exactly $\frac{1}{10}$.

When an unknown object from P such as **1010010111** is presented to the agent to be classified, it calculates the distances of the unknown from the three prototypes using the

Index	Substitution	Weight
1	$\mathbf{0} \leftrightarrow \theta$	0
2	$\mathbf{1} \leftrightarrow \theta$	$\frac{1}{10}$
3	$\mathbf{00} \leftrightarrow \theta$	0
4	$\mathbf{01} \leftrightarrow \theta$	$\frac{1}{10}$
5	$\mathbf{10} \leftrightarrow \theta$	$\frac{1}{10}$
6	$\mathbf{11} \leftrightarrow \theta$	$\frac{1}{10}$
7	$\mathbf{000} \leftrightarrow \theta$	0
8	$\mathbf{001} \leftrightarrow \theta$	$\frac{1}{10}$
9	$\mathbf{010} \leftrightarrow \theta$	$\frac{1}{10}$
10	$\mathbf{011} \leftrightarrow \theta$	$\frac{1}{10}$
11	$\mathbf{100} \leftrightarrow \theta$	$\frac{1}{10}$
12	$\mathbf{101} \leftrightarrow \theta$	$\frac{1}{10}$
13	$\mathbf{110} \leftrightarrow \theta$	$\frac{1}{10}$
14	$\mathbf{111} \leftrightarrow \theta$	0

Fig. 3. Perfect weights.

perfect weight vector. When the distance between prototype i and the unknown is zero, then the unknown belongs to class i. In this way, all strings from the underlying infinite environment P can be classified correctly in the on-line recognition stage even though only a small-sized training set is used and only one object is chosen from each class for comparison.

This triple parity problem is very difficult to handle for methods that use vector representations. The current discriminant analysis, and neural network assume vector representations of objects. Fundamentally, the training procedures of these approaches involve a statistical discriminant analysis in the input vector space [5, pp. 114–121], [14, 18]. Basically, they favor spaces with larger between-class variances (or distances between class means) and smaller within-class variances. Even when weighted Euclidean distance (e.g., [11, p. 235], [13, p. 98], [16]) or more generally, Mahalanobis distance, is used, the space is still fundamentally Euclidean because the matrix of the symmetric bilinear form (or the inner product) is always positive semidefinite. (One easy way to get round this limitation is to use an indefinite pseudo-Euclidean inner product.) They begin with a finite dimensional vector space. Then this input space is partitioned by a sequence of linear and non-linear surfaces to produce the decision regions where class labels are finally assigned. A neural network plus its sequence of training vectors can be thought of as defining an equivalent relation on the rigid input vector space. It partitions the space into equivalent classes where each equivalent class corresponds to a category of objects. Changing the initial conditions of the neural network can change the partition (i.e., the equivalent relation) on this rigid space. In their current popular forms of vector representations and gradient descent learning algorithms, they are most suitable for problems that are statistical in character and cannot naturally handle infinite dimensional vectors. (Note that

metric approaches could also be applied to statistical problems [3,9,19].) Typically, the architectural complexity and the training time complexity of the neural network increases as the dimensionality of the problem increases. For the parity problem, C4.5 requires exponential time to learn the decision tree [13, p. 100]. Ishikawa [8] provided an interesting modular neural network learning method to alleviate some of these difficulties by using problem decomposition. (See also [6].)

By contrast, parity objects in an abstract metric space do not have dimensions to speak of, so that abstract metric approaches can naturally handle objects of unspecified lengths. The parity vectors, however, must be limited to fixed dimensions, let's say, 99 dimensions. Consider a vector v belonging to class i in this space. Its set of Euclidean nearest neighbors is only a distance of 1 from it. There are 99 of them and none belong to class i. For any parity vector v, its closest neighborhood is completely overwhelmed by parity vectors from the other two classes, so that the otherwise very useful k-nearest neighbor algorithm will fail miserably in this context. Its second set of Euclidean nearest neighbors is only a distance of 1.41 from it. There are $\binom{99}{2} = 4851$ of them and only some of them belong to class i. These heavily intermingled class distributions of the triple parity problem are even more complex than the ones for the even-odd parity. The distributions make the decision surfaces in the 99-dimensional vector space for the three classes extremely complicated and extremely difficult to train by a sequence of linear and non-linear surfaces. The space itself resembles a unit hypercube in a 99-dimensional boolean space rather than a genuine 99-dimensional real vector space. Every parity vector is on a corner (vertex) of the cube with no objects between the corners. Finally, otherwise important vector space tools, such as mean vector, covariance matrix, eigenvalue, Euclidean distance, inner product, derivative, etc., are rather meaningless and useless in this boolean setting. Discriminant analysis depends heavily on covariance matrix, and neural network's gradient descent on derivative.

