Incremental Hierarchical Discriminant Regression

Juyang Weng
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824
(weng@cse.msu.edu)
and
Wey-Shiuan Hwang
Rudolph Technologies, Inc.
One Rudolph Road
Flanders, NJ 07836
(whwang@rudolphtech.com)
December 11, 2005

Abstract

This paper presents Incremental Hierarchical Discriminant Regression (IHDR) which incrementally builds a decision tree or regression tree for very high dimensional regression or decision spaces by an online, real-time learning system. Biologically motivated, it is an approximate computational model for automatic development of associative cortex, with both bottom-up sensory inputs and top-down motor projections. At each internal node of the IHDR tree, information in the output space is used to automatically derive the local subspace spanned by the most discriminating features. Embedded in the tree is a hierarchical probability distribution model used to prune very unlikely cases during the search. The number of parameters in the coarse-to-fine approximation is dynamic and data-driven, enabling the IHDR tree to automatically fit data with unknown distribution shapes (thus, it is difficult to select the number of parameters up front). The IHDR tree dynamically assigns long-term memory to avoid the loss-of-memory problem typical with a global-fitting learning algorithm for neural networks. A major challenge for an incrementally built tree is that the number of samples varies arbitrarily during the construction process. An incrementally updated probability model, called sample size dependent negative-log-likelihood (SDNLL) metric is used to deal with large-sample size cases, small-sample size cases, and unbalanced-sample size cases, measured among different internal nodes of the IHDR tree. We report experimental results for four types of data: synthetic data to visualize the behavior of the algorithms, large face image data, continuous video stream from robot navigation, and publicly available data sets that use human defined features.

Keywords: online learning, incremental learning, cortical development, discriminant analysis, local invariance, plasticity, decision trees, high dimensional data, classification, regression, and autonomous development.

I. Introduction

The cerebral cortex performs regression with numerical inputs and numerical outputs. At an appropriate temporal scale, the firing rate (or frequency of the spikes) has been modeled as a major source of information carried by the signals being transmitted through the axon of a neuron [1, pages 21-33]. A cortical region develops (adapts and self-organizes) gradually through its experience of signal processing. Pandya & Seltzer [2] proposed that there are four types of cortex: primary, association, multimodal, and paralimbic. The association cortex lies between the primary cortex and motor areas [3, pages 183-188]. The mathematical model described here can be considered as a coarse, approximate computational model for the development of the association cortex, whose main goal is to establish the association (i.e., mapping) between the primary sensory cortex and motor cortex. However, a lot of puzzles about the biological brain are unknown. The computational model described here does not intend to fit all biological details.

Classification trees (class labels as output) and regression trees (numerical vectors as output) have two purposes: indexing and prediction. Indexing trees, such as K-D trees and R-trees, have been widely used in database for known data retrieval with a goal to reach a logarithmic time complexity. Prediction trees, also called decision trees, have been widely used in machine learning to generate a set of tree-based prediction rules for better prediction for unknown future data. Although it is desirable to construct a shallow decision tree, the time complexity is typically not an issue for decision trees. The work presented here is required for a real-time, online, incrementally learning artificial neural network system [4]. Therefore, both goals are thought: fast logarithmic time complexity and superior generalization. Further, we require the tree to be built incrementally, since the tree must be used for operation while data arrive incrementally, such as in a system that takes real-time video streams. Thus, in the work presented here we concentrate on trees and, in particular, incrementally built trees.

Traditionally, classification and regression trees use a univariate split at each internal node, such as in CART, [5], C5.0, [6] and many others. This means that the partition of input space by each node uses hyper-planes that are orthogonal to the axes of the input space X. Multivariate linear splits correspond to partition hyper-planes that are not necessarily orthogonal to any axis of the input space and thus, potentially can cut along the decision boundaries more appropriately for better prediction or generalization. Trees that use multivariate splits are called oblique trees. As early as the mid-70s, Friedman [7] proposed a discriminating node-split for building a tree which resulted in an oblique tree. The OC1 by Murthy et al. [8] and SHOSLIF tree by Swets & Weng [9] are two methods for constructing oblique trees. For an extensive survey of decision trees, see a survey by [10].

The OC1 uses an iterative search for a plane to find a split. The SHOSLIF uses the principal component analysis (PCA) and linear discriminant analysis (LDA) to directly compute splits. SHOSLIF uses multivariate nonlinear splits corresponding to curved partition surfaces with any orientation. Why is discriminant analysis such as LDA important? LDA uses information of the output space in addition to the information in the input space to compute the splits. PCA only uses information in the input space. Consequently, variations in the input space that are totally useless for output (e.g., pure noise components) are also captured by the PCA. A discriminant analysis technique, such as LDA, can disregard input components that are irrelevant to output (e.g., pure noise components).

The problem gets harder when the output is a multidimensional analogue signal, as is the case with a real-time motor controlled by a robot. The class label is unavailable and thus, the LDA method is not directly applicable. Developed concurrently with the incremental version presented here, the Hierarchical Discriminant Regression (HDR), by the same authors [11], performs clustering in both output space and input space, while clusters in the output space provide virtual labels for membership information in forming clusters in the input space. An HDR tree uses multivariate nonlinear splits, with multivariate linear splits as a special case.

A. Incremental regression

The problem becomes even harder if the learning must be fully incremental. By fully incremental, we mean that the tree must be updated with every input vector. It is not unreasonable to assume that the brain is fully incremental: it must function and update for every sensory input. In an incremental learning system, the data frames arrive sequentially in time. Each frame (vector) $x(t)$ is a sample (snapshot) of the changing world at time t. Since the streams of observation are very long and often open-ended, the system must be updated using one frame at a time. Each frame is discarded as soon as it is used for updating. The major reasons for sequential learning include: (a) The total amount of data is too much to be stored. (b) Batch processing (including block-sequential

processing, where the system is updated with each temporal block of data) takes time and introduces time delay between the time when the first batch is collected and the time the next batch is collected. (c) Updating a tree is fast, significantly faster than constructing the entire tree. Thus, the latency between updating using a frame and using the updated tree is short.

Since incremental learning works under a more restricted condition (e.g., all the training data are not available all at once), the design of an effective incremental algorithm is typically more challenging than a batch version. In the neural network community, incremental learning is common since the network alone does not have space to store all the training data. Further, incremental learning is a must for simulating what is called autonomous mental development [12] [13].

Due to the fact that incremental learning operates under more restrictive conditions, typically one should not expect an incremental learning system to out-perform a batch learning method in terms of, e.g., error rate. But, our experimental results presented in Section V indicated that the difference of error rates between HDR and IHDR is small, and in a test (Table III) the error of IHDR is smaller.

However, we can also take advantage of the incremental learning nature: (a) Perform while being built. The tree can work before all the data are available. (b) Concentration on hard cases. The later training samples can be collected based on the performance of the current tree, allowing for the collection of more training cases for weak cases or hard-to-learn cases. (c) Dynamic determination of the number of training samples. Given a classification or regression task, it is very difficult to determine how many training samples are needed. Incremental learning enables dynamic determination of the total number of samples needed based on current system performance. (d) The same sample (micro-cluster in the leaf nodes of IHDR), received at different times, can be stored at different positions of the tree, potentially improving the performance.

The batch processing version of HDR appeared in [11]. This paper presents the Incremental HDR (IHDR). We concentrate on the incremental nature of the technique and refer the reader to [11] for issues common to HDR and IHDR. In other words, IHDR follows the learning principle of typical neural networks — incremental learning. The incrementally generated IHDR tree is a kind of network. Unlike traditional neural networks, such as feed forward networks and radial basis functions, the IHDR network has the following characteristics:

1) A systematic organization of long-term memory and short-term memory. The long-term memory corresponds to information in shallow nodes and micro-clusters (also called primitive prototypes, when the meaning of micro-clusters is alluded) in leaf nodes which are not visited often. The short-term memory corresponds to micro-clusters in leaf nodes that are visited often, so that the detail is forgotten through incremental averaging. The long-term memory prevents catastrophic loss of memory in, e.g., back-propagation learning for feed-forward networks.
2) A dynamically, automatically determined (not fixed) set of system parameters (degrees of freedom). The parameters correspond to the mean and covariance matrix of probability models in all of the internal nodes and the dynamically created micro-clusters in leaf nodes. The dynamically determined degrees of freedom present severe local minima that might result from a network with a fixed number of layers or a fixed number of nodes in each layer, when it intends to minimize the error of fitting for desired outputs. IHDR does not use fitting at all.
3) A course-to-fine distribution approximation hierarchy so that coarse approximation is finished at parent nodes before finer approximation by their children. The imperfection of the boundaries determined by early frozen ancestor nodes are corrected and refined by their later children. Such a scheme also contributes to the avoidance of the local minima problem: limited experience (the total number of micro-clusters in all leaf nodes) may result in lack of detail in regression but not local minima in overall fitting.
4) Fast local update through the tree without iteration. Some existing methods, such as evolutionary computation and simulated annealing can deal with local minima, but they require iterative computations which are not suited for the real-time updating and performance.

We did not find, in the literature, an efficient technique that performs discriminant analysis incrementally while satisfying all of the seven stringent requirements discussed below. As far as we know, there was no prior published incremental statistical method suited for constructing a regression tree for high dimensional input space based on discriminant analysis. By high dimensional space we mean that the dimension ranges from a few hundred to a few thousand and beyond. The number of samples can be smaller than the dimension. When the number of samples is

smaller than the input dimension (i.e., the number of features), [5] and [8] described the situation as data underfits the concept and thus, disregarded the situation.

This high dimensional, small sample case becomes very important with increased use of high dimensional digital multimedia data, such as images and video, where each pixel value is a component of the input vector. [1] These applications give rise to high dimensional data with strong correlations among components of input. CART, C5.0, OC1, and other published tree classifiers that we know perform reasonably well for relatively low-dimensional data that was prepared by humans. Each component of such training data is a human-defined feature and thus, correlation among these features are relatively low. However, they are not designed for highly-correlated, directly-sensed, high-dimensional data such as images and video. Further, the elements in such a high-dimensional vector are highly correlated, since pixels are highly correlated. The SHOSLIF tree is for high input dimension and has an incremental version, [18], but it uses PCA for the splits. It is technically challenging to incrementally construct a classification or regression tree that uses discriminant analysis due to the complex nature of the problem involved.

B. Regression requirements

With the demand of online, real-time, incremental, multi-modality learning with high-dimensional sensing by an autonomously learning embodied agent, we require a general purpose regression technique that satisfies all of the following seven (7) challenging requirements:

1) It must take high-dimensional inputs with very complex correlation between components in the input vector (e.g., an image vector has over 5000 dimensions). Some input components are not related to output at all (e.g., posters on a wall are not related to navigation).
2) It must perform one-instance learning. An event represented by only a single input sensory frame must be learned and recalled. Thus, iterative learning methods, such as back-propagation learning, are not applicable.
3) It must adapt to increasing complexity dynamically. It cannot have a fixed number of parameters like a traditional neural network, since the complexity of the desired regression function is unpredictable.
4) It must deal with the local minima problem. If the tree currently being built sticks into a local minima, the tree being built is a failed tree. In online real-time learning of open-ended autonomous development of an agent, such a failed case means that the agent failed to develop normally. Traditionally, a variety of engineering methods have been used to alleviate the problem of local minima, e.g., (a) simultaneously keeping multiple networks, each starting with a different random initial guess, and only the best performing network is selected, (b) simulated annealing, and (c) evolutionary computation. However, these methods are not applicable to real-time online development where every system must develop normally.
5) It must be incremental. The input must be discarded as soon as it is used for updating the memory. It is impossible to save all the training samples since the space required is too large.
6) It must be able to retain most of the information of the long-term memory without catastrophic memory loss. However, it must also forget and neglect unrelated details for memory efficiency and generalization. With an artificial network with back-propagation learning, the effect of old samples will be lost if these samples do not appear later.
7) It must have a very low time complexity in computing and updating so that the response time is within a fraction of second for real-time learning, even if the memory size has grown very large. Thus, any slow learning algorithm is not applicable here. Of course, the entire tree construction process can extend to a long time period.

Some existing artificial neural networks can satisfy some of the above requirements, but not all.

For example, consider feed forward neural networks with incremental back-propagation learning. They perform incremental learning and can adapt to the latest sample with a few iterations (not guarantee to fit well), but they do not have a systematically organized long-term memory, and thus, early samples will be forgotten in later training. Cascade-Correlation Learning Architecture [19] improves them by adding hidden units incrementally and fixing their weights to become permanent feature detectors in the network. Thus, it adds long-term memory. Major problems for them include the high-dimensional inputs and local minima.

