PERGAMON

Neural Networks 12 (1999) 1037–1051

Neural Networks

www.elsevier.com/locate/neunet

Unsupervised visual learning of three-dimensional objects using a modular network architecture

H. Ando[a,*], S. Suzuki[a,b], T. Fujita[a,c]

[a]*ATR Human Information Processing Research Laboratories, Soraku-gun, Kyoto 619-0288, Japan*
[b]*NTT Communication Science Laboratories, Soraku-gun, Kyoto 619-0237, Japan*
[c]*Sony Corporation Media Processing Laboratories, Shinagawa-ku, Tokyo 141-001, Japan*

Received 1 December 1998; accepted 28 May 1999

Abstract

This paper presents a modular network architecture that learns to cluster multiple views of multiple three-dimensional (3D) objects. The proposed network model is based on a mixture of non-linear autoencoders, which compete to encode multiple views of each 3D object. The main advantage of using a mixture of autoencoders is that it can capture multiple non-linear sub-spaces, rather than multiple centers for describing complex shapes of the view distributions. The unsupervised training algorithm is formulated within a maximum-likelihood estimation framework. The performance of the modular network model is evaluated through experiments using synthetic 3D wire-frame objects and gray-level images of real 3D objects. It is shown that the performance of the modular network model is superior to the performance of the conventional clustering algorithms, such as the *K*-means algorithm and the Gaussian mixture model.

Keywords: Three-dimensional object recognition; Unsupervised learning; Clustering; Modular networks; Mixture of autoencoders; Multiple views

1. Introduction

One of the fundamental issues in vision science is how the information of the three-dimensional (3D) environment is represented in the brain. The biological visual systems have a remarkable ability of recognizing the 3D world from constantly changing retinal images. In particular, although the projected images of a 3D object significantly change as the viewpoint or the pose of the object changes, the object can be readily identified. It is therefore a great challenge to elucidate computational mechanisms of how the information on 3D object is learned and represented in the neural systems.

Recent computational studies have explored a view-based approach to 3D object recognition, where 3D objects are described by the representations in the viewer-centered coordinate frame (e.g. Poggio & Edelman, 1990; Murase & Nayar, 1995; Moghaddam & Pentland, 1996). This approach contrasts with a conventional approach that constructs structural descriptions of objects using 3D volumetric primitives in the object-centered coordinate frame (e.g. Marr & Nishihara, 1978). In this structural description approach, however, the reliable construction of volumetric primitives is often very difficult and time-consuming. In the view-based approach, on the contrary, a limited number of view examples are learned and view invariant recognition achieved by interpolating the acquired representations.

The use of view-dependent representations of 3D objects in the brain was suggested by empirical experiments: psychophysical studies have shown that representations of 3D objects in the human visual system are viewpoint-specific (Bülthoff & Edelman, 1992; Edelman & Bülthoff, 1992). Electrophysiological experiments on behaving monkeys have also indicated that the primate inferotemporal (IT) cortex employs viewer-centered object representations (Logothetis & Pauls, 1995; Logothetis, Pauls & Poggio, 1995).

Although a number of research works have addressed the issue of the representation of a single object, how the images of a number of objects are learned and organized has not been studied very much. In particular, as the images of objects are in general not explicitly labeled as belonging to objects, the images should be able to be organized with no knowledge on their object identities. This paper, therefore, investigates an unsupervised learning model that automatically clusters multiple views into multiple object classes.

* Corresponding author. Tel.: + 81-774-95-1059; fax: + 81-774-95-1008.

E-mail address: ando@hip.atr.co.jp (H. Ando)

0893-6080/99/$ - see front matter
PII: S0893-6080(99)00052-0

Unsupervised data clustering can be regarded as one of the advanced inductive functions of the brain. Recent physiological data from single-unit recording experiments showed that cells in the anterior IT cortex of the primate brain selectively responds to moderately complex object features, and that cells responding to similar features cluster in a columnar region (Tanaka, 1993; Tanaka, Saito, Fukada & Moriya, 1991; Fujita, Tanaka, Ito & Cheng, 1992). An optical imaging study further revealed that in the primate IT, the activation spot gradually shifts as the viewing angle of a face changes (Wang, Tanaka & Tanifuji, 1996). These observations suggest that the representations of multiple object images may be self-organized into modular structures in the anterior IT.

Clustering multiple views into multiple object classes is generally a difficult computational task due to the following reasons: firstly, the dimensionality of object images is very high. Secondly, the distribution of the views of an object in the data space is often very complex, i.e. the object image significantly changes as the viewpoint changes. Thirdly, the view distributions of multiple objects are not easily separable, i.e. some views of different objects are more similar than the views of the same object. Despite these difficulties, there are two properties of the data structure that we can make good use of. First, although the input dimension is very high, the view data of an object often resides in a low-dimensional sub-space. Second, the view distribution of an object is inherently continuous, i.e. a chain of multiple views constitutes the continuous data manifold.

A clustering strategy that we explore in this paper is, therefore, to identify multiple non-linear sub-spaces each of which contains the views of each object class. The clustering algorithm proceeds by iterating the computation of the distances between each view data and the estimated sub-spaces, and the re-estimation of the sub-spaces using these distances. This iterative procedure is similar to the expectation-maximization (EM) clustering algorithm of the Gaussian mixture model (Dempster, Laird & Rubin, 1977). A critical difference, however, is that in the proposed model the distance of the data is defined not to a single prototypical view but to a non-linear sub-space which contains multiple view data. Once these sub-spaces are identified, the given view can be classified into an object class whose sub-space is closest to the data.

This multiple sub-space strategy can be achieved by using a modular network architecture that consists of a combination of multiple autoencoders. In this mixture of autoencoder architecture, each autoencoder network is considered as a module that discovers a non-linear sub-space of each object class. Here, an unsupervised learning algorithm for estimating unknown parameters of the model can be formulated in a statistical framework, i.e. the connection weights in the modular networks are iteratively estimated based on maximum likelihood estimation. Preliminary investigations on the modular network model have been discussed earlier (Suzuki and Ando, 1999, 1995; Ando, 1996; Fujita, Suzuki & Ando, 1996). However, this paper presents a more detailed description of the model with its relation to other conventional clustering models, and experimental evaluations on its performance using not only synthetic 3D objects but also real 3D objects.

The paper is organized as follows: Section 2 describes the details of the clustering models examined in this paper. This section first describes the *K*-means algorithm, and then shows how the algorithm is extended to the Gaussian mixture model and to the modular network model. Section 3 describes the clustering experiments using computer-generated images of 3D wire-frame objects and gray-level images of real 3D objects. The performance of the modular network model is compared with the performance of the *K*-means algorithm and the Gaussian mixture model. Section 4 finally concludes the paper.

2. Modular network model

This section describes a modular network model based on the mixture of autoencoders. We first describe the conventional clustering models, i.e. the *K*-means method and its extension, the Gaussian mixture model (Duda & Hart, 1973; Bishop, 1995). We then point out some limitations of these methods when applied to object view data and explain how the mixture of autoencoders may overcome these limitations. The main advantage of using a mixture of autoencoders is that the model assumes multiple sub-spaces for describing continuous data distributions whereas the *K*-means algorithm and the Gaussian mixture model assume multiple center points. We will explain the training methods based on the maximum likelihood estimation.