The unifying metric approach proposed here, however, does not assume any vector space or even any fixed topology. Instead, when it encounters a bad topology, it throws it out completely and gets another one by explicitly changing the distance function. By changing the distance function directly, it eventually changes the topology of the space. (Of course, not every topological space is metrizable. See [12]. But for our purpose, it is sufficient that we confine the discussion to metrizable topological spaces, that is, metric spaces.) A particular distance function can be thought of as defining an equivalent relation on the input pattern language which is a space far more flexible than the orderly, Cartesian vector space. It partitions the language space into (equivalent) classes. Changing the distance function can change the partition. For the parity problem, we need structural representation, not vector representation. Our method relies on the string representation of the objects to find a solution for separating the classes. Naturally, string operations, like the insertion and deletion of a symbol or substring, are used in the hope of obtaining good separation of classes in terms of string distances. Instead of a normed vector space, families of metric spaces are dynamically selected. Eventually, the learning agent successfully devised a way to classify bit strings according to triple parity, even though the agent was never told its concept or meaning. It was able on its own to acquire the idea of counting, dividing, and taking the remainder by evolving a set of substitution rules.

It is not exactly surprising that the insertions and deletions of **1**, **11**, **111**, etc. can capture the idea of triple parity. There is really no magic here. The representations of the input patterns must implicitly somehow already contain the information for class discrimination. Otherwise no method can derive a successful classifier no matter what you do. Minsky and Papert [10] capture this important idea on the nature of learning rather succinctly: "No machine can learn to recognize X unless it possesses, at least potentially, some scheme for representing X". Every pattern learning session is characterized by three things: the initial set of assumptions or facts (the initial knowledge), an efficient domain-specific learning procedure that can work on these facts, and finally, the learned new facts/concepts (i.e., the learned procedure). There is no universal learning machine, in the sense that one machine can learn everything efficiently. The push in the field of pattern learning is to develop different learning technologies and models of learning agents that can learn non-obvious concepts quickly from as few initial facts as possible, so that human intervention is minimal. Of course, one is willing to spend more time learning the more difficult concepts, especially in the absence of a teacher.

4. Unsupervised case

Now, let us consider the unsupervised triple parity problem, which is quite hopeless for any vector space approach to attempt. In essence, the unsupervised problem is the supervised problem with erroneous labels applied mistakenly. There could be many sources of errors. The three primary sources are the teacher, a noisy environment, and imprecision in the measuring instruments. It does not matter where the mistakes come from; it does not matter whether they are supervised or unsupervised tasks. In any case, we always presume, at least temporarily, that every object is correctly labeled. In this spirit, we corrupt group 1 with one erroneous string from class 2, group 2 with one erroneous string from class 3, and group 3 with three erroneous strings, one from class 1 and two from class 2, as shown in Fig. 4. Note that with group 3, there are actually more erroneous labels than proper labels. There are a total of 5 mislabels among 12 training objects.

We use the same initialization as previously. At step $t = 0$, the learner can find no satisfactory weight vector for classification. The system begins to evolve to the next step $t = 1$ by generating more operations just as before with similarly unsatisfactory stability values and misclassification rates. At $t = 2$, unlike the previous supervised case, there is

	Group 1
1	**00**
2	**111**
3	**10101**
4	**10**

	Group 2
1	**01010101**
2	**1011001**
3	**11**

	Group 3
1	**101**
2	**11011001**
3	**000**
4	**010111**
5	**01**

Fig. 4. Training groups with 5 mislabels.