[1] This corresponds to a well-accepted and highly successful approach called the appearance-based approach, with which the human system designer does not define features at all but rather applies statistical methods directly to high-dimensional, preprocessed image frames, as seen in the work of [14], [15], [16] and, [17].

We present IHDR to deal with the above 7 requirements altogether, which is a very challenging task of design. Further, we deal with the unbalanced sample problem in that some regions of input space have a large number of samples, while other regions have sparse samples. A sample-size dependent likelihood measure is proposed to make suboptimal decisions for different sample sizes, which is critical for an incremental algorithm; it must perform reasonably well while being constructed.

We present experimental results that demonstrate the performance of the new IHDR technique and compare it with some major published methods.

II. The IHDR Method

We first discuss briefly classification and regression.

A. Unification of classification and regression

The tasks of discriminant analysis can be categorized into two types according to their output: class-label (symbolic) output and numerical output. The former case is called classification and the latter case is called regression. A classification task can be defined as follows.

Definition 1: Given a training sample set $L = \{(x_i, l_i) \mid i = 1, 2, \ldots, n\}$, where $x_i \in \mathcal{X}$ is an input (feature) vector and l_i is the symbolic label of x_i, the classification task is to determine the class label of any unknown input $x \in \mathcal{X}$.

A regression task is similar to the corresponding classification one, except that the class label l_i is replaced by a vector y_i in the output space, $y_i \in \mathcal{Y}, i = 1, 2, \ldots, n$. The regression task can be defined as follows.

Definition 2: Given training set $L' = \{(x_i, y_i) \mid i = 1, 2, \ldots, n\}$ and any testing vector $x \in \mathcal{X}$, the regression task is to estimate the vector $y(x) \in \mathcal{Y}$ from $x \in \mathcal{X}$.

Regression tasks are very common. As long as the output of the system needs to control effectors that take graded values, the learning problem is regression. Examples include motors, steering wheels, brakes, and various machines in industrial settings. The biological cortex also performs regression.

In a classification problem, the class labels themselves do not provide more information in terms of how different two classes are. Any two different classes are just different, although some may differ more than others in the original application. We can cast a classification task into a regressive one so that we can conveniently form coarse classes by merging some original classes. These coarse classes are useful for performing coarse-to-fine classification and regression using a decision tree, as we will explain later in this paper.

Here, we outline three ways to cast a classification task into a regression one. More detail is available in [11].

1) Canonical mapping. Map n class labels into an n-dimensional output space. For the i-th class, the corresponding output vector y_i is an n-dimensional vector which has 1 as its i-th component and all the other components are zero. For incremental learning, this method has a limitation since the maximum number of classes is limited to n.
2) Embedding cost matrix. If a cost matrix $[c_{ij}]$ is available, the n class labels can be embedded into an $(n-1)$-dimensional output space by assigning vector y_i to class i, $i = 1, 2, \ldots, n$, so that $||y_i - y_j||$ is as close to c_{ij}, the cost of confusing classes i and j, as possible. This process is not always practical since a pre-defined cost matrix $[c_{ij}]$ is not always easy to provide. This also means that the number of classes is limited for the incremental learning case.
3) Class means in input space. Each sample (x_{ij}, l_i) belonging to class l_i is converted to (x_{ij}, y_i), where y_i, the vector class label, is the mean of all x_{ij} that belong to the same class l_i. In the incremental learning, the mean is updated by using an amnesic average as described in Section III-F. This is often a desirable method since the distance in the output space is closely related to the distance in the input space.

On the other hand, one cannot map a numeric output space into a set of class labels without losing the numeric properties among an infinite number of possible numerical vectors. Therefore, a regression problem is more general than the corresponding classification problem.

B. Technical motivation

In the remainder of this paper, we consider a general regression problem: incrementally approximating a mapping

$$h : \mathcal{X} \mapsto \mathcal{Y}$$

constructed from a set of training samples $\{(x_i, y_i) \mid x_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i = 1, 2, \ldots, n\}$. By incremental approximation, we mean that the incremental developer d takes the previous mapping $h^{(i-1)}$ and input the sample (x_i, y_i) to produce the updated mapping $h^{(i)}$:

$$h^{(i)} = d(h^{(i-1)}, x_i, y_i)$$

for $i = 1, 2, ...$

With the high-dimensional input space $\mathcal{X}$, many components are not related to the desired output at all. A straightforward nearest neighbor classifier, using the Euclidean distance in $\mathcal{X}$ space, will fail miserably. Fig. 1 illustrates a 2-D example. Fig. 1(a) shows the decision boundary of the straightforward nearest neighbor rule called the *Voronoi diagram*. Its error rate is high (about 50%) as shown in Fig. 1(c). If the discriminating feature, X_1 in this example, is detected and a nearest neighbor classifier is used in this discriminating subspace (1-D), the error rate is drastically smaller as shown in Fig. 1(d).

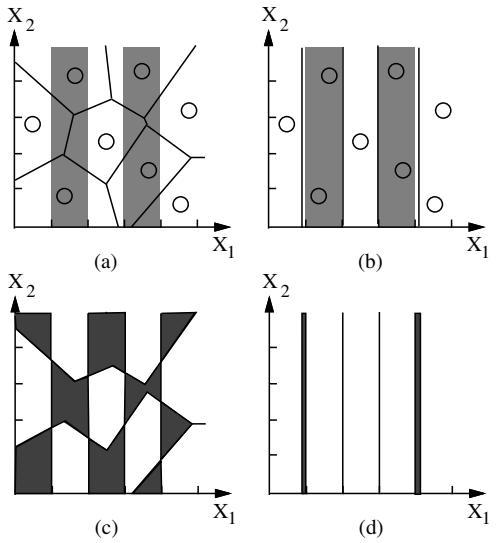


Fig. 1. The importance of finding the discriminating features. (a) The training samples are marked as small circles. The colors indicate the corresponding output. The Voronoi diagram is shown by line segments. (c) A large classification error resulted from the straightforward nearest neighbor classifier. Misclassified areas are marked by a dark shade. (b) Only the discriminating feature X_1 is used by the nearest neighbor rule. (d) Misclassified areas of nearest neighbor rule using X_1 only, which are smaller than (c).

The phenomenon illustrated in Fig. 1 becomes more severe in high-dimensional space because there are more dimensions (like X_2) that distract the Euclidian distance used by the nearest neighbor classifier. Therefore, it is necessary to automatically derive discriminating features by the regressor.

If y_i, in the incrementally arriving training samples (x_i, y_i), is a class label, we could use the linear discriminant analysis (LDA) as in [20]'s work since the within-class scatter and between-class scatter matrices are all defined. Unfortunately, if each class has a small number of training samples, the within-class scatter matrix is poorly estimated and often degenerate, and thus, the LDA is not very effective. If the classification problem is cast into a regression one, it is possible to form coarse classes, each having more samples, which enables a better estimation of the within-class scatter matrix. However, if y_i is a numerical output, which can take any value for each input component, it is a challenge to figure out an effective discriminant analysis procedure that can disregard input components that are either irrelevant to output or contribute little to the output.

Such a challenge becomes intertwined with other challenges when a discriminant analysis must be performed incrementally, in a sense that samples are provided one at a time and the mapping approximator must be updated for each training sample.

With these challenges in mind, we introduce a new hierarchical statistical modeling method. Consider the mapping $h : \mathcal{X} \mapsto \mathcal{Y}$, which is to be approximated by a regression tree called an IHDR tree for the high-dimensional space $\mathcal{X}$. Our goal is to automatically derive discriminating features, although no class label is available (other than the numerical vectors in space $\mathcal{Y}$). In addition, for the real-time requirement, we must process each sample (x_i, y_i) to update the IHDR tree using a minimal amount of computation.

With the above motivation, three types of clusters (i.e., neurons) are maintained in IHDR: x-clusters, y-clusters and micro-clusters. The goal of x-clusters and y-clusters is to form partition boundaries in the discriminating feature subspace $\mathcal{D}$ in $\mathcal{X}$ for every internal node. The y-clusters are represented in output space $\mathcal{Y}$ with a goal to provide virtual labels for samples x_i contributing to x-clusters. The x-clusters are represented in input space $\mathcal{X}$ with a goal to partition input space along the discriminating subspace $\mathcal{D}$ of the internal node. In contrast, a micro-cluster is represented as pair (x_i, y_i), where x_i is a primitive prototype in the input space, and y_i is an output corresponding to x_i.

Micro-clusters are needed only by leaf nodes. However, if the IHDR tree allows local deletion (forgetting) so that the physical limit of the storage can be dynamically recruited for changing statistics of the input space, it is desirable for each internal nodes (at least deep internal nodes) to maintain micro-clusters so that the internal nodes can immediately act as leaf nodes when their leaf nodes are deleted.

C. Outline

For notation simplicity, we consider a complete tree where each node has q-children, and the tree has L levels. Thus, the tree has q^l nodes at level l with the root at $l = 0$. Mathematically, the space Y is incrementally partitioned into q^l mutually non-intersecting regions at level l:

$$Y = Y_{l,1} \cup Y_{l,2} \cup \cdots \cup Y_{l,q^l}$$

$l = 1, 2, ..., L$, where $Y_{l,i} \cap Y_{l,j} = \phi$ if $i \neq j$. The partitions are nested. That is, the region $Y_{l,j}$ at level l is further partitioned into q regions at level $l + 1$:

$$Y_{l,j} = Y_{l+1,(j-1)q+1} \cup Y_{l+1,(j-1)q+2} \cup \cdots \cup Y_{l+1,jq} \tag{1}$$

for $j = 1, 2, ..., q^l$. Denote the center of region $Y_{l,j}$ by a vector $\bar{y}_{l,j}$ computed as the mean of the samples in region $Y_{l,j}$. The subregions $\{Y_{l+1,(j-1)q+1}, Y_{l+1,(j-1)q+2}, ..., Y_{l+1,jq}\}$ of $Y_{l,j}$ are the Voronoi diagram of the corresponding centers

$$\{\bar{y}_{l+1,(j-1)q+1}, \bar{y}_{l+1,(j-1)q+2}, ..., \bar{y}_{l+1,jq}\}$$

at level $l + 1$.

Given any training sample pair (x, y), its label at level l is determined by the location of y in $\mathcal{Y}$. If $y \in Y_{l,j}$ for some j, then x has a label j at level l. Among all the sample pairs in the form (x, y), all the x's that share the same label j at level l form the jth x-cluster at level l, represented by the center $\bar{x}_{l,j}$.

In reality, the IHDR tree is updated incrementally from arriving training samples (x_i, y_i). Therefore, it is typically not a complete tree.

D. Double clustering

Each internal node of the IHDR tree incrementally maintains y-clusters and x-clusters, as shown in Fig. 2. There are a maximum of q (e.g., $q = 20$) clusters of each type at each node.

Mathematically, the clustering of $\mathcal{X}$ of each node is conditioned on the class of $\mathcal{Y}$ space. The distribution of each x-cluster is the probability distribution of random variable $x \in \mathcal{X}$ conditioned on random variable $y \in \mathcal{Y}$. Therefore, the conditional probability density is denoted as $p(x|Y \in c_i)$, where c_i is the i-th y-cluster, $i = 1, 2, ..., q$.

The q y-clusters determine the virtual class label of each arriving sample (x, y) based on its y part. The virtual class label is used to determine which x-cluster the input sample (x, y) should update using its x part. Each x-cluster approximates the sample population in the $\mathcal{X}$ space for the samples that belong to it. It may spawn a child node from the current node if a finer approximation is required. The incremental updating is done in the following way. At each node, y in (x, y) finds the nearest y-cluster in Euclidean distance and updates (pulling) the center of the y-cluster. This y-cluster indicates to which corresponding x-cluster the input (x, y) belongs. Then, the x part of

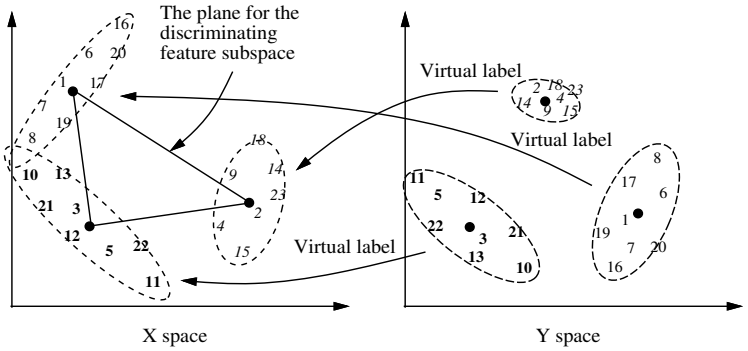


Fig. 2. The Y-clusters in space $\mathcal{Y}$ and the corresponding x-clusters in space $\mathcal{X}$. The number of each sample indicates the time of arrival.