2.1. K-means algorithm

The *K*-means algorithm is a simple yet effective unsupervised clustering algorithm, hence it has widely been used for data clustering. It has also been used for determining initial center values for the subsequent supervised training of radial basis function networks (Moody & Darken, 1989). The *K*-means algorithm finds K vectors $\mu_i (i = 1, \ldots, K)$ given N data points $\{x_n\}$. The algorithm partitions the data points into K classes C_i which contain N_i data points. The number K must be given in advance. To find the partitions, we can minimize a sum-of-squares cost function of the form

$$E = \sum_i \sum_{n \in C_i} \|x_n - \mu_i\|^2, \tag{1}$$

where μ_i is the mean vector of the data points in class C_i, which is written as

$$\mu_i = \frac{1}{N_i} \sum_{n \in C_i} x_n. \tag{2}$$

The batch version of the *K*-means algorithm starts from arbitrary initial mean vectors and iterates between the

assignment of each data point to the class that has the nearest mean vector, and the computation of the mean vectors of the data points in each redefined class. It can be shown that the value of the cost function does not increase at each iteration (Linde, Buzo & Gray, 1980). The iteration proceeds until there is no further change in the assignment of the data points.

The online version of the *K*-means algorithm, on the contrary, updates the mean vectors as each data point is presented. The update rule is given by

$$\Delta\mu_i = \kappa(x_n - \mu_i) \tag{3}$$

where κ is the learning rate parameter.

2.2. Gaussian mixture models

The mixture models regard unsupervised data clustering as a mixture density estimation problem (Duda & Hart, 1973). The mixture density of the input data is given by a linear combination of component densities $p(x_n|i)$ of the form

$$p(x_n) = \sum_i p(x_n|i)\, P(i). \tag{4}$$

In the probabilistic framework, the mixing coefficients $P(i)$ can be regarded as the prior probability of the data being generated from component i, and the component densities $p(x_n|i)$ can be regarded as the conditional probability. Using Bayes' theorem, the posterior probability, which gives the probability that the data x_n belongs to a class i, is written as

$$P(i|x_n) = \frac{p(x_n|i)\, P(i)}{p(x_n)}. \tag{5}$$

In the case of the Gaussian mixture model, the component densities are modeled by Gaussian distribution functions of the form

$$p(x_n|i) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(x_n - \mu_i)^{\mathrm{T}}\Sigma_i^{-1}(x_n - \mu_i)\right\}, \tag{6}$$

where the mean μ_i is a d-dimensional vector, Σ_i is a $d \times d$ covariance matrix, and $|\Sigma_i|$ is the determinant of Σ_i. The distance defined by $D_i(x_n) = (x_n - \mu_i)^{\mathrm{T}}\Sigma_i^{-1}(x_n - \mu_i)$ is called the Mahalanobis distance. If we assume that the covariance matrix has the form of $\Sigma_i = \sigma_i^2 I$, where I is the identity matrix, the component density can be written as

$$p(x_n|i) = \frac{1}{(2\pi\sigma_i^2)^{d/2}} \exp\left\{-\frac{1}{2\sigma_i^2}\|x_n - \mu_i\|^2\right\}. \tag{7}$$

To find the unknown parameters $\{\mu_i, \sigma_i, P(i)\}$ for each class i, we can employ the maximum likelihood estimation method. The log-likelihood function is given by

$$\ln L = \ln\left\{\prod_n p(x_n)\right\} = \sum_n \ln p(x_n) = \sum_n \ln\left\{\sum_i p(x_n|i)\, P(i)\right\}. \tag{8}$$

We can maximize the log-likelihood function, or equivalently, minimize the energy function defined by the negative of log-likelihood,

$$E = -\ln L. \tag{9}$$

To maximize the log-likelihood function, standard nonlinear optimization algorithms can be used. The simplest method among these is the gradient descent algorithm, which is generally written in the continuous form as

$$\frac{\mathrm{d}\theta_i}{\mathrm{d}s} = -\kappa\frac{\partial E}{\partial\theta_i}, \tag{10}$$

where $\{\theta_i\}$ is the set of parameters to be estimated, s is the time in the gradient descent procedure, and κ is the learning rate parameter. In order to enhance the gradient descent performance, we can employ more efficient methods, such as the conjugate gradient algorithm or the quasi-Newton algorithm. In the case of maximizing (8), using (5) and (7), we can derive the following gradients, which are used for the gradient descent algorithm:

$$\frac{\partial E}{\partial\mu_i} = -\sum_n P(i|x_n)\frac{(x_n - \mu_i)}{\sigma_i^2}, \tag{11}$$

$$\frac{\partial E}{\partial\sigma_i} = \sum_n P(i|x_n)\left\{\frac{d}{\sigma_i} - \frac{\|x_n - \mu_i\|^2}{\sigma_i^3}\right\}. \tag{12}$$

For estimating the prior probability $P(i)$, the constraint $\sum_i P(i) = 1$ must be incorporated. We can use the Lagrange method, which yields

$$\frac{\partial E^*}{\partial P(i)} = N - \frac{\sum_n P(i|x_n)}{P(i)}, \tag{13}$$

where N is the number of data points, and E^* is a modified energy function where a term $\lambda(\sum_i P(i) - 1)$ is added to (9). Here, we set the Lagrange multiplier λ to N, which is derived by setting the derivative (13) to zero with the use of $\sum_i P(i) = \sum_i P(i|x_n) = 1$.

An alternative powerful optimization method for the maximum likelihood estimation is the EM algorithm (Dempster et al., 1977). The EM algorithm can be applied to the maximum likelihood problem when the data set is *incomplete*, i.e. the class labels are not given, or some variables are missing in the training data. The EM algorithm iterates between the E-step which computes the expectation of the complete data likelihood assuming that each data is completely labeled and the M-step, which maximizes this expectation. The E-step then reduces the computation of the posterior probability $P(i|x_n)$ given by (5). Using the estimated posterior probability, the M-step yields the following

update equations:

$$\mu_i^{s+1} = \frac{\sum_n P^s(i|x_n)x_n}{\sum_n P^s(i|x_n)}, \tag{14}$$

$$(\sigma_1^{s+1})^2 = \frac{\sum_n P^s(i|x_n)\|x_n - \mu_i^{s+1}\|^2}{d\sum_n P^s(i|x_n)}, \tag{15}$$

$$P(i)^{s+1} = \frac{1}{N}\sum_n P^s(i|x_n). \tag{16}$$

The *K*-means algorithm described in Section 2.1 can be regarded as a special case of the EM algorithm of the Gaussian mixture model. In the limit where σ_i goes to zero, it can be shown that the posterior probability becomes zero except for the probability whose center μ_i is closest to the data x_n, hence the EM Eq. (14) reduces to the *K*-means Eq. (2) (Bishop, 1995). Therefore, the Gaussian mixture model can be regarded as a probabilistic extension of the *K*-means algorithm, i.e. the Gaussian mixture model allows the soft partition of data points in proportion to the responsibility defined by the posterior probability which indicates the relative probability that the data was derived from each class.

Moghaddam and Pentland (1996), for example, presented a method that combines the principal component analysis with a Gaussian mixture model for object recognition and detection. As the dimensionality of object images is in general very high in the input space, the method first finds a principal sub-space, and then the Gaussian mixture model is applied to estimate the multimodal densities within the principal sub-space.