	1	2	3	4	5
1	0	0	2ε	ε	ε
2	0	0	2ε	ε	ε
3	2ε	2ε	0	ε	ε
4	ε	ε	ε	0	0
5	ε	ε	ε	0	0

Fig. 5. Distance table for group 3.

still no perfect stability value. However, the system notices a peculiar occurrence. Here, the learning agent observes that near a boundary point of the weight space, at the cost vector

$$W' = (0, \varepsilon, 0, w_4, w_5, w_6, 0, w_8, w_9, w_{10}, w_{11}, w_{12}, w_{13}, 0)$$

with $w_4, w_5, w_6, w_8, w_9, w_{10}, w_{11}, w_{12}, w_{13} > \varepsilon > 0$ and

$$\varepsilon + w_4 + w_5 + w_6 + w_8 + w_9 + w_{10} + w_{11} + w_{12} + w_{13} = 1,$$

there is an asymptote phenomenon related to the numerator of the stability quotient.

First, it observes that for the denominator $\upsilon_{1,2}\upsilon_{1,3}\upsilon_{2,3} \neq 0$. This means that no two groups have an average distance of 0 between them. This is good because if any one of the factors is 0, then one class has its training objects split into exactly two groups while the third group contains the combined training objects of the other two classes. As long as the denominator is not equal to 0, there is hope for an asymptote in the numerator.

Second, for the average intra-group distances, $\rho_1 = \frac{3}{6}\varepsilon$, $\rho_2 = \frac{2}{3}\varepsilon$ and $\rho_3 = \varepsilon$, so that the numerator is equal to $\frac{13}{6}\varepsilon$. Fig. 5 shows the intra-group distances for group 3. Interestingly, the numerator can be made to approach zero as ε also approaches 0. We could have included even more erroneous labels in the groups and the asymptote phenomenon would still occur. We do not want $\varepsilon = 0$, because when it does, the inter-group distances will also be zero. Indeed, the formulation of the stability quotient is such that even if only one inter-group distance is 0, it is unacceptable. When there are noisy training data in the training groups, and there are multiple operations, any of which can suitably separate the classes, this asymptotic phenomenon is not unusual (similar occurrence in [1]).

Now, the learner can use W' to repartition the training set by performing a simple spanning tree clustering algorithm in the finite metric space $(\check{Q}_1 \cup \check{Q}_2 \cup \check{Q}_3, W')$. Objects of the same classes in this space have a distance of 0. Objects of different classes in this space have a distance of at least ε. Clearly, there are three point-clusters in this finite space. After the repartition, the system once again finds the same W^* as before to be the perfect weight vector and stores the shortest string from each group as representative to do classification for any members of P with 100% accuracy.

For the no-label unsupervised problem, the agent is given, let's say, the same initial set of substitution operations, generator, and seven training examples as in Section 3, but importantly—no class labels. However, it is told that there are three classes among these seven training objects. Without further information, the agent would randomly partition the seven parity strings into three groups and give them temporary labels. Then it will execute the strategy outlined in Fig. 1 until a stable space is found. Given these standard initial conditions (no problem-specific bias), the most natural class concept to be found is the

triple parity. When the learner was given the same initial conditions except that it was told that there were only two classes, then it found the even-odd parity concept as occurred in [1]. Changing any of the initial conditions, i.e., a different starting set of substitutions, generator, training examples, labeling, distance algorithm, or number of classes, would result in different bit string concepts to be learned (more of this in a future paper). For example, without the generator, it could solve the 2-bit exclusive-or problem [1] and the chromosome classification problem [2].

The basic model here with its (macro) transformation system and stability optimization naturally lends itself to unsupervised learning for more than three classes. One could, for example, concentrate on learning one class at a time, and temporarily put all the other clusters together as the second class. Learning then will proceed in a hierarchical (vertical) or tall manner. (See [17].) Other schemes, such as in a flat (horizontal) manner, are also possible, by dividing the training set into 2 subsets (groups), with each subset actually containing a number of classes. The goal is to try to keep all intra-group distances equal to zero and none of the inter-group distances equal to zero at all levels of the decision (learning) tree.