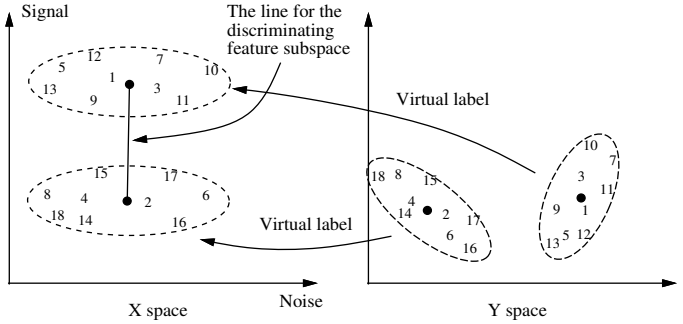


Fig. 3. The discriminating subspace of the IHDR tree disregards components that are not related to outputs. In the figure, the horizontal component is a component in the input space, but is irrelevant to the output. The discriminating feature space, the linear space that goes through the centers of the x-clusters (represented by a vertical line), disregards the irrelevant input components. The number of each sample indicates the time of arrival.

(x, y) is used to update the statistics of the x-cluster (the mean vector and the covariance matrix). The statistics of every x-cluster are then used to estimate the probability for the current sample (x, y) to belong to the x-cluster, whose probability distribution is modeled as a multi-dimensional Gaussian at this level. In other words, each node models a region of the input space $\mathcal{X}$ using q Gaussians. Each Gaussian will be modeled by more small Gaussians in the next tree level if the current node is not a leaf node.

Moreover, the centers of these x-clusters provides essential information for discriminating subspace, since these x-clusters are formed according to the virtual labels in the $\mathcal{Y}$ space. We define the most discriminating feature (MDF) subspace $\mathcal{D}$ as the linear space that passes through the centers of these x-clusters. A total of q centers of the q x-clusters give $q - 1$ discriminating features which span $(q - 1)$-dimensional discriminating space $\mathcal{D}$.

Why is it called the most discriminating space? For example, suppose that there are two components in the input, one contains signals relevant to outputs, and the other is irrelevant to the output as shown in Fig. 3. Although the latter component contains information probably useful for other purposes, it is "noise" as far as the regression task is concerned. Because the centers of the x-clusters have similar coordinates in the "noise" direction and different coordinates in the signal direction, the derived discriminating feature subspace that goes through the centers of the x-clusters successfully catches the signal component and discards the "noise" component, as illustrated in Fig. 3. Other directions are not as good as the MDF direction. Of course, an irrelevant component can be in any orientation and does not have to be along an input axis. The presented technique is a general technical that deals with any orientation.

E. Adaptive local quasi-invariance

Input samples (x, y) are received consecutively in time. The consecutive vectors $x(t)$ and $x(t+1)$ may correspond to an image of a tracked object (e.g., moving away, moving along a direction, rotating, deformation, facial expression, lighting changes, etc). This kind of quasi-invariance is adaptive and local. By adaptive, we mean that the quasi-

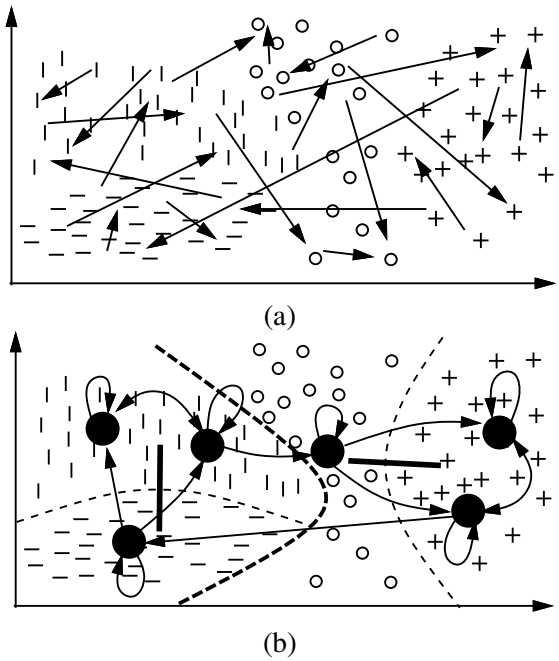


Fig. 4. Automatically sort out a temporal "mess" for local quasi-invariance. A sign, $-, |, \circ$ or $+$, indicates an input sample (x, y), whose position indicates the position of x in the input space $\mathcal{X}$ and whose sign type indicates the corresponding y in the output space $\mathcal{Y}$. Input samples are observed but are not stored in the IHDR tree. (a) A "mess" of temporal transitions: A 2-D illustration of many-to-many temporal trajectories between samples (consecutively firing neuron patterns). (b) Cleaner transitions between fewer primitive prototypes (black circles), enabled by IHDR. The thick dashed curve indicates the nonlinear partition of the input space, $\mathcal{X}$, by the grandparent (internal node). The two thin dashed curves indicate the next-level nonlinear partition by its two children (internal nodes). The two thick straight lines denote the most discriminating subspaces of the two children, respectively. The spread of samples in the direction orthogonal to the solid line represents many types of variations. A solid black cycle indicates a primitive prototype (context state) in one of the four leaf nodes. An arrow between two states indicates observed temporal transitions.

invariance is derived from sensorimotor experience, not hand-designed. By local, we mean that the quasi-invariance is applicable to a local manifold in the input space, not the entire input space. By quasi, we mean that the invariance is approximate, but not absolutely true.

In the literature, there has been no systematic methods that can effectively detail with all kinds of local quasi-invariance in an incremental learning setting. If we compare two vectors using Euclidian distance $\|x(t+1)-x(t)\|$, the variation of every pixel value will be summed in such a distance.

The most discriminating feature subspace derived using the above method can acquire such adaptive local quasi-invariance.

Suppose that a series of training samples are received by the IHDR tree (x_t, y_t), $t = 1, 2, ...,$. For simplicity, we assume that the output y_t takes only four values a_1, a_2, a_3 and a_4, represented by four different signs in Fig. 4. Because of the variations that are irrelevant to the output, the trajectory of $x_1, x_2, ...$ shown in Fig. 4(a) is a mess. That is, there is no clearly visible invariance.

The labels generated by the y-clusters allow x-clusters to be formed according to output. In each internal node, the most discriminating subspace is created, shown as thick line segments in Fig. 4(b). Irrelevant factors while an object is tracked, such as size, position, orientation, deformation, lighting, are automatically disregarded by the feature subspace. There is no need to hand-model what kind of physical invariance that the tracked object exhibits. This is a significant advantage of such internally generated representation (hierarchical feature subspaces). In each leaf node of the IHDR tree, these samples are not stored. They participate in the amnesic average of the primitive prototypes in the leaf node. As shown in Fig. 4(b), due to the nonlinear decision boundaries determined by the parents, not many primitive prototypes are needed in a leaf node if the leaf node is pure (samples are from a single y-cluster).

From Fig. 4(b) we can see that the transition diagrams among the primitive prototypes (i.e., context states) are

similar to a traditional Markov Decision Process (MDP). However, this is an Observation-driven Markov Decision Process (OMDP) as discussed in [21]. The major differences between the OMDP and a traditional MDP include:

1) The states in OMDP are automatically generated from sensory inputs, i.e., observation-driven. The states in a traditional MDP are based on the hand-constructed model about a known task.
2) The states in OMDP are vectors without specified meanings (i.e., distributed numerical representation), while those in a traditional MDP are symbols with hand-assigned meanings (i.e., atomic symbolic representation).
3) The number of states in OMDP is fully automatically determined. The number of states in a traditional MDP is hand-selected.
4) The states in OMDP are adaptive local quasi-invariant in the most discriminating feature subspaces, while states in a traditional MDP are not necessarily so.
5) The OMDP is supported by the hierarchy of most discriminating subspaces in IHDR which is fully automatically and incrementally constructed, while that for a multilevel traditional MDP is hand-designed.

These properties are necessary for automatically generating a task-specific (or context-specific) internal representation through experiences, but during the programming time the programmer of IHDR does not know what tasks that the system will end up learning later, as discussed in [13].

F. Biological view

Fig. 5 illustrates an IHDR tree. Each node in the IHDR tree corresponds to a cortical region. The space $\mathcal{X}$, represented as d-dimensional vectors, is first partitioned coarsely by the root. The root partitions the space $\mathcal{X}$ into q subregions ($q = 4$ in the figure), each corresponds to one of its q children. Each child partitions its own smaller region into q subregions again. Such a coarse-to-fine partition recursively divides input space into increasingly small input regions through the learning experience. The q neurons (called x-clusters) in each node correspond to q feature detectors. With incrementally arriving input vectors (x_i, y_i), these q neurons perform competitive incremental updates for these y-clusters and x-clusters. If a leaf node has received enough samples, it spawns q children. In the leaf node, a collection of micro-clusters in the form (x_i, y_i) are kept.

If x is given, but y is not, a search process is carried through the IHDR tree until a leaf node is reached. The algorithm finds the nearest neighbor x_i of x among the micro-clusters of the form (x_i, y_i) in the leaf node. The corresponding y_i in (x_i, y_i) is the estimated output: $y_i = h(x)$.

III. THE IHDR ALGORITHM

The algorithm incrementally builds an IHDR tree from a sequence of training samples $(x_t, y_t), t = 0, 1, 2, ...$ The deeper a node is in the tree, the smaller the variances of its x-clusters. When the number of samples in a node is too small to give a good estimate of the statistics of q x-clusters, this node is a leaf node.

The mode of IHDR program is run as follows.

Procedure 1: **IHDR Algorithm.** Initialize root. For $t = 0, 1, 2, ...,$ do the following:

- Grab the current sample (x_t, y_t).
- If y_t is given, call update-tree(root, x_t, y_t), otherwise call compute-response(root, x_t, y) and output y.

The following sections explain procedures update-tree and compute-response.

A. Update tree

The following is the incremental algorithm for updating the tree. Given the root of the tree and the training sample (x, y), update the tree using a single training sample (x, y). Each call to update-tree may grow the tree. This procedure handles adaptive cortical plasticity (tree layers). The system parameters: q is the number of maximum children for each internal node. b_s is the number of samples needed per scalar parameter (e.g., $b_s = 20$).

Procedure 2: **Update-tree**(root, x, y).

1) Initialization. Let A be the active node waiting to be searched. A is initialized to be the root.
2) Tree search. While A is not a leaf node, do the following.
   a) Compute response. Compute the response for all neurons in A from input x.
   b) Compete. Among maximum of q neurons in A, the neuron that has the maximum response wins.
   c) The child of A that corresponds to the winning neuron is set as the new A, the next active node.