2.3. Mixtures of autoencoders

One of the limitations of using the *K*-means algorithm and the Gaussian mixture model is that the density estimation is not accurate if their underlying assumptions do not hold for the given distributions. In particular, the *K*-means algorithm and the Gaussian mixture model assume that the distribution of each class is unimodal, i.e. the distribution of each class is assumed to have a center in the data space. However, when the training data represents multiple views of a 3D object, the data is not generally distributed around a fixed center point but continuously distributed in the data space. Moreover, the manifold formed by each object should, in general, form a complex shape in the data space, and cannot be easily separated from the manifold of other objects; therefore, the within-class similarity is not always less than the between-class similarity. For example, the frontal view of a person's face can be more similar to the frontal view of another person's face than the profile view of the same person.

We can certainly approximate an arbitrary data distribution with multimodal densities, i.e. modeling a data distribution using a number of Gaussian densities. Nonetheless, if the goal of unsupervised clustering is to find meaningful classes, simply representing the data with multiple densities is not sufficient. Multiple views of *n* objects, for instance, could be described by *m* ($>n$) Gaussian densities each of which covers a subset of views of individual objects. However, in order to find *n* object classes, an additional clustering procedure is needed for combining *m* multiple densities. This additional clustering is generally not an easy task because the closest Gaussian centers do not necessarily belong to the same object class.

The basic idea of using a mixture of autoencoders for clustering data distributions is that it can capture multiple non-linear sub-spaces rather than multiple centers. In particular, when the data distributions form continuous and complex manifolds, which have no fixed centers, it is more effective to estimate the sub-spaces where the data distributions lie. During the training of a mixture of autoencoders, competition among the autoencoders allows the system to identify multiple non-linear sub-spaces. The following sections provide detailed descriptions of linear and non-linear autoencoders, their mixture models, and the maximum likelihood formulation for estimating unknown parameters in the models.

2.3.1. Linear and non-linear autoencoders for dimensionality reduction

It is often essential to reduce the dimensionality of high-dimensional data for extracting the intrinsic information in a low dimensional sub-space. Linear sub-space methods, such as principal component analysis (PCA), also known as Karhunen–Loéve transform (KLT), are widely used for dimensionality reduction. These methods, however, yield only approximate dimensions when the underlying statistical structure is non-linear in nature.

In the case of projected views of a 3D object, the number of parameters constraining the input distribution is intrinsically limited, despite the significant image variations as the viewpoint changes. In fact, any rigid object transformation can be described by six parameters or degrees of freedom, three for rotation and three for translation. As a mapping from the input views to the viewpoint parameters should be non-linear, a non-linear dimensionality reduction method is needed to extract such intrinsic dimensionality of the view distribution.

An autoencoder, which is also called an auto-associative network, can be used to perform linear and non-linear dimensionality reduction. An autoencoder consists of a multilayer perceptron which finds an identity mapping through a bottleneck in the hidden layer, i.e. the number of units in the hidden layer is set to be smaller than the number of input and output units. Hence, the network approximates functions F and F^{-1} such that $R^D \xrightarrow{F} R^M \xrightarrow{F^{-1}} R^D$, where $M < D$. The autoencoder network

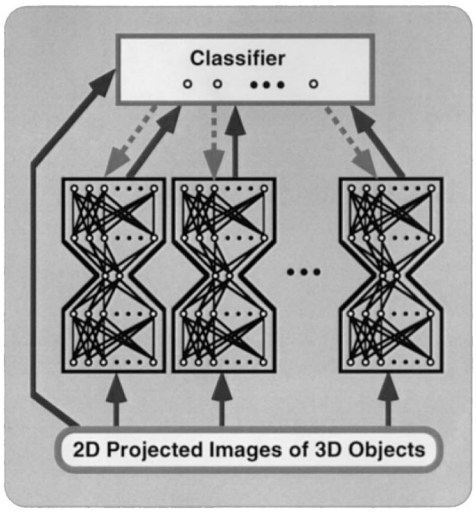

Fig. 1. A modular network architecture for unsupervised clustering of multiple views of multiple 3D objects. The modular network model consists of a set of five-layer autoencoders and a classifier.

compresses the input into a low-dimensional representation by eliminating redundancy in the input data distribution. The autoencoder network can be trained by minimizing a sum-of-squares error of the form

$$E = \sum_n \sum_k \{y_k(x_n) - x_{nk}\}^2, \tag{17}$$

where $x_n = \{x_{nk}\}$ is the nth data vector and y_k is the output of the kth unit of the network. Standard optimization methods, such as the gradient descent algorithm, can be used for estimating the network weights.

It has been shown that a three-layer autoencoder with a bottleneck in the second layer performs a linear dimensionality reduction specified by the principal components of the input distribution (Bourlard & Kamp, 1988; Baldi & Hornik, 1989; Oja, 1991). Specifically, if an autoencoder containing M hidden units is trained by minimizing the error function (17), the network projects the data onto the M-dimensional sub-space spanned by the first M principal components of the data. These M vectors, however, need not be orthogonal or normalized. For the new data, x^*, the error $\sum_k \{y_k(x^*) - x_k^*\}^2$ can be regarded as the distance between the data and the estimated sub-space.

If we use a five-layer autoencoder, on the contrary, with a bottleneck in the third layer, the network can perform a non-linear dimensionality reduction, which is a non-linear analogue to the principal component analysis (Oja, 1991; DeMers & Cottrell, 1993). In this case, the estimated sub-space can be non-planar. The error $\sum_k \{y_k(x^*) - x_k^*\}^2$ can again be regarded as the distance between the data x^* and the estimated sub-space. A five-layer autoencoder is, therefore, a simple yet powerful method for non-linear dimensionality reduction, although training the non-linear network may result in it becoming trapped in a sub-optimal local minimum.

A five-layer autoencoder has been applied to various tasks, such as time series data compression and face recognition (DeMers & Cottrell, 1993), color information encoding (Usui, Nakauchi & Nakano, 1992), and visual and motor information integration for grasping (Uno, Fukumura, Suzuki & Kawato, 1995). The non-linear dimensionality reduction is expected to overcome some limitations of linear principal component analysis. If, for example, the intrinsic dimension of D-dimensional data is two, and the data is not distributed on a planar surface, linear PCA cannot detect this two-dimensional (2D) sub-space but requires a sub-space with more dimensions for describing the data, whereas a non-linear method can, in principle, capture this 2D curved surface. Therefore, we expect that a five-layer autoencoder is more effective for estimating a non-linear mapping between the multiple views of 3D objects and their intrinsic viewpoint parameters.

2.3.2. Modular network architecture based on the mixture of autoencoders

When samples of multiple classes are distributed in a data space, the dimensionality reduction should be performed for the data of each class separately in order to extract the intrinsic information of each class. If we do not know which class each data belongs to, we should cluster the data into multiple classes while performing dimensionality reduction within each class at the same time. Therefore, the goal is to find multiple sub-spaces that capture the data distribution of each class. To achieve this, we could use the mixture of autoencoders, which is the combination of multiple autoencoders.