5. Conclusion

The idea of concept learning is seen here as the process of deriving a stable metric space to separate the training groups. A stable metric space is one containing well-separated, compact clusters. When the space is not stable, the learner would change the space by directly changing the way distance is to be calculated until it has found a stable space. Each training group contains examples of a concept. Ideally, each concept can be represented by a single prototype. The problem then becomes finding a transformation system that can transform objects to their respective prototypes at a relatively cheap or nil cost.

The proposed method has been applied to learning with an error-free teacher, learning with an erroneous teacher, and learning without a teacher. For the unsupervised learning tasks, it turns out that it is instructive to treat them as supervised tasks with mislabels. Interestingly, under the unifying metric model, such mistakes can be detected by the learner in the observing of an asymptotic behavior during the metric learning process.

I have extended the model in [1] for unsupervised inductive learning in three classes. All bit strings were classified correctly during on-line recognition. The agent has successfully learned to devise a way to classify bit strings according to the concept of triple parity. The system itself is able to acquire the concept of counting, dividing, and then taking the remainder by evolving a set of substitution operations by which to transform one object into another.

Acknowledgements

I'd like to thank the anonymous reviewer for his most insightful comments.

References

[1] T.Y.T. Chan, Inductive pattern learning, IEEE Trans. Systems Man Cybernet.—Part A: Systems and Humans 29 (6) (1999) 667–674.
[2] T.Y.T. Chan, Unsupervised classification of noisy chromosomes, Bioinformatics 17 (5) (2001) 438–444.
[3] T.Y.T. Chan, Fast naive Euclidean concept learning, in: Proceedings of the 2nd IEEE International Conference on Systems, Man, and Cybernetics, IEEE Computer Society, 2002.
[4] T.Y.T. Chan, Pattern Learning: A Unifying Approach, Kluwer Academic, Boston, MA, 2003.
[5] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.
[6] L. Franco, S.A. Cannas, Generalization properties of modular networks: Implementing the parity function, IEEE Trans. Neural Networks 12 (6) (2001) 1306–1313.
[7] U. Grenander, General Pattern Theory: A Mathematical Study of Regular Structures, Oxford University Press, Oxford, 1993.
[8] M. Ishikawa, Learning of modular structured networks, Artificial Intelligence 75 (1995) 51–62.
[9] J.J. Liang, V. Clarson, A new approach to classification of brainwaves, Pattern Recognition 22 (6) (1989) 767–774.
[10] M. Minsky, S. Papert, Perceptrons, Expanded edition, MIT Press, Cambridge, MA, 1988.
[11] T. Mitchell, Machine Learning, McGraw Hill, New York, 1997.
[12] J.R. Mundres, Topology, Prentice Hall, Englewood Cliffs, NJ, 1975.
[13] J.R. Quinlan, C4.5, Morgan Kaufmann, San Mateo, CA, 1993.
[14] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley, B.W. Suter, The multilayer perceptron as an approximation to a Bayes optimal discriminant function, IEEE Trans. Neural Networks 1 (4) (1990) 296–298.
[15] D.E. Rumelhart, J.L. McClelland, PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, MIT Press, Cambridge, MA, 1986.
[16] J. Schürmann, Pattern Classification: A Unified View of Statistical and Neural Approaches, Wiley, New York, 1996.
[17] L. Talavera, J. Béjar, Generality-based conceptual clustering with probabilistic concepts, IEEE Trans. Pattern Anal. Machine Intelligence 23 (2) (2001) 196–206.
[18] A.R. Webb, D. Lowe, The optimal internal representation of multilayer classifier networks performs nonlinear discriminant analysis, Neural Networks 3 (1990) 367–375.
[19] A.K.C. Wong, H.C. Shen, P. Wong, Search-effective multi-class texture classification, Internat. J. Pattern Recognition Artificial Intelligence 4 (4) (1990) 527–552.