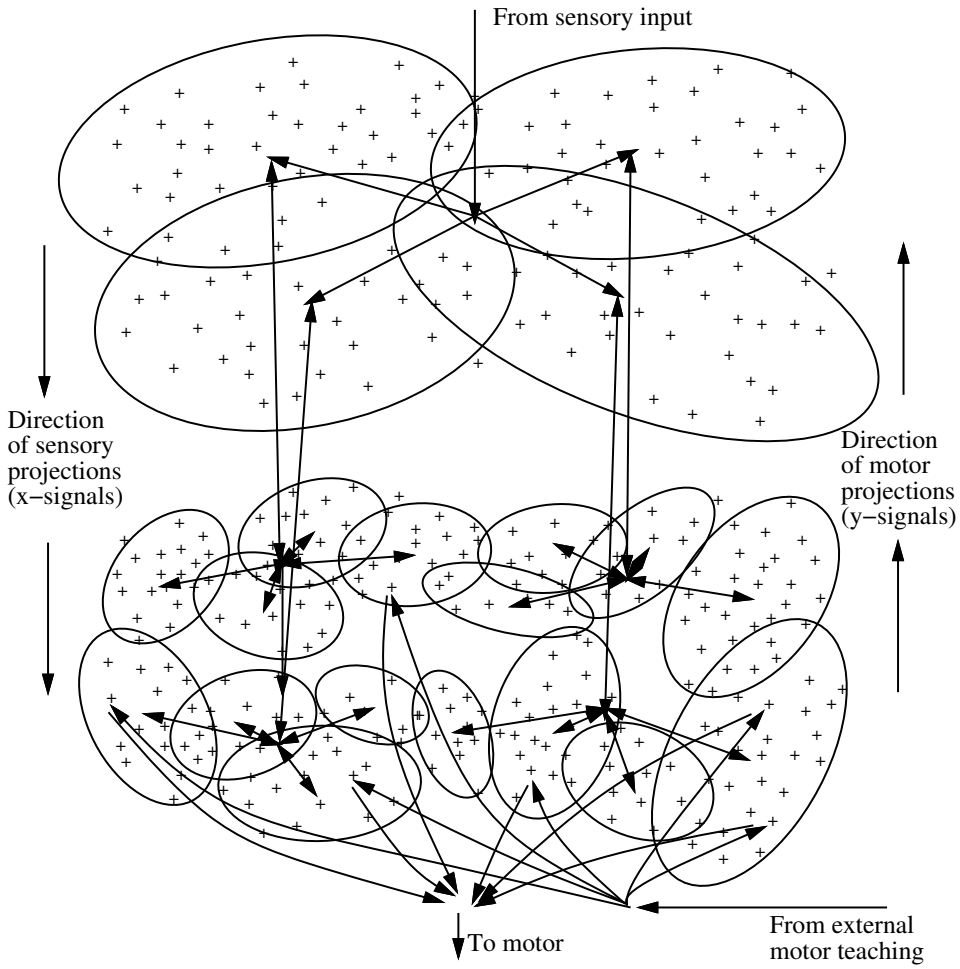


Fig. 5. Illustration of an IHDR tree incrementally developed (self-organized) from learning experience. The sensory space is represented by sensory input vectors, denoted by "+". The sensory space is repeatedly partitioned in a coarse-to-fine way. The upper level and lower level represent the same sensory space, but the lower level partitions the space finer than the upper level does. An ellipse indicates the space for which the neuron at the center is responsible. Each node (cortical region) has q feature detectors (neurons) ($q = 4$ in the figure), which collectively determine to which child node (cortical region) the current sensory input belongs (excites). An arrow indicates a possible path of the signal flow. In every leaf node, prototypes (marked by "+") are kept, each of which is associated with the desired motor output.

3) Update the plastic internal node. Update the parent of A by calling update-note(A, x, y). Note that the ancestors of A are not updated further to avoid large-scale temporal space partition for constrained plasticity.
4) Incremental clustering. Call update-cluster-pair(A, x, y).
5) For the leaf node A, if the number of samples, n, received is larger than a threshold automatically computed based on NSPP $= 2(n - q)/q^2 > b_s$ to be explained later, turn A into an internal node and spawn q leaf nodes as q children of A by distributing the micro-clusters of A into its children.

B. Update node

The above procedure update-node in the update-tree is explained below. Given a node N and a training sample (x, y), it updates the node N using (x, y). This procedure handles adaptive plasticity for cortical patches (within a tree node).

Procedure 3: **Update-node**(N, x, y).

1) Update y-clusters. Let c be the set of y-clusters in node N. Update y-clusters by calling update-clusters(y, c), which returns the index i of the closet cluster y_i.
2) Update the i-th x-cluster associated with y_i. That is, update the mean of the x-cluster using x, employing amnesic average to be explained later.

3) Update the subspace of the most discriminating features of the node N, since the i-th x-cluster has been updated.

C. Update clusters

The above procedure requires the procedure update-clusters. Given sample y and the set of clusters $c = \{y_i \mid i = 1, 2, ..., n\}$, the procedure, update-clusters, updates the clusters in c. This procedure is partially responsible for handling adaptive plasticity for neurons (clusters). The parameters include: q is the bound on the number of clusters in the set c; $\delta_y > 0$ is the resolution in the output space $\mathcal{Y}$.

Procedure 4: **Update-clusters**(y, c).

1) If $n < q$ and $\|y - y_j\| > \delta_y$ (to prevent very similar or even the same samples to form different cluster centers), increment n by one, set new cluster $y_n = y$, add y_n into c and return. Otherwise, do the following.
2) Find the nearest neighbor y_j in the following expression

$$j = \text{argmin}_{1 \le i \le n}\{\|y_i - y\|\},$$

where argmin denotes the argument j that reaches the minimum.
3) Update a certain portion p (e.g., $p = 0.2$, i.e., pulling top 20%) of nearest clusters using amnesic average and return the index j.

D. Update cluster pair

The procedure update-tree also requires the procedure update-cluster-pair. Given a sample (x, y) and the set of cluster pairs $c = \{(x_i, y_i) \mid i = 1, 2, ..., n\}$, the procedure update-cluster-pair updates the best matched cluster in c. This procedure is partially responsible for handling adaptive plasticity for each output neuron (micro-cluster). The parameters include: $b_l > 0$ is the bound on the number of micro-clusters in a leaf node; $\delta_x > 0$ is the resolution in the input space $\mathcal{X}$.

Procedure 5: **Update-cluster-pair**(x, y, c).

1) If $n < b_l$ and $\|x - x_j\| > \delta_x$ (to prevent very similar or even the same samples to form different cluster centers), increment n by one, set a new cluster $(x_n, y_n) = (x, y)$, add (x_n, y_n) into c, and return. Otherwise, do the following.
2) Find the nearest neighbor x_j in the following expression

$$j = \text{argmin}_{1 \le i \le n}\{\|x_i - x\|\}.$$

3) Update cluster x_j by adding the new sample x using amnesic average.
4) Update cluster y_j by adding the new sample y using amnesic average.
5) Return the updated c.

E. Compute response

When the desired output y is not given, IHDR calls the procedure compute-response. Given the root of the tree and sample x, the procedure compute-response computes the response of the tree to produce the final output y. The system parameters include q, the number of maximum children for each internal node.

Procedure 6: **Compute-response**$(root, x, y)$.

1) Do steps "initialization" and "tree search" the same way as the corresponding steps in procedure update-tree, which finds the leaf node A.
2) Compute the output y. Let the set of micro-clusters in A to be $c = \{(x_i, y_i) \mid i = 1, 2, ..., n\}$. Find the best match in the input space:

$$j = \text{argmin}_{1 \le i \le n}\{\|x - x_i\|\}.$$

Assign output y to be the associated y_j, i.e., $y = y_j$, and return.

F. Amnesic average

The amnesic average is motivated by the scheduling of neuronal plasticity which should adaptively change with on-going experience. It is also motivated by the mathematical notion of statistical efficiency in the following sense: To estimate the mean vector θ of a distribution (e.g., the mean vector of observations x_t, $t = 1, 2, 3, ...$, as the synaptic weight vector of the neuron), the sample mean is the most efficient estimator for a large class of time-invariant distributions (e.g., exponential distributions such as Gaussian). By definition, the most efficient estimator Γ has the least expected error variance, $E\|\Gamma - \theta\|^2$, among all possible estimators. However, since the distribution of observations is typically not time-invariant in practice, the amnesic average is needed to adapt to the slowly changing distribution while keeping the estimator to be quasi-optimally efficient.

From the algorithm point of view, in incremental learning, the initial centers of each state clusters are largely determined by early input data. When more data are available, these centers move to more appropriate locations. If these new locations of the cluster centers are used to judge the boundary of each cluster, the initial input data was incorrectly classified. In other words, the center of each cluster contains some earlier data that do not belong to this cluster. To reduce the effect of the earlier data, the amnesic average can be used to compute the center of each cluster. The amnesic average can also track the dynamic change of the input environment better than a conventional average.

The average of t input data $x_1, x_2, ..., x_t$ is given by:

$$\bar{x}^{(t)} = \frac{1}{t}\sum_{i=1}^{t} x_i = \sum_{i=1}^{t} \frac{1}{t} x_i. \tag{2}$$

In the above expression, every x_i is multiplied by a weight $1/t$ and the product is summed. Therefore, each x_i receives the same weight $1/t$. This is called an equally weighted average. If x_i arrives incrementally and we need to compute the average for all the inputs received so far, it is more efficient to recursively compute the current average based on the previous average:

$$\bar{x}^{(t)} = \frac{(t-1)\bar{x}^{(t-1)} + x_t}{t} = \frac{t-1}{t}\bar{x}^{(t-1)} + \frac{1}{t}x_t. \tag{3}$$

In other words, the previous average $\bar{x}^{(t-1)}$ gets a weight $(t-1)/t$ and the new input x_t gets a weight $1/t$. These two weights sum to one. The recursive equation Eq. (3) gives an equally weighted average. In amnesic average, the new input gets more weight than old inputs as given in the following expression:

$$\bar{x}^{(t)} = \frac{t-1-\mu}{t}\bar{x}^{(t-1)} + \frac{1+\mu}{t}x_t. \tag{4}$$

where $\mu \geq 0$ is an amnesic parameter. The two weights still always sum to one. For example $\mu = 1$, which means that the weight for the new sample is doubled.

The amnesic weight for the new data $(1+\mu)/t$ will approach zero when t goes to infinity. This means that when t grows very large without bound, the new data would hardly be used and thus the system will hardly adapt. We would like to enable μ to change dynamically. Thus, we denote μ as $\mu(t)$.

We use two transition points, t_1 and t_2. When $t \leq t_1$, we like to let $\mu(t) = 0$ to fully use the limited data. When $t_1 < t \leq t_2$, we let μ change linearly from 0 to c (e.g., $c = 1$). When $t_2 < t$, we let $\mu(t)$ to grow slowly and its growth rate gradually approaches $1/m$. The above discussion leads to the following expression for $\mu(t)$:

$$\mu(t) = \begin{cases} 0 & \text{if } t \leq t_1 \\ c(t-t_1)/(t_2-t_1) & \text{if } t_1 < t \leq t_2 \\ c + (t-t_2)/m & \text{if } t_2 < t \end{cases}$$

Since $\lim_{t\to\infty}(1+\mu(t))/t = 1/m$, when t grows without bound, the weight for the new data $x(t)$ is approximately the same as that of the non-amnesic average with m data points. Such a growing $\mu(t)$ enables the amnesic average to track the non-stationary random input process x_t, whose mean changes slowly over time.

The update expression for incrementally computing sample covariance matrix is as follows:

$$\Gamma_x^{(t)} = \frac{t-1-\mu(t)}{t}\Gamma_x^{(t-1)} + \frac{1+\mu(t)}{t}(x_t - \bar{x}^{(t)})(x_t - \bar{x}^{(t)})^\top \tag{5}$$

The amnesic function $\mu(t)$ changes with t as we discussed above.

Note that the above expression assumes t degrees of freedom, instead of $t-1$ in the batch computation of the sample covariance matrix, for the following reason: Even with a single sample x_1, the corresponding covariance matrix should not be estimated as a zero vector, since x_1 is never exact if it is measured from a physical event. For example, the initial variance matrix $\Gamma_x^{(1)}$ can be estimated as $\sigma^2 I$, where σ^2 is the expected digitization noise in each component and I is the identity matrix of the appropriate dimension.

IV. Discriminating Subspace and Optimal Class Boundary

Each internal node automatically drives the subspace of the most discriminating features (MDF) for superior generalization. The MDF subspace is tuned to each internal node for characterizing the samples assigned to the node. For our partition purpose, each child of the internal node represents a class. Probability-based optimal class boundary is needed to partition the input space of the internal node, based on the MDF subspace.

A. Discriminating subspace

Due to a very high input dimension (typically at least a few thousand), for computational efficiency, we should not represent data in the original input space $\mathcal{X}$. Further, for better generalization characteristics, we should use discriminating subspaces $\mathcal{D}$ in which input components that are irrelevant to output are disregarded.

We first consider x-clusters. Each x-cluster is represented by its mean as its center and the covariance matrix as its size. However, since the dimension of the space $\mathcal{X}$ is typically very high, it is not practical to directly keep the covariance matrix. If the dimension of $\mathcal{X}$ is 3000, for example, each covariance matrix requires $3000 \times 3000 = 9,000,000$ numbers! We adopt a more efficient method that uses subspace representation.

As explained in Section II-A, each internal node keeps up to q x-clusters. The centers of these q x-clusters are denoted by,

$$C = \{c_1, c_2, ..., c_q \mid c_i \in \mathcal{X}, i = 1, 2, ..., q\}. \tag{6}$$

The locations of these q centers tell us the subspace $\mathcal{D}$ in which these q centers lie. $\mathcal{D}$ is a discriminating space since the clusters are formed based on the clusters in output space $\mathcal{Y}$.