More specifically, the network architecture that we use for clustering the multiple views of multiple 3D objects consists of a set of modules, as illustrated in Fig. 1. Each module is a five-layer auto-associative network, which encodes and decodes multiple views of an object based on non-linear mappings. If each module can be trained to encode and decode multiple views of a single object, we can identify an input view as an object by selecting the module whose output view best fits the input view. We, therefore, define a classifier whose output vector is given by the softmax function of the negative squared difference between the input and the output of the module, i.e.

$$f_i = \frac{\exp[-\|x_n - y_i\|^2]}{\sum_j \exp[-\|x_n - y_i\|^2]}, \tag{18}$$

where x_n and y_i denote the input vector and output vector of the ith module, respectively. We regard each component f_i as the output of a unit in the classifier, which indicates a probability of the input view belonging to the corresponding class i. Therefore, if only one of the modules has an output that best matches the input, then the output value of the corresponding unit in the classifier becomes nearly one and the output values of the other units become nearly zero.

2.3.3. Unsupervised training based on maximum likelihood estimation

Algorithms for training the modular network architecture described in the previous section can be formulated based on a maximum likelihood estimation. In unsupervised training algorithms, the views provided to the networks are clustered without any knowledge of the object identities. As in the case of the Gaussian mixture model, we maximize the log-likelihood function,

$$\ln L = \sum_n \ln\{\sum_i p(x_n|i)\, P(i)\}. \tag{19}$$

While the component density $p(x_n|i)$ for the Gaussian mixture model is given by Eq. (6) or (7), for the mixture of autoencoders, the output of the ith autoencoder y_i should replace the mean vector μ_i. Therefore, assuming that the covariance matrix has the form $\Sigma_i = \sigma_i^2 I$, the component density function for the mixture of autoencoders is written as

$$p(x_n|i) = \frac{1}{(2\pi\sigma_i^2)^{d/2}} \exp\left\{-\frac{1}{2\sigma_i^2}\|x_n - y_i\|^2\right\}, \tag{20}$$

where x_n is the input data.

The gradient descent algorithm (10) for minimizing the energy function $E = -\ln L$ can be used to estimate the set of unknown parameters $\{w_i, \sigma_i, P(i)\}$, where w_i is the set of connection weights in each autoencoder network. By replacing the mean vector μ_i in Eqs. (11), (12) and (13) with the output vector of the ith autoencoder y_i, we obtain the following gradients for a batch formulation of the optimization

$$\frac{\partial E}{\partial y_i} = -\sum_n P(i|x_n)\frac{(x_n - y_i)}{\sigma_i^2}, \tag{21}$$

$$\frac{\partial E}{\partial \sigma_i} = \sum_n P(i|x_n)\left\{\frac{d}{\sigma_i} - \frac{\|x_n - y_i\|^2}{\sigma_i^3}\right\}, \tag{22}$$

$$\frac{\partial E}{\partial P(i)} = N - \frac{\sum_n P(i|x_n)}{P(i)}. \tag{23}$$

The posterior probability $P(i|x_n)$ is given by Eq. (5). The chain rule is applied to Eq. (21) to derive the derivative of the energy function with respect to weights w_i.

To obtain an online formulation, where the parameters are updated after the presentation of each data, we may simply drop the summation sign of n in the likelihood function (19), use (20) for the component density function, and approximate the prior probability in the likelihood function with the softmax function f_i described in Eq. (18). These operations yield a log-likelihood function of the form

$$\ln L = ln\left\{\frac{\sum_i \frac{1}{(2\pi\sigma_i^2)^{d/2}} \exp\left[-\left(1 + \frac{1}{2\sigma_i^2}\right)\|x_n - y_i\|^2\right]}{\sum_j \exp[-\|x_n - y_j\|^2]}\right\}. \tag{24}$$

This log-likelihood function can be maximized using the gradient ascent method. The obtained algorithm forces the output of at least one module to fit the input, and it also forces the rest of the modules to increase the error between the input and the output. As the constraint imposed by the bottleneck in the hidden layer prevents each module from encoding more than one object, we expect that the networks should eventually converge to a state where each module can identify an object.

The algorithms for training the mixture of autoencoders are related to a formulation of the adaptive mixture model (Jacobs & Jordan, 1993; Jacobs, Jordan, Nowlan & Hinton, 1991; Jordan & Jacobs, 1994). The adaptive mixture model can be regarded as a supervised version of the mixture of autoencoders model. In supervised learning problems, such as regression or classification, when the task can be divided into distinct subtasks, it is more useful and efficient to use multiple separate networks, each of which handles a subregion of the input space, than a single homogeneous network covering the whole input space. The adaptive mixture model is designed to learn how to allocate different 'expert' networks into different input regions. An appropriate decomposition of the input space is achieved by forcing the expert networks to compete in specializing the training data, and simultaneously training an extra 'gating' network that decides which of the expert networks should be used for each training data. By using the softmax activation function as the output of the gating network, the gating output can be regarded as a prior probability. As a result, a maximum likelihood estimation algorithm can be applied to the training of the expert and gating networks. While the adaptive mixture model uses a gating network for selecting a suitable network, the mixture of autoencoder's model does not require an additional network, as the classifier (18) can choose a module based on the reconstruction error of each module.

We should note that similar modular architectures using autoencoders for pattern recognition have been presented independently by several authors (Suzuki & Ando, 1995; Hinton, Revow & Dayan, 1995; Schwenk & Milgram, 1995). The model presented in this paper is significantly distinct from other models in the training methods, i.e. the proposed model is based on the unsupervised training of multiple autoencoders, whereas all other models train each module assuming that the identity of each pattern is provided during the training.

3. Experiments and discussions

The modular network model based on the mixture of autoencoders was implemented to evaluate its performance in unsupervised classification of images of 3D objects. In the experiments, multiple views of multiple 3D objects were randomly presented to the networks without providing their object labels, and the networks learned to cluster the given

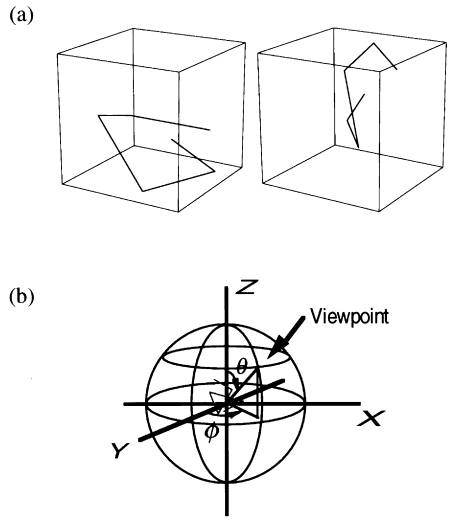

Fig. 2. (a) Examples of 3D wire-frame objects used for the experiments. The 3D objects were produced by connecting six vertices randomly generated in a unit cube. (b) A viewpoint which covers a sphere around the 3D objects. The position of the viewpoint is specified by two parameters (θ, ϕ).

views into multiple object classes. The difficulty of clustering multiple views of 3D objects lies in the fact that some views of an object are often more similar to the views of different objects than to other views of the same object. Therefore, in the following experiments, we selected multiple objects whose views were relatively difficult to discriminate, and compared the clustering abilities of the modular network model and conventional unsupervised models.

In Section 3.1, we describe the clustering experiments using computer-synthesized wire-frame objects. Here, we test how the modular network model clusters multiple views of multiple objects where the only available information is a set of positions of vertices or angles between segments in the projected images. In the experiments, we assume that a correspondence of these features is given among multiple views. We describe analyses of the encoded representations acquired in the hidden layer and the generalization properties of the networks. In Section 3.2, we describe the experiments using gray-level images of real 3D objects. Here, we test how the model clusters multiple views distributed in a high-dimensional input space. In the experiments, we assume that a correspondence among multiple views is unknown. We compare the performance of the mixture of autoencoders to the performance of conventional methods, such as the *K*-means algorithm and the Gaussian mixture model.

3.1. Wire-frame objects

This section examines the clustering ability of the modular network model using synthetic 3D wire-frame objects. Wire-frame objects have been used for computational studies on the supervised learning of 3D objects (Poggio & Edelman, 1990), psychophysical analyses of the human recognition system (Bülthoff & Edelman, 1992; Edelman & Bülthoff, 1992), and physiological investigations on object representations in the primate IT cortex (Logothetis & Pauls, 1995; Logothetis, Pauls & Poggio, 1995).

3.1.1. Three-dimensional objects and training procedures

The 3D objects that we used for the experiments were novel five-segment wire-frame objects whose six vertices were randomly selected in a unit cube $\{-1 < x, y, z < 1\}$, as shown in Fig. 2(a). Multiple views of the objects were obtained by orthographically projecting the objects onto an image plane whose position covered a sphere around the objects as shown in Fig. 2(b). The view position was defined by two parameters: θ and ϕ ($0 \leq \theta \leq \pi$, $0 \leq \phi < 2\pi$). A vector describing the view direction is given by ($\sin\theta \cos\phi$, $\sin\theta \sin\phi$, $\cos\theta$). In the following experiments, we used two types of features for the input—one was the x and y image coordinates of the six vertices which formed a twelve-dimensional vector, and the other was the cosines of the angles between the projected images of the adjoining segments which formed a four-dimensional vector.

The network model consisted of a set of modules where the number of modules was set equal to the number of objects used for the experiments. The number of units in the third layer of each module was set equal to the number of view parameters, which was two in the experiments. The number of units in the second and fourth layers was varied. In most of the experiments, five units were enough to achieve a reasonable clustering performance. To obtain a more accurate clustering performance, the number of units was increased up to twenty for these layers. During the training, objects were randomly selected among the object set and their views were randomly selected within the whole view range. Alternatively, we initially limited the ranges of θ and ϕ to $\pi/4$ and $\pi/2$, respectively, and gradually increased the ranges until they covered the whole sphere. These two types of training methods yielded similar results, but the latter method converged roughly ten times faster than the former method. We also conducted an experiment where the networks learned only a limited range of views in order to study how the networks generalize their clustering ability beyond the trained data set. In this experiment, the networks were trained within the ranges of $\pi/4$ and $\pi/2$ for θ and ϕ, respectively.

3.1.2. Results and analyses

The clustering performance of the networks was first examined using three wire-frame objects. To train the networks, we maximized the log-likelihood function described in Section 2.3.3. In this experiment, we used an on-line version of the training algorithms. The steepest ascent method was used to maximize the log-likelihood function. To improve the convergence rate, more efficient methods, such as the conjugate gradient method, can also be applied. We conducted experiments with two types of input

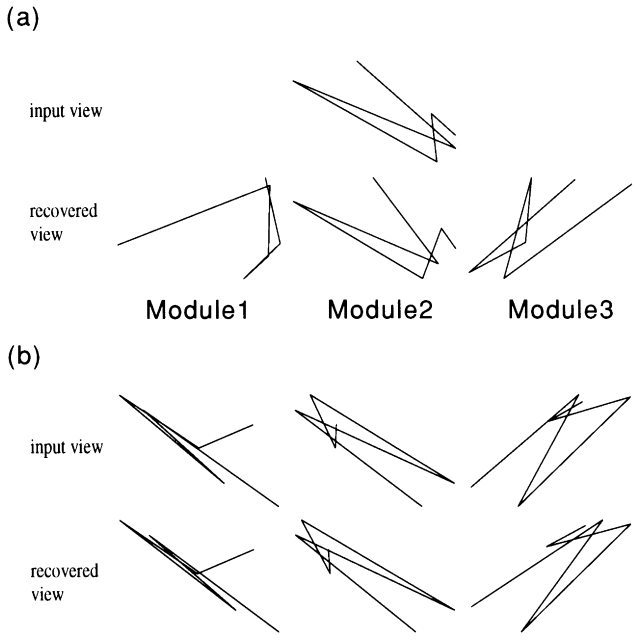

Fig. 3. (a) Recovered images of the three modules when a view of Object 3 was used for the input to the networks. The output image of Module 2 shows the most accurate recovery of the input view. (b) Examples of the input views of Object 3 and the views recovered by Module 2. The module can recover multiple views of the object.

features of the wire-frame objects: the x and y image coordinates of the vertices and the cosines of the angles formed by the adjoining segments in the image. Both types of features yielded a satisfactory classification performance, although the training using the angles required a longer learning period. In the following analyses, we only show the results that used the coordinates of vertices for the input to the networks.

Fig. 3(a) shows the output images of the three modules in the case where a view of Object 3 was used as the input to the trained networks. The output image of Module 2 exhibits the most accurate recovery of the input, hence this view can be classified into Class 2. Fig. 3(b) shows some other input views of Object 3 and the images recovered by Module 2. The figure shows that the module can recover multiple views of Object 3 although the shapes of these images are significantly different.

To determine the recovery ability of the networks in more detail, we provided the trained networks with 2500 views to cover the entire view range ($0 \leq \theta \leq \pi, 0 \leq \phi < 2\pi$) for each object. Fig. 4(a) shows the squared difference between the input views of the three objects and the recovered views of Module 2. As shown in the figure, the recovery error is nearly zero for all the views of Object 3 but the error is much larger for the views of the other two objects. The results, therefore, show that each module can recover the views of only one object, which is due to the dimensionality reduction in the hidden layer of each network. The ability of each network to recover the views of a single object can be used to classify multiple views of 3D objects. Fig. 4(b) shows the output of the classifier defined by Eq. (18) plotted over the view direction (θ, ϕ) when the inputs are all of the views of Object 3. The output value of Unit 2 in the classifier is nearly one over all the view directions, but the outputs of the two other units are nearly zero. When the views of other objects were used for the inputs, similar binary output patterns were obtained. Therefore, the results indicate that