The discriminating subspace $\mathcal{D}$ can be computed as follows. Suppose that the number of samples in cluster i is n_i and thus the grand total of samples is $n = \sum_{i=1}^{q} n_i$. Let $\bar{C}$ be the mean of all the q x-cluster centers. $\bar{C} = \frac{1}{n}\sum_{i=1}^{q} n_i c_i$. The set of scatter vectors from their centers then can be defined as $s_i = c_i - \bar{C}$, $i = 1, 2, ..., q$. These q scatter vectors are not linearly independent because their sum is equal to a zero vector. Let S be the set that contains these scatter vectors: $S = \{s_i \mid i = 1, 2, ..., q\}$. The subspace spanned by S, denoted by span(S), consists of all the possible linear combinations from the vectors in S, as shown in Fig. IV-A.

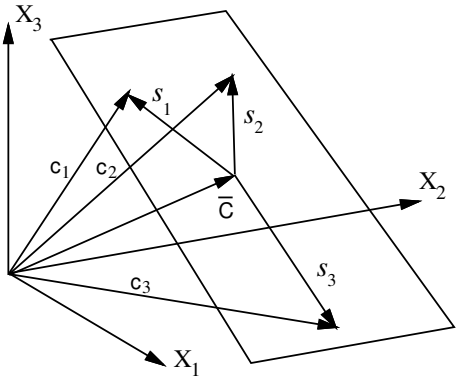


Fig. 6. The linear manifold represented by $\bar{C} + \text{span}(S)$, the spanned space from scatter vectors translated by the center vector $\bar{C}$.

The orthonormal basis $a_1, a_2, ..., a_{q-1}$ of the subspace span(S) can be constructed from the radial vectors $s_1, s_2, ..., s_q$ using the *Gram-Schmidt Orthogonalization* (GSO) procedure:

Procedure 7: `GSO Procedure`. Given vectors $s_1, s_2, ..., s_{q-1}$, compute the orthonormal basis vectors $a_1, a_2, ..., a_{q-1}$.

1) $a_1 = s_1/\|s_1\|$.
2) For $i = 2, 3, ..., q-1$, do the following:
   a) $a'_i = s_i - \sum_{j=1}^{i-1}(s_i^T a_j)a_j$.
   b) $a_i = a'_i/\|a'_i\|$.

In the above procedure, a degeneracy occurs if the denominator is zero. In the first step, the generacy means s_1 is a zero vector. In the remaining steps, it means that the corresponding vector s_i is a linear combination of the previous radial vectors. If a degeneracy occurs, the corresponding s_i should be discarded in the computation for the basis vectors. The number of basis vectors that can be computed by the GSO procedure is the number of linearly independent radial vectors in S.

Given a vector $x \in \mathcal{X}$, we can compute its scatter part $s = x - \bar{C}$. Then compute the projection of x onto the linear manifold by $f = M^T s$, where $M = [a_1, a_2, \ldots, a_{q-1}]$. We call the vector f the discriminating features of x in the linear manifold S. The mean and the covariance of the clusters are then computed on the discriminating subspace.

The Fisher's linear discriminant analysis by [20] finds a subspace that maximizes the ratio of between-cluster scatter and within-cluster scatter. Since we decided to use the entire discriminating space $\mathcal{D}$, we do not need to consider the within-cluster scatter here in finding $\mathcal{D}$ since probability will be used in defining distance. This simplifies the computation for discriminating features. Once we find this discriminating space $\mathcal{D}$, we will use size-dependent negative-log-likelihood (SDNLL) distance as discussed in Section IV-B to take care of the reliability of each dimension in $\mathcal{D}$ using information that is richer than the within-cluster scatter.

B. The probability-based metrics

The subspace of the most discriminating features is automatically derived from a constraint that the dimension allowed is $q-1$. Within this subspace, we need to automatically determine the *optimal* class boundaries for every child node, based on the estimated probability distribution. Different models of probability distribution correspond to different distance metrics.

To relate the distance metrics with the response of neurons, we model the response of a neuron from an input x as $g(1/d(x,c))$ where $d(x,c)$ is distance from x and the center vector c (i.e., the synaptic vector) of the neuron, and g is a smooth sigmoidal nonlinear function.

Let us consider the negative-log-likelihood (NLL) defined from Gaussian density of dimension $q-1$:

$$G(x, c_i) = \frac{1}{2}(x - c_i)^T \Gamma_i^{-1}(x - c_i) + \frac{q-1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\Gamma_i|). \tag{7}$$

We call it Gaussian NLL for x to belong to the cluster i. c_i and Γ_i are the cluster sample mean and sample covariance matrix, respectively, computed using the amnesic average in Section III-F. Similarly, we define Mahalanobis NLL and Euclidean NLL as:

$$M(x, c_i) = \frac{1}{2}(x - c_i)^T \Gamma^{-1}(x - c_i) + \frac{q-1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\Gamma|), \tag{8}$$

$$E(x, c_i) = \frac{1}{2}(x - c_i)^T \rho^2 I^{-1}(x - c_i) + \frac{q-1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\rho^2 I|), \tag{9}$$

where Γ is the within-class scatter matrix of each node — the average of the covariance matrices of the q clusters:

$$\Gamma = \frac{1}{q-1}\sum_{i=1}^{q-1}\Gamma_i, \tag{10}$$

computed using the same technique of the amnesic average.

Suppose that the input space is $\mathcal{X}$ and the discriminating subspace for an internal node is $\mathcal{D}$. The Euclidean NLL treats all of the dimensions in the discriminating subspace $\mathcal{D}$ the same way, although some dimensionalities can be more important than others. It has only one parameter ρ to estimate. Thus, it is the least demanding among the

three NLL in the richness of the observation required. When very few samples are available for all of the clusters, the Euclidean likelihood is the suited likelihood.

The Mahalanobis NLL uses the within-class scatter matrix Γ computed from all of the samples in all of the q x-clusters. Using Mahalanobis NLL as the weights for subspace $\mathcal{D}$ is equivalent to using Euclidean NLL in the basis computed from Fisher's LDA procedure. It decorrelates all dimensions and weights each dimension using a different weight. The number of parameters in Γ is $q(q-1)/2$, and thus, the Mahalanobis NLL requires more samples than the Euclidean NLL.

The Mahalanobis NLL does not treat different x-clusters differently because it uses a single within-class scatter matrix Γ for all of the q x-clusters in each internal node. For Gaussian NLL, $L(x, c_i)$ in Eq. (7) uses the covariance matrix Γ_i of x-cluster i. In other words, the Gaussian NLL not only decorrelates the correlations but also applies a different weight at a different location along each rotated basis. However, it requires that each x-cluster has enough samples to estimate the $(q-1) \times (q-1)$ covariance matrix. Thus, it is the most demanding on the number of observations. Note that the decision boundary of the Euclidean NLL and the Mahalanobis NLL is linear, but by the Gaussian NLL, it is quadratic.

C. Automatic soft transition among different matrices

Due to the challenge of incrementally arrived samples, different distance metrics are needed based on the number of available samples in each node.

We would like to use the Euclidean NLL when the number of samples in the node is small. Gradually, as the number of samples increase, the within-class scatter matrix of q x-clusters are better estimated. Then, we would like to use the Mahalanobis NLL. When a cluster has very rich observations, we would like to use the full Gaussian NLL for it. We would like to make an automatic transition when the number of samples increase. We define the number of samples n_i as the measurement of maturity for each cluster i. $n = \sum_{i=1}^{q} n_i$ is the total number of samples in a node.

For the three types of NLLs, we have three matrices, $\rho^2 I$, Γ, and Γ_i. Since the reliability of the estimates are well indicated by the number of samples, we consider the number of scales received to estimate each parameter, called the number of scales per parameter (NSPP), in the matrices. The NSPP for $\rho^2 I$ is $(n-1)(q-1)$, since the first sample does not give any estimate of the variance and each independent vector contains $q-1$ scales. For the Mahalanobis NLL, there are $(q-1)q/2$ parameters to be estimated in the (symmetric) matrix Γ. The number of independent vectors received is $n-q$ because each of the q x-clusters requires a vector to form its mean vector. Thus, there are $(n-q)(q-1)$ independent scalars. The NSPP for the matrix Γ is $\frac{(n-q)(q-1)}{(q-1)q/2} = \frac{2(n-q)}{q}$. To avoid the value becoming negative when $n < q$, we take NSPP for Γ to be $\max\left\{\frac{2(n-q)}{q}, 0\right\}$. Similarly, the NSPP for Γ_i for the Gaussian NLL is $\frac{1}{q}\sum_{i=1}^{q} \frac{2(n_i-1)}{q} = \frac{2(n-q)}{q^2}$. Table I summarizes the result of the NSPP values of the above derivation. The procedure update-tree used NSPP for Gaussian NLL.

TABLE I

CHARACTERISTICS OF THREE TYPES OF SCATTER MATRICES

Type	Euclidean $\rho^2 I$	Mahalanobis Γ	Gaussian Γ_i
NSPP	$(n-1)(q-1)$	$\frac{2(n-q)}{q}$	$\frac{2(n-q)}{q^2}$

A bounded NSPP is defined to limit the growth of NSPP so that other matrices that contain more scalars can take over when there are a sufficient number of samples for them. Thus, the bounded NSPP for $\rho^2 I$ is $b_e = \min\{(n-1)(q-1), n_s\}$, where n_s denotes the soft switch point for the next, more complete matrix to take over. To estimate n_s, we consider a series of random variables drawn independently from a distribution with a variance σ^2, the expected sample mean of n random variables has an expected variance $\sigma^2/(n-1)$. We can choose a switch confidence value α for $1/(n-1)$. When $1/(n-1) = \alpha$, we consider that the estimate can take about a 50% weight. Thus, $n = 1/\alpha + 1$. As an example, let $\alpha = 0.05$ meaning that we trust the estimate with 50% weight when the expected variance of the estimate is reduced to about 5% of that of a single random variable. This is like a confidence value in hypothesis testing except that we do not need an absolute confidence and a relative one suffices. We then get $n = 21$, which leads to $n_s = 21$.

The same principle applies to Mahalanobis NLL and its bounded NSPP for Γ is $b_m = \min\left\{\max\left\{\frac{2(n-q)}{q}, 0\right\}, n_s\right\}$. It is worth noting that the NSPP for the Gaussian NLL does not need to be bounded, since among our models it is the best estimate with increasing number of samples beyond. Thus, the bounded NSPP for Gaussian NLL is $b_g = \frac{2(n-q)}{q^2}$.

How do we realize automatic transition? We define a *size-dependent scatter matrix* (SDSM) W_i as a weighted sum of three matrices:

$$W_i = w_e\rho^2 I + w_m\Gamma + w_g\Gamma_i, \tag{11}$$

where $w_e = b_e/b$, $w_m = b_m/b$, $w_g = b_g/b$ and b is a normalization factor so that these three weights sum to 1: $b = b_e + b_m + b_g$. Using this size-dependent scatter matrix W_i, the *size-dependent negative log likelihood* (SDNLL) for x to belong to the x-cluster with center c_i is defined as:

$$L(x, c_i) = \frac{1}{2}(x - c_i)^T W_i^{-1}(x - c_i) + \frac{q-1}{2}\ln(2\pi) + \frac{1}{2}\ln(|W_i|). \tag{12}$$

With b_e, b_m, and b_g changing automatically, $(L(x, c_i))$ transits smoothly through the three NLLs. It is worth noting the relation between the LDA and SDNLL metric. The LDA in space $\mathcal{D}$ with original basis η gives a basis ϵ for a subspace $\mathcal{D}' \subseteq \mathcal{D}$. This basis ϵ is a properly oriented and scaled version for $\mathcal{D}$ so that the within-cluster scatter in $\mathcal{D}'$ is a unit matrix, [20] (Sections 2.3 and 10.2). In other words, all of the basis vectors in ϵ for D' are already weighted according to the within-cluster scatter matrix Γ of $\mathcal{D}$. If $\mathcal{D}'$ has the same dimension as $\mathcal{D}$, the Euclidean distance in $\mathcal{D}'$ on ϵ is equivalent to the Mahalanobis distance in $\mathcal{D}$ on η, up to a global scale factor. However, if the covariance matrices are very different across different x-clusters and each of them has enough samples to allow a good estimate of every covariance matrix, the LDA in space $\mathcal{D}$ is not as good as the Gaussian likelihood because covariance matrices of all x-clusters are treated the same in the LDA, while Gaussian likelihood takes into account such differences. The SDNLL in (12) allows automatic and smooth transition between three different types of likelihood: Euclidean, Mahalanobis and Gaussian, according to the predicted effectiveness of each likelihood. Hwang & Weng [11] demonstrated that SDNLL effectively deals with various sample cases, including small, moderate, large, and unbalanced samples.