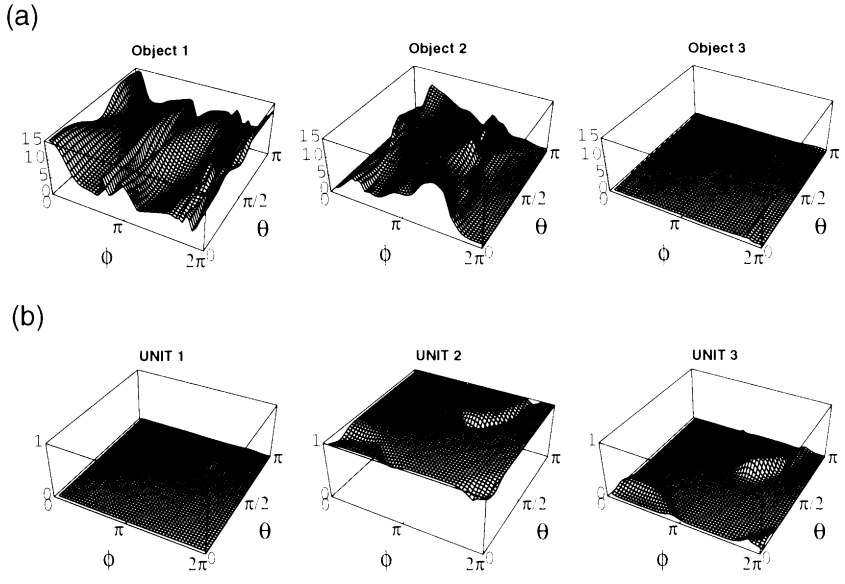

Fig. 4. (a) The squared difference between the input views and the views generated from Module 2 over an entire view range. The recovery error is nearly zero for the views of Object 3. (b) The output of a classifier when the views of Object 3 were used for the inputs. The output value of Unit 2 in the classifier is nearly one over the entire view range, while the outputs of the two other units are nearly zero.

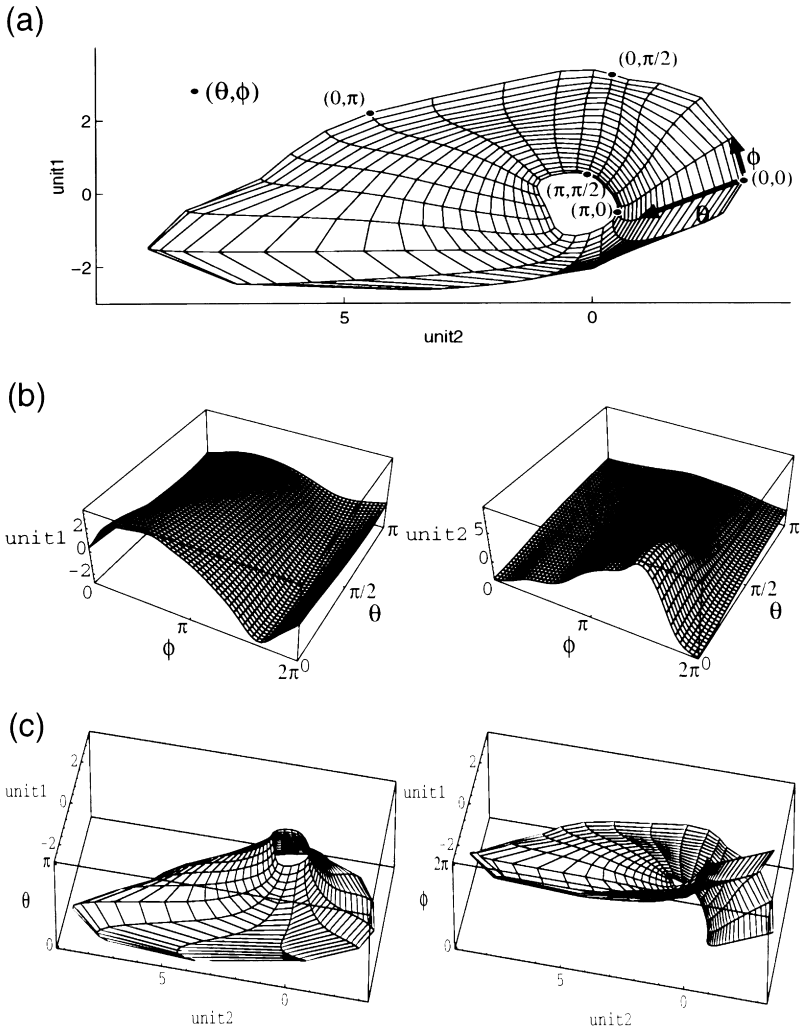

Fig. 5. Encoded representations in the third layer of a network. (a) The relationship between the outputs of the two hidden units and the view directions (θ, ϕ) of the input views; (b) the output values of the two hidden units as a function of the view directions; and (c) the view directions as a function of the outputs of the two hidden units.

the modular network model effectively clusters the multiple views of 3D objects without any object identity given to the networks during the training.

We analyzed the information acquired by the hidden units of the third layer. Fig. 5 shows the relationship between the encoded representations in the hidden layer of a network and the view directions (θ, ϕ) of the input images. Fig. 5(a) plots the outputs of the two units in the third layer of the network with labels of the view directions (θ, ϕ), when the images of the entire view range were provided to the networks. In this figure, the curves radiating in all directions from the center indicate contours at certain values of the periodic variable ϕ, and the circular curves indicate contours at certain values of the non-periodic variable θ. Therefore, each input view is mapped onto a point in the disk-like region of the hidden state. This figure shows that the information on the view direction, or equivalently, the pose of each 3D object is completely extracted and compactly described in the hidden layer of each module. To see this encoded information in more detail, the output values of the two hidden units are shown as a function of the view directions of the input views in Fig. 5(b). This graph is re-plotted conversely in Fig. 5(c) where the view direction (θ, ϕ) is shown as a function of the outputs of the hidden units. Fig. 5(b) and (c) show that these functions are single-valued, and this indicates that there are one-to-one correspondences between the view directions and the representations in the hidden layer. This viewpoint information encoded in the third layer enables each autoencoder network to reconstruct the multiple views of each 3D object in the fifth layer.

The generalization ability of the networks was investigated using the same set of data. Fig. 6 shows a clustering result obtained when the networks were trained within a limited view range, ($3\pi/8 \leq \theta \leq 5\pi/8$, $3\pi/4 \leq \phi \leq 5\pi/4$), which is indicated by the square in the figure. Fig. 6(a) shows the recovery error of one module when all the views of an object were provided to the trained networks. As

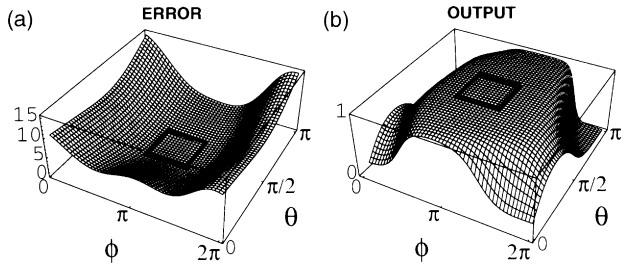

Fig. 6. A generalization ability of the networks. The networks were trained within a limited view range, which is indicated by the square in the figure. (a) The recovery error of one module; and (b) the output value of the corresponding unit of the classifier.

shown in the figure, the recovery error is almost zero within the trained region, and slowly increases as the view goes away from the trained region. Fig. 6(b) shows the output value of the corresponding unit of the classifier. The figure shows that the views were correctly classified not only within the trained region but also far beyond this region. We also conducted experiments to test how the model performs when the networks are trained with discretely sampled views. The result showed that even when the networks are trained with a set of 16 views evenly spaced in the entire view range, the model achieves a clustering rate of 90% for all the views. These results suggest that the modular network model has some generalization ability for clustering data when the training data is spatially limited or sparsely distributed in the data space.