D. Computational considerations

Due to the challenge of real-time computation, an efficient non-iterative computational scheme must be designed for every internal node.

The matrix weighted squared distance from a vector $x \in \mathcal{X}$ to each x-cluster with center c_i is defined by,

$$d^2(x, c_i) = (x - c_i)^T W_i^{-1}(x - c_i), \tag{13}$$

which is the first term of Eq.(12).

This distance is computed only in $(q-1)$-dimensional space using the basis M. The SDSM W_i for each x-cluster is then only a $(q-1)\times(q-1)$ square symmetric matrix, of which only $q(q-1)/2$ parameters need to be estimated. When $q = 6$, for example, this number is 15.

Given a column vector v represented in the discriminating subspace with an orthonormal basis whose vectors are the columns of matrix M, the representation of v in the original space $\mathcal{X}$ is $x = Mv$.

To compute the matrix weighted squared distance in Eq.(13), we use a numerically efficient method, Cholesky factorization, [22] (Sec. 4.2). The Cholesky decomposition algorithm computes a lower triangular matrix L from W so that W is represented by $W = LL^T$ as stated in the following procedure.

Procedure 8: Cholesky factorization: Given an $n \times n$ positive definite matrix $A = [a_{ij}]$, compute lower triangular matrix $L = [l_{ij}]$ so that $A = LL^T$.

1) For $i = 1, 2, ..., n$ do:

   a) For $j = 1, 2, ..., i-1$ do:

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk})/l_{jj},$$

   b) $l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$.

With the lower triangular matrix L, we first compute the difference vector from the input vector x and each x-cluster center c_i: $v = x - c_i$. The matrix weighted squared distance is given by:

$$d^2(x, c_i) = v^T W_i^{-1} v = v^T (LL^T)^{-1} v = (L^{-1}v)^T (L^{-1}v). \tag{14}$$

We solve for y in the linear equation $Ly = v$ and then $y = L^{-1}v$ and $d^2(x, c_i) = (L^{-1}v)^T(L^{-1}v) = \|y\|^2$. Since L is a lower triangular matrix, the solution for y in $Ly = v$ is trivial since we simply use the back-substitution method as described in [23] (page 42).

Typically, many different clusters in leaf nodes point to the same output vector as the label. To get the class label quickly, each cluster (or sample) (x_i, y_i) in the leaf node of the regression tree has a link to label l_i so that when (x_i, y_i) is found as a good match for the unknown input x, l_i is directly the output as the class label. There is no need to search for the nearest neighbor in the output space for the corresponding class label.

Therefore, the incrementally constructed IHDR tree gives a coarse-to-fine probability model. If we use a Gaussian distribution to model each x-cluster, this is a *hierarchical version* of the well-known mixture-of-Gaussian distribution models: the deeper the tree is, the more Gaussians are used and the finer these Gaussians are. At shallow levels, the sample distribution is approximated by a mixture of large Gaussians (with large variances). At deep levels, the sample distribution is approximated by a mixture of many small Gaussians (with small variances). The multiple search paths guided by probability allow a sample x, that falls in-between two or more Gaussians at each shallow level, to explore the tree branches that contain its neighboring x-clusters. Those x-clusters to which the sample (x, y) has little chance to belong to are excluded from further exploration. This results in the well-known logarithmic time complex for tree retrieval: $O(d \log n)$ where n is the number of leaf nodes in the tree, and d is the dimension of the input vector, assuming that the number of samples in each leaf node is bounded above by a constant (e.g., 50). See [11] for the proof of the logarithmic time complexity.

V. The Experimental Results

Several experiments were conducted using the proposed incremental algorithm. First, we present the experimental results using synthetic data. Then we show the power of the method using real face images as high dimensional input vectors for classification. For the regression problem, we demonstrated the performance of our algorithm for autonomous navigation where input is the current image and output is the required steering signal.

A. For synthetic data

The purpose of using synthetic data is to examine the near optimality potential of our new algorithm with the known distributions as a ground truth (but our algorithm does not know the distribution).

The synthetic experiment presented here is for 3-D, where the discriminating space D is 2-D. There were 3 clusters, each being modeled by a Gaussian distribution with means, respectively, $(0, 0, 0)$, $(5, 0, 0)$, $(0, 5, 0)$ and covariance matrices

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 4 & 0 & 0 \\ 0 & 2.25 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

There are 500 samples per class for training and testing, respectively.

Fig. 7 plots these samples in (x_1, x_2) plane, along with the decision boundaries from five types of distance metrics: (1) The Bayesian ground-truth decision rule (Bayesian-GT) where all the ground truths about distributions are known (i.e., the rule does not use samples). (2) The Euclidean distance measured from a scalar covariance matrix $\rho^2 I$. (3) The Mahalanobis distance using a single estimated covariance matrix S_w. (4) The Gaussian NLL (Beyesian-Sample) using estimated full sample covariance matrices for all clusters. (5) Our SDNLL.

Table II shows the classification errors of (1) Bayesian-GT, (2) Bayesian-Sample), and (3) the new distance metrics (SDNLL). It shows that the classification errors are very close among all the measurements. Of course, our method would not be able to be close to the Bayesian error rates if there are not enough samples per class.

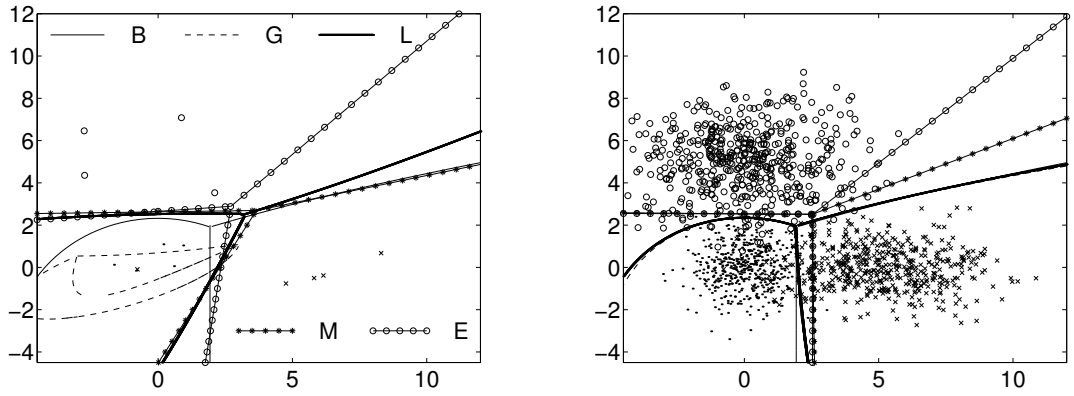


Fig. 7. Training samples in (x_1, x_2) plane and the decision boundaries estimated by different distance metrics. (a) A small-sample case. Lines 'B': Bayesian-GT. Lines 'E': Euclidean distance. Lines 'G': Gaussian NLL. Lines 'M': Mahalanobis distance. Lines 'L': our SDNLL. (b) A large-sample case.

TABLE II
ERROR RATES FOR 3-D SYNTHETIC DATA

	Bayesian-GT	Bayesian-Sample	SDNLL
class 1	7%	7.2%	4%
class 2	6.8%	8.2%	4.6%
class 3	1.6%	2.0%	4.8%

B. For real face data

A critical test of the presented algorithm is to directly deal with high-dimensional, multi-media data, such as images, video, or speech. We present the results for images here. Each image of m rows and n columns is considered a mn-dimensional vector, where each component of the vector corresponds to the intensity of each pixel. Statistical methods have been applied directly to these vectors of high dimension. This type of approach has been very successful in the field of computer vision and now has been commonly called appearance-based methods, [15], [24] and [17]. Although appearance-based methods themselves do not have invariance in position, size, and orientation when applied to appearance-based object recognition, they have been well-accepted for their superior performance when input images are preprocessed images with roughly normalized position and size.

The first experiment used face images from the Weizmann Institute in Israel. The image database was constructed from 28 human subjects, each having thirty images: all combinations of two different expressions under three different lighting conditions with five different orientations. An example of the face images from one human subject is shown in Fig 9. The preprocessed images have a resolution of 88×64, resulting in a input dimension of 5632. The task here is to classify images into a person's ID as class label. We used the mean of all training images of each person as the corresponding y vector. For this classification problem, a node does not spawn children as long as it contains only samples of the same class (pure node).

For our disjoint test, the data set was divided into two groups: training set and testing set. The training set contains 504 face images. Each subject contributed 18 face images in the training set. The 18 images include three different poses, three different lightings, and two different expressions. The remaining 336 images were used for the testing set. Each subject had 12 images for testing, which include two different poses, three different lightings, and two expressions. In order to present enough training samples for the IHDR algorithm to build a stable tree, we artificially increased the samples by presenting training samples to the program 20 times (20 epochs). Table III compares different appearance-based methods. Table IV completes a 2-fold cross-validation with Table III. In other words, the test and training sets in Table III exchange their roles in Table IV. We used 95% sample variance in determining the number of basis vectors (eigenvectors) in the principal component analysis (PCA). The 95% of

Fig. 8. Face images from the Weizmann Institute. The training images of one subject. 3 lightings, 2 expressions, and 3 orientations are included in the training set.

Fig. 9. Face images from the Weizmann Institute. The testing images of one subject. 3 lightings, 2 expressions, and 2 orientations are included in the testing set.

variance results in about 98 eigenvectors which are much less than that of NN (5632-D!). The PCA organized with a binary tree was faster than straight NN as shown in the Table III. It is the fastest algorithm among all of the methods we tested but the performance is worse than those of the PCA and NN. The accuracy of the LDA is the third best. Our new IHDR method is faster than the LDA and resulted in the lowest error rate.

For comparison, we also applied the support vector machines (SVM) by [25] to this image set. The support vector machines utilizes a structural risk minimization principle, [26]. It results in a maximum separation margin and the solution depends only on the training samples (support vectors) which are located on the supporting planes. The SVM has been applied to both classification and regression problems. We used the SVM software obtained from Royal Holloway, University of London, by [27], for this experiment. To avoid excessively high dimension that SVM is unable deal with, we applied the PCA first to provide feature vectors for the SVM.[2] The best result we obtained, by tuning the parameters of the software, is reported in Table III. The data showed that the recognition rate of the SVM with the PCA is similar to that of the PCA alone. However, the SVM with the PCA is faster than the PCA. This is because the SVM has more compact representation and the PCA alone needs to conduct linear search for every training sample. However, the SVM is not suited for the real-time, incremental learning that our applications require because its training is extremely slow, in addition to its inferior performance shown in Tables III and IV.

The error rate of the proposed IHDR algorithm was compared with some major tree classifiers. CART of [5] and C5.0 of [28] are among the best known classification trees[3]. However, like most other decision trees, they are univariate trees in that each internal node used only one input component to partition the samples. This means that the partition of samples is done using hyperplanes that are orthogonal to one axis. We do not expect that this type of tree can work well in a high dimensional or highly correlated space. Thus, we also tested a more recent multivariate tree OC1 of [10]. We realize that these trees were not designed for high-dimensional spaces like those from the images. Therefore, to fully explore their potential, we also tested the corresponding versions by performing the PCA before using CART, C5.0, and OC1 and called them CART with the PCA, C5.0 with the PCA, and OC1 with the PCA, respectively, as shown in Tables III and IV.

Further, we compared the batch version of our HDR algorithm. Originally we expected the batch method to out-perform the incremental one. However, the error rate of the IHDR tree turned out lower than that of the HDR tree for this set of data. A major reason for this is that the same training samples may distribute in different leaf nodes for the IHDR tree because we ran several iterations during training. For the batch version, each training

[2]The software failed when we used the original image input with dimension 5362.

[3]We have also experimented the same data set using CART implemented by OC1. The performance is significantly worse than those reported in the Table III.

sample can only be allocated to a single leaf node.