Finally, the number of objects was varied and the performance of the modular network model compared with that of the *K*-means algorithm described in Section 2.1. We used sets of 3, 5, and 10 wire-frame objects, which were randomly generated in a unit cube. The average classification rate was computed for each condition by repeating the experiments 12 times using different sets of objects. As the experiments were based on unsupervised learning, there was

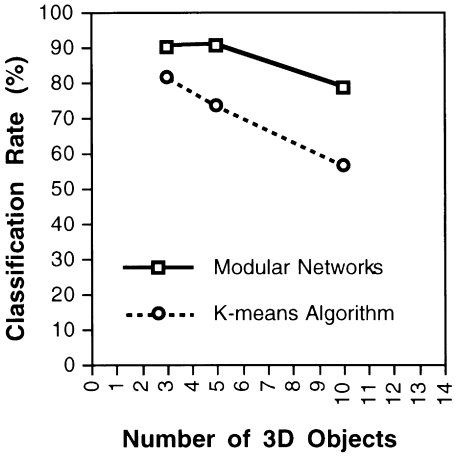

Fig. 7. The average classification rate when the number of objects were varied. The figure shows that the performance of the modular network model is superior to the performance of the *K*-means algorithm.

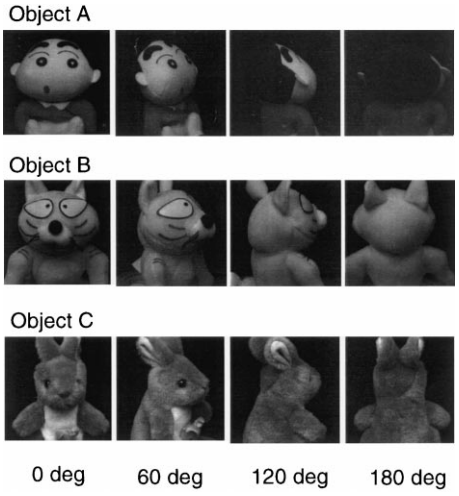

Fig. 8. Some examples of multiple views of 3D objects used for clustering experiments.

no fixed correspondence between the objects that we used and the classes that the networks estimated. Therefore, to compute the classification rate for each trial, we selected the best classification rate among the rates of all possible combinations of the objects and the classes. As shown in Fig. 7, the average classification rates were 90.2, 90.6, and 78.7% for 3, 5, and 10 objects, respectively.

The performance of the modular network model was compared with that of the *K*-means clustering algorithm. Using the same set of objects, the batch version of the *K*-means algorithm yielded average classification rates of 81.4, 73.3, and 56.6% for 3, 5, and 10 objects, respectively (see Fig. 7). The results, therefore, show that the performance of the modular network model is superior to the performance of the *K*-means algorithm.

3.2. Gray-level images of three-dimensional objects

The purpose of the experiments described in this section is to evaluate the ability of clustering gray-level images of real 3D objects, whose dimensions are much higher than those of the wire-frame objects used in the previous experiments. The performance of the mixture of autoencoders is compared to the performance of the *K*-means algorithm and the Gaussian mixture model.

3.2.1. Three-dimensional objects

The 3D objects that we used for the experiments included a doll and two stuffed animals. We chose these three objects because clustering the views of these objects is relatively difficult due to the similarity of their images. To obtain multiple views of the objects, we used a motorized turn table on which we placed each object. The objects were illuminated by two light sources located in front and an ambient dim light. The turn table was rotated around the vertical axis, and multiple views of the objects were taken by a fixed CCD camera at every one degree of rotation. We,

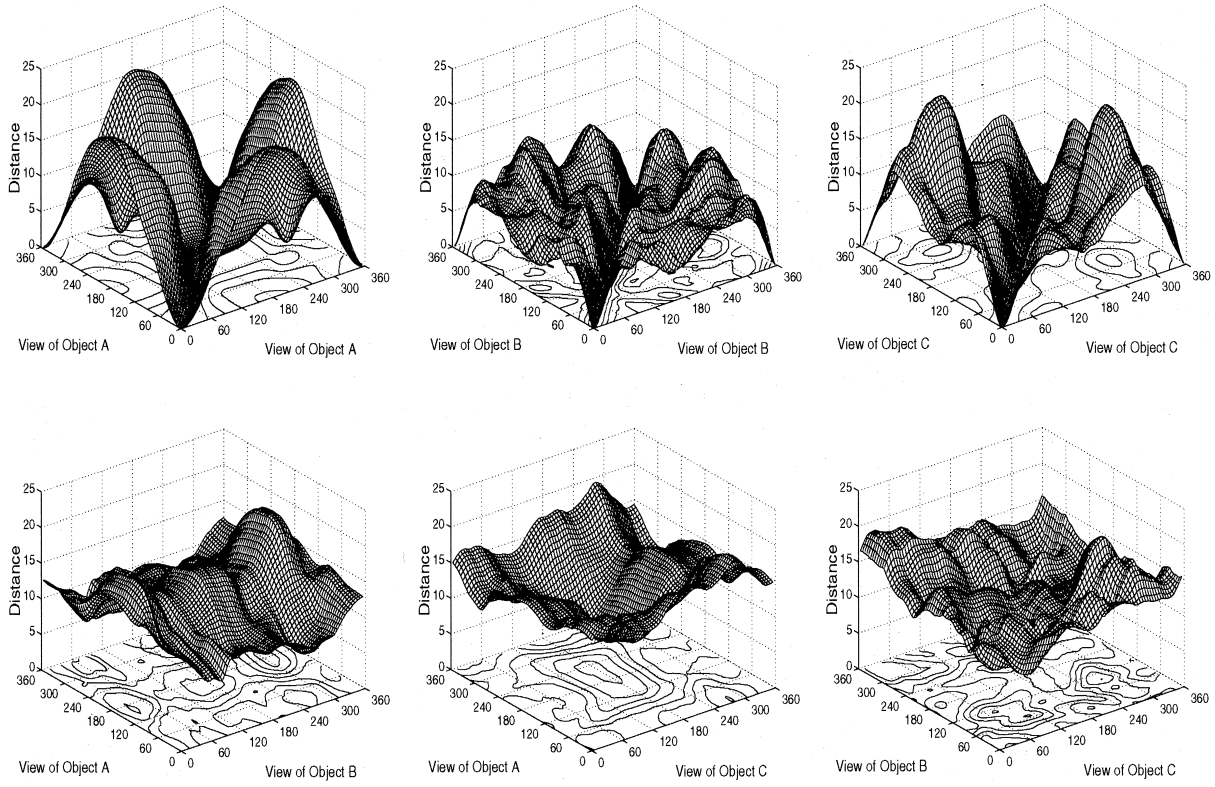

Fig. 9. Euclidean distance between every pair of object views. Contours of the constant distance are shown at the bottom of each figure.

therefore, obtained 360 images for each object. Each image had 512×480 pixels with 8-bit gray-scale values. Some examples of these images are shown in Fig. 8.

To reduce the noise and dimensionality of the images, the raw images were first blurred by a Gaussian filter whose standard deviation was 4.0 pixels and the gray-level values of the filtered images were then sampled by 16×15 grid points. The filtering and sampling yielded quantized images with 240 dimensions, which were used for the inputs to the networks. As each network contained sigmoidal functions whose outputs were bounded by (0,1), we divided the 8-bit gray-level values of the images by 100, which yielded values within the range (0.4,1.6).