TABLE III
THE PERFORMANCE FOR WEIZMANN FACE DATA SET 1

Method	Error rate	Avg. testing time (msec)
PCA	12.8%	115
PCA tree	14.58%	34
LDA	2.68%	105
NN	12.8%	164
SVM with PCA	12.5%	90
C5.0 with PCA	45.8%	95
OC1 with PCA	44.94%	98
HDR tree	1.19%	78
IHDR tree	0.6%	74

TABLE IV
THE PERFORMANCE FOR WEIZMANN FACE DATA SET 2 (SWAPPING TRAINING AND TEST SETS FROM SET 1)

Method	Error rate	Avg. testing time (msec)
PCA	15.3%	81
PCA tree	17.4%	29
LDA	9.52%	75
NN	13.5%	108
SVM with PCA	7.41%	69
C5.0 with PCA	42.3%	73
OC1 with PCA	41.5%	72
HDR tree	3.37%	56
IHDR tree	3.97%	53

For a large sample test, we performed an experiment on the FERET (Face Recognition Technology) face image set from [29]. The FERET face image set was developed under the FERET project sponsored by the US Army Research Laboratory. Not all of the images of the FERET project were available publicly due to the need to keep some of the test sets. We used the front view images that were made publicly available to us during our participation with blind FERET tests. It contains face images of 34 individuals, under different lighting conditions and expressions. Each person had three face images for the purpose of training. The other face image was used for testing.

A face normalization program was used to translate, scale, and rotate each face image into a canonical image of 88 rows and 64 columns so that eyes are located at prespecified positions as shown in Fig 10.

To reduce the effect of background and non-facial areas, image pixels are weighted by a number that is a function of the radial distance from the image center. Further, the image intensity is masked by a linear function so that the minimum and maximum values of the images are 0 and 255, respectively. Fig 10 shows the effect of such a normalization procedure.

A summary of comparison is listed in Table V. Notice that the training time is measured for the total time to train the corresponding system. The testing time is the average time per query. To make a fair comparison, training and testing times for the PCA are included in C5.0 with the PCA, OC1 with the PCA, and CART with the PCA. As shown, none of these existing decision trees can deal with the FERET set well, not even the versions that use the PCA as a preprocessing step. The batch version of the proposed algorithm (HDR) tree shares the same error rate as the IHDR tree. The HDR tree is faster than the IHDR in both training and testing. This is because we ran several epochs for the IHDR tree and the IHDR tree has more redundant information inside.

In order to display how the IHDR tree converges, Fig 11 shows the error rates vs. epoch plot. Each epoch means that the training set is fed into the algorithm once, one image at a time. As can be seen, the resubstitution error rate converges to zero at the 5th epoch. The testing error rate reaches zero at 6th epoch.

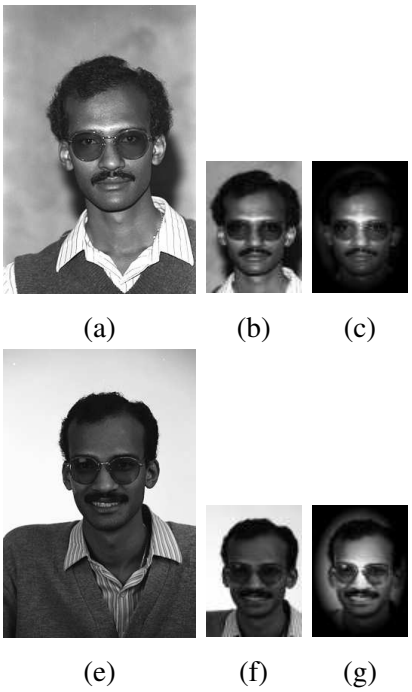


Fig. 10. The demonstration of the image normalization process. (a), (e) The original image from the FERET data set. (b), (f) The normalized image. (c), (g) The masked image.

TABLE V
THE PERFORMANCE OF DECISION TREE FOR THE FERET TEST

Method	Error rate		Time (sec)	
	Training	Testing	Training	Testing
CART	10%	53%	2108.00	0.029
C5.0	41%	41%	21.00	0.030
OC1	6%	56%	2206.00	0.047
CART with PCA	11%	53%	10.89	0.047
C5.0 with PCA	6%	41%	9.29	0.047
OC1 with PCA	5%	41%	8.89	0.046
HDR tree	0%	0%	12.25	0.027
IHDR tree	0%	0%	23.41	0.041

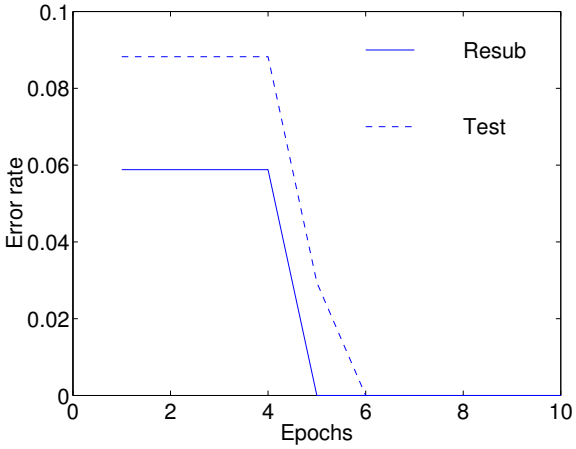


Fig. 11. The performance plot of the FERET face test 1. The plot of error rate vs. number of epochs. The "Resub" line means the resubstitution error rate. The "Test" line represents the testing error rate.

C. Autonomous navigation by a robot

All of the tasks above are classification tasks. We present the result for a regression task. Our vision-based navigation system accepts an input image, X, and outputs the control signal, C, to update the heading direction of the vehicle. The navigator can be denoted by a function f that maps the input image space $\mathcal{X}$ to control signal space $\mathcal{C}$. The learning process of the autonomous navigation problem can then be realized as a function approximation. This is a very challenging task since the function to be approximated is for a very high-dimensional input space and the real application requires the navigator to perform in real-time.

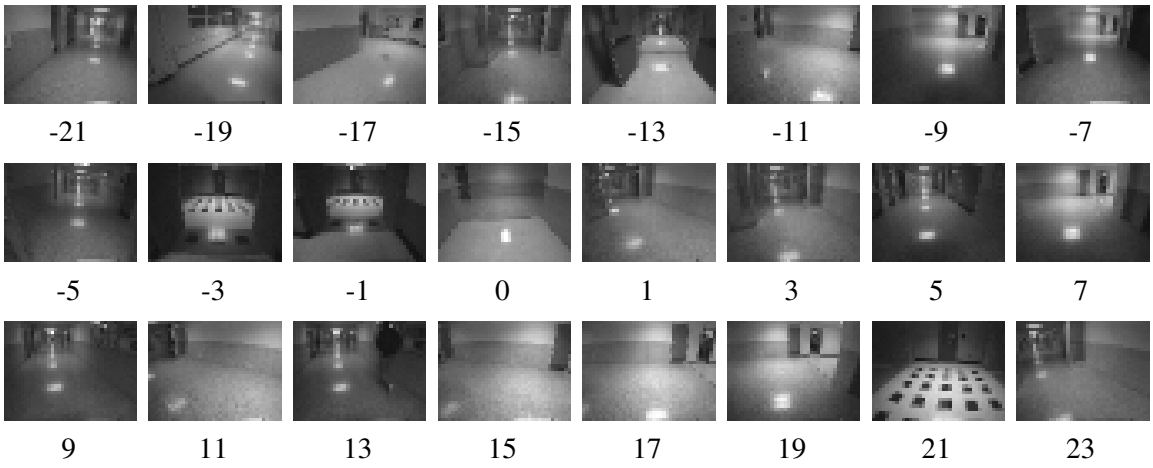


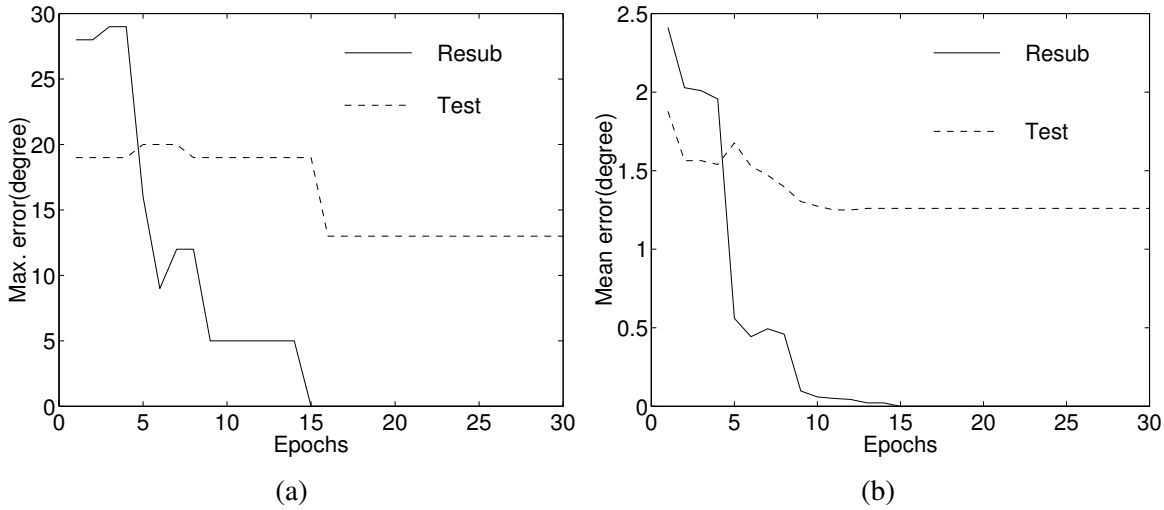
Fig. 12. A subset of images used in autonomous navigation problem. The number right below the image shows the needed heading direction (in degrees) associated with that image.

We applied our algorithm to this challenging problem. Some of the example input images are shown in Fig 12. As can be seen in the images, surface specularity, floor tile patterns, various doors and windows along the straight sections and turning corners pose an extremely challenging computer vision task for traditional vision systems. Our method uses the entire image directly and the feature basis vectors of subspaces are automatically derived online instead of manually designed off-line. Totally, 318 images with the corresponding heading directions were used for training. Desired heading direction, which must keep the robot in the center of straight hallways and turn properly at 6 kinds of corners, was recorded online at each time. The resolution of each image is 30 by 40. We used the other 204 images to test the performance of the trained system. Fig. 13 shows the maximum error rates and the mean error rates versus the number of training epochs. Both the maximum error and the mean error converge around the 15th epoch. Fig. 14 gives plots of the histograms of the error rates at different epochs. As shown, even after the first epoch, the performance of the IHDR tree is already reasonably good due to our coarse-to-fine approximation scheme using the tree. With the increase of the epochs, we observed the improvement of the maximum error and mean error. The improvement stopped at the 15th epoch because the algorithm did not use any new training samples in that epoch and the system has perfectly fit the existing training data set. Our test on a real mobile robot has shown that a system of such an error level of epoch 5 can navigate the robot very reliably for hours even with passers-by until the on board batteries are exhausted.

We also compare our experimental results with two artificial neural networks (ANN) with a consideration that the pattern-by-pattern training mode of artificial neural networks is also an incremental learning method. A two-layer feed-forward (FF) network and a radial basis function (RBF) network were used to train and test for the mapping from the image space to control signal space using the same data set as used in our IHDR tree algorithm. The results are listed in Table VI which shows that the mean error of FF was 60% larger ($2.00/1.25 = 1.6$) than that of IHDR while that of RBF was 47% larger. The maximum error of IHDR was slightly larger, but the difference was not significant to draw a general conclusion here.

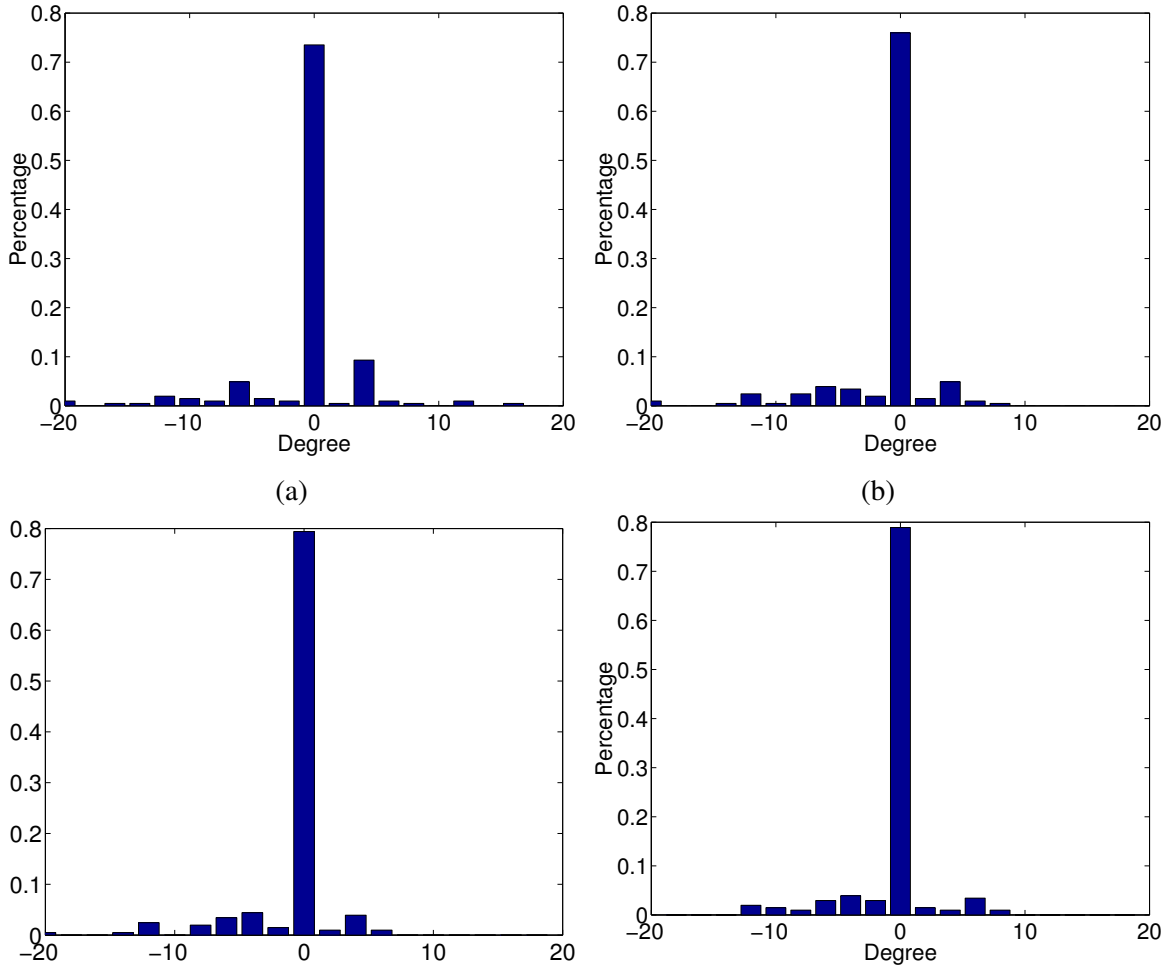
D. For data with manually extracted features

We have also further investigated how our IHDR algorithm performs on lower dimensional real data, such as those publicly available data sets that use human defined features. We tested our algorithm on two publicly available



(a) (b)

Fig. 13. Error Comparison among IHDR, FF and RBF for Vision-Based Navigation. (a) The plot for maximum error rates vs. epochs. (b) The plot for mean error rates vs. epochs. The solid line represents the error rates for resubstitution test. The dashed line represents the error rates for the testing set.



(a) (b) (c) (d)

Fig. 14. The histograms of the error rates. Plot (a), (b), (c), and (d) correspond to the histograms at epoch 1, 6, 11, 20, respectively.

TABLE VI
THE PERFORMANCE FOR VISION-BASED NAVIGATION

Algorithm	Mean error (degree)		Max. error (degree)	
	Resubstitution	Test set	Resubstitution	Test set
FF	1.02	2.00	10	12
RBF	1.53	1.84	12	12
IHDR tree	0.00	1.25	0	13

TABLE VII
TEST RESULTS ON LETTER IMAGE RECOGNITION DATA

Algorithm	Error rate		Time (sec)	
	Training	Testing	Training	Testing
Alloc80	0.065	0.064	39575	?
KNN	0	0.068	15	2135
** HDR tree	0	0.070	212.7	30
* IHDR tree	0	0.072	1150	41
LVQ	0.057	0.079	1487	48
QuaDisc	0.101	0.113	3736	1223
Cn2	0.021	0.115	40458	52
BayTree	0.015	0.124	276	7
NewId	0	0.128	1056	2
IndCart	0.010	0.130	1098	1020
C4.5	0.042	0.132	309	292
Dipol92	0.167	0.176	1303	80
Radial	0.220	0.233	?	?
LogDisc	0.234	0.234	5062	39
Ac2	0	0.245	2529	92
Castle	0.237	0.245	9455	2933
Kohonen	0.218	0.252	?	?
Cal5	0.158	0.253	1033	8
Smart	0.287	0.295	400919	184
Discrim	0.297	0.302	326	84
BackProp	0.323	0.327	277445	22
Bayes	0.516	0.529	75	18
Itrule	0.585	0.594	22325	69
Default	0.955	0.960	?	?
Cascade	1.0			
Cart	1.000			

data sets, and we report the comparison results for these data sets.

1) Letter image recognition data: There are 26 classes which correspond to 26 capital letters. Each sample has 16 numeric features. 15000 samples were used for training and 5000 samples were used for testing.
2) Satellite image dataset: There are six decision classes representing different types of soils from satellite images. Each sample has 36 attributes. The training set includes 4435 samples and the testing set includes 2000 samples.

We listed the results of our HDR and IHDR algorithms with those published in the StatLog project from [30] as Tables VII and VIII. For these lower dimensional data sets, the performance of our IHDR tree algorithm is comparable with other best existing ones.

IHDR and its batch version HDR have also been used in several applications as a component. HDR was used for vision-based motion detection, object recognition (appearance classification), and size dependent action (appearance regression) in [31]. IHDR was used for recognition of hand-written numerals and detection of orientation of natural images in [32]. This is the archival journal version of IHDR which explains IHDR in its entirety.

TABLE VIII
TEST RESULTS ON SATELLITE IMAGE DATASET

Algorithm	Error rate		Time (sec)	
	Training	Testing	Training	Testing
KNN	0.089	0.094	2105	944
LVQ	0.048	0.105	1273	44
** HDR tree	0	0.108	2.36	0.41
Dipol92	0.051	0.111	746	111
Radial	0.111	0.121	564	74
Alloc80	0.036	0.132	63840	28757
* IHDR tree	0	0.135	220	0.85
IndCart	0.023	0.138	2109	9
Cart	0.079	0.138	330	14
BackProp	0.112	0.139	72495	53
BayTree	0.020	0.147	248	10
NewId	0.067	0.150	226	53
Cn2	0.010	0.150	1664	36
C4.5	0.040	0.150	434	1
Cal5	0.125	0.151	764	7
QuaDisc	0.106	0.155	157	53
Ac2	?	0.157	8244	17403
Smart	0.123	0.159	27376	11
LogDisc	0.119	0.163	4414	41
Cascade	0.112	0.163	7180	1
Discrim	0.149	0.171	68	12
Kohonen	0.101	0.179	12627	129
Castle	0.186	0.194	75	80
Bayes	0.308	0.287	75	17
Default	0.758	0.769		
Itrule	?	100.00		

VI. CONCLUSIONS

IHDR is an approximation for fully automatic development of an associative cortex with bottom-up sensory pathways, top-down motor projections. Various automatic, adaptive plasticities occur in different cortical regions (layers), cortical patches (nodes) and neurons (clusters). The proposed IHDR technique is for the very challenging 7 simultaneous requirements: high-dimensional inputs, one-instance learning, adaptation to increasing complexity, avoidance of local minima, incremental learning, long-term memory, and logarithmic time complexity. The proposed IHDR technique is applicable to both regression and classification problems.

We propose to cluster in both output and input spaces. Clusters in the output space provide coarse-to-fine virtual class labels from clusters in the input space. Thus, discriminant analysis is possible. To deal with high-dimensional input space, in which some components are not very useful and some can be very noisy, a discriminating subspace is incrementally derived at each internal node of the tree. Such a discriminant subspace is especially necessary for high dimensional input space. A size-dependent probability-based distance metric SDNLL is proposed to deal with large sample cases, small sample cases, and unbalanced sample cases, which occur at different nodes at different levels with different observation richness.

Our experimental study with the synthetic data has showed that the method can achieve near-Bayesian optimality for both low-dimensional data and high-dimensional data with low-dimensional data manifolds. With the help of the decision tree, the retrieval time for each sample is of logarithmic complexity making real-time performance a reality even for high-dimensional inputs. The output of the system can be both class label or numerical vectors, depending on how the system trainer gives the training data. The experimental results have demonstrated that the algorithm can deal with a wide variety of sample sizes with a wide variety of dimension. The presented IHDR technique enables real-time, online, interactive training where the number of training samples is too large to be stored or to be processed in a batch, but the resulting IHDR tree does not need to store all of the training samples.

REFERENCES

[1] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, Eds., *Principles of Neural Science*, 4th ed. New York: McGraw-Hill, 2000.

[2] D. N. Pandya and B. Seltzer, "Association areas of the cerebral cortex," *Trends in Neurosciences*, vol. 5, pp. 386–390, 1982.

[3] B. Kolb and I. Q. Whishaw, *Fundamentals of Human Neuropsychology*, 3rd ed. New York: Freeman, 1990.

[4] J. Weng, W. S. Hwang, Y. Zhang, C. Yang, and R. Smith, "Developmental humanoids: Humanoids that develop skills automatically," in *Proc. First IEEE Conf. on Humanoid Robots*. MIT, Cambridge, Massachusetts: IEEE Press, Sept. 7-8 2000.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. New York: Chapman & Hall, 1993.

[6] J. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, California: Morgan Kaufmann, 1993.

[7] J. H. Friedman, "A recursive partition decision rule for nonparametric classification," *IEEE Trans. on Computers*, vol. 26, pp. 404–408, April 1977.

[8] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of Artificial Intelligence*, vol. 2, pp. 1–33, August 1994.

[9] D. L. Swets and J. Weng, "Hierarchical discriminant analysis for image retrieval," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 386–401, 1999.

[10] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[11] W. S. Hwang and J. Weng, "Hierarchical discriminant regression," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1277–1293, 2000.

[12] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, "Autonomous mental development by robots and animals," *Science*, vol. 291, no. 5504, pp. 599–600, 2001.

[13] J. Weng, "Developmental robotics: Theory and experiments," *International Journal of Humanoid Robotics*, vol. 1, no. 2, pp. 199–235, 2004.

[14] M. Kirby and L. Sirovich, "Application of the karhunen-loève procedure for the characterization of human faces," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, no. 1, pp. 103–108, Jan. 1990.

[15] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[16] H. Murase and S. K. Nayar, "Visual learning and recognition of 3-D objects from appearance," *International Journal of Computer Vision*, vol. 14, no. 1, pp. 5–24, January 1995.

[17] D. L. Swets and J. Weng, "Using discriminant eigenfeatures for image retrieval," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, pp. 831–836, 1996.

[18] J. Weng and S. Chen, "Incremental learning for vision-based navigation," in *Proc. Int'l Conf. Pattern Recognition*, vol. IV, Vienna, Austria, Aug. 25-30 1996, pp. 45–49.

[19] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-90-100, Feb. 1990.

[20] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. New York: Academic Press, 1990.

[21] J. Weng and S. Zeng, "A theory of developmental mental architecture and the dav architecture design," *International Journal of Humanoid Robotics*, vol. 2, no. 2, pp. 145–179, 2005.

[22] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore, Maryland: The Johns Hopkins University Press, 1989.

[23] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. New York: Cambridge University Press, 1986.

[24] H. Murase and S. K. Nayar, "Illumination planning for object recognition in structured environments," in *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, Seattle, Washington, June 1994, pp. 31–38.

[25] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[26] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[27] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schlkopf, and A. Smola, "Support vector machine reference manual," Royal Holloway, University of London, Egham, UK, Tech. Rep. CSD-TR-98-03, March. 1998.

[28] J. Quinlan, "Introduction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[29] P. J. Phillips, H. Moon, P. Rauss, and S. A. Rizvi, "The FERET evaluation methodology for face-recognition algorithms," in *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, San Juan, Puerto Rico, June 1997, pp. 137–143.

[30] D. Michie, D. Spiegelhalter, and C. T. (eds), *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[31] C. Yang and J. Weng, "Visual motion based behavior learning using hierarchical discriminant regression," *Pattern Recognition Letters*, vol. 23, no. 8, pp. 1031–1038, 2002.

[32] J. Weng and W. Hwang, "Online image classification using IHDR," *International Journal on Document Analysis and Recognition*, vol. 5, no. 2-3, pp. 118–125, 2002.