To illustrate the similarity among these object images, we computed the Euclidean distance between every pair of images. Fig. 9 plots the distance over all the views, which is defined by the sum of the squared differences between the corresponding pixel values of the images. A key observation in this figure is that the distance of the data within each object class is not always smaller than the distance of the data between the object classes. For instance, the distance between the 0° view of Object A and its 180° view is much larger than the distance between the former view and all the views of Object B. Fig. 8 shows that the back head of Object A is much darker than its front and all the views of Object B shown in this figure. This observation suggests that an unsupervised method that simply assumes the center of each class will not easily cluster these images into object classes. We discuss this argument in more detail by comparing the performance of the mixture of autoencoders and conventional clustering algorithms.

3.2.2. Results and analyses

We used three five-layer autoencoders each of which contained 240 units for the first and fifth layers, five units in the second and fourth layers, and two units in the third layer. In general, the number of units in the third layer can be set to the degrees of freedom of the data distribution. As we used rigid objects rotated around a single axis, the intrinsic dimensionality was one. However, as the rotational parameter is periodic, its distribution cannot be described in a one-dimensional (1D) space. We therefore added one more dimension to the number of third-layer units and described a closed 1D manifold in a 2D encoded space.

For the networks used for training, a sigmoidal non-linear function was used for the units in the second and fourth layers, and a linear function was used for the units in the third and fifth layers. The autoencoders were trained by an online version of the learning method described in Section 2.3.3. We maximized the log-likelihood function using the steepest ascent method. For the initial weight values, a random number within the range of $(-0.5,0.5)$ was selected. At each training epoch, an image was randomly selected from all 1080 images.

After training the networks, all 360 views of each object were presented to the network in order to test the clustering

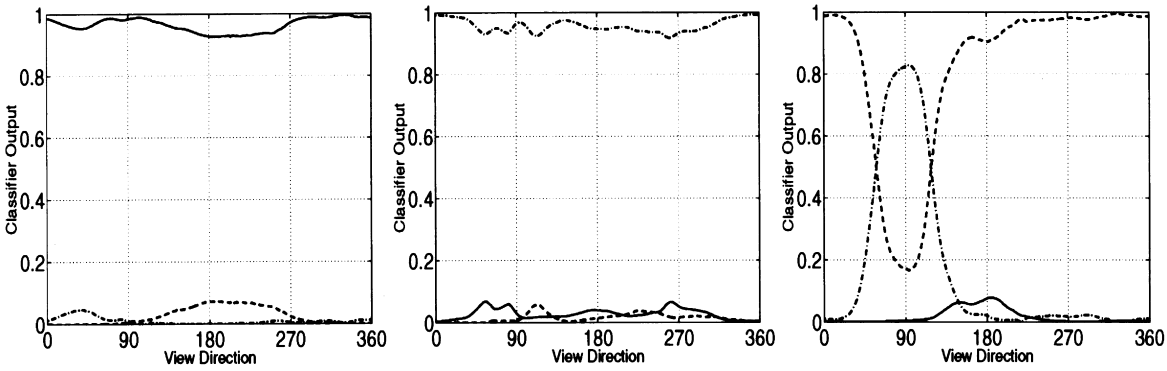

Fig. 10. A typical clustering result of the modular network architecture. Solid curves, dash-dotted curves, and broken curves indicate the output values of Class 1 unit, Class 2 unit, Class 3 unit in the classifier, respectively.

ability of the trained networks. The different views were encoded as a closed non-overlapping curve in the 2D hidden space of the third layer of each network. Fig. 10 shows a typical clustering result, which indicates the output of the classifier defined by Eq. (18) for all 360 views of each object. Fig. 10(a) shows that only the solid curve has a value near one, indicating that the views of Object A were clustered into Class 1. This is rather surprising because the front and the back views of Object A are all clustered into a single class, although the distance between these views is large as shown in Fig. 9. Fig. 10(b) shows that only the dash-dotted curve has a value near one, indicating that the views of Object B were all clustered into Class 2 as well. Fig. 10(c), on the contrary, shows that most of the views of Object C were clustered into Class 3 which is denoted by the broken curve, but for the views from 57° to 117°, the classifier yields a larger output value for the Class 2 unit than the Class 3 unit. The reason for this misclassification is that the images of Objects B and C are very similar for these view angles. In fact, as shown in the contours of the constant distance shown at the bottom of Fig. 9, the Euclidean distance between the images of Objects B and C takes the smallest values around this range of view angles. Even though such a misclassification exists, the overall classification rate reaches 94.40% for the result shown in this figure. The experiments were repeated 12 times with different initial weight values and we obtained the average classification rate of 92.26%. A perfect clustering occurred once out of the 12 trials.

We also tested the performance of the networks with three instead of five layers. In the case of a three-layer autoencoder, each network performs no better than a linear principal component analysis, as discussed in Section 2.3.1. The number of hidden units in the second layer was set to two, as in the case of the third layer of the five layer autoencoders. In this case, the classification rate averaged over 12 trials with different initial weight values was 80.81%. The results indicated that the mixture of the five-layer autoencoders specifying non-linear sub-spaces performs much better than the mixture of three-layer autoencoders specifying only linear sub-spaces. Increasing the number of hidden units from two to five for the three-layer network yielded an even poorer performance, i.e. the average classification rate was 62.43%. A linear space with a larger number of dimensions may have covered not only each class data, but the data of other classes.

We compared the performance of the mixture of autoencoders model to that of the *K*-means algorithm using the same set of data. We used the batch version of the *K*-means algorithm described in Section 2.1, assuming that the number of classes was three. As the objective function was non-quadratic and may have contained many local minima, the experiments were conducted with different initial values. To set the initial mean estimates, a number was randomly generated within the range (0.5,0.6) and was assigned to each of the 15×16 pixels. We regarded that the algorithm converged when the amount of data each class

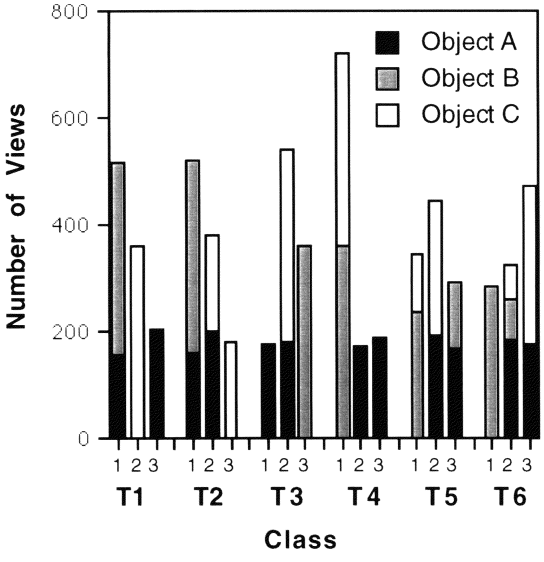

Fig. 11. Clustering results of the *K*-means algorithm. The converged patterns were categorized into six clustering types indicated by T*i* ($i = 1, 2, \ldots, 6$). The number of views that was clustered into each class for each type is shown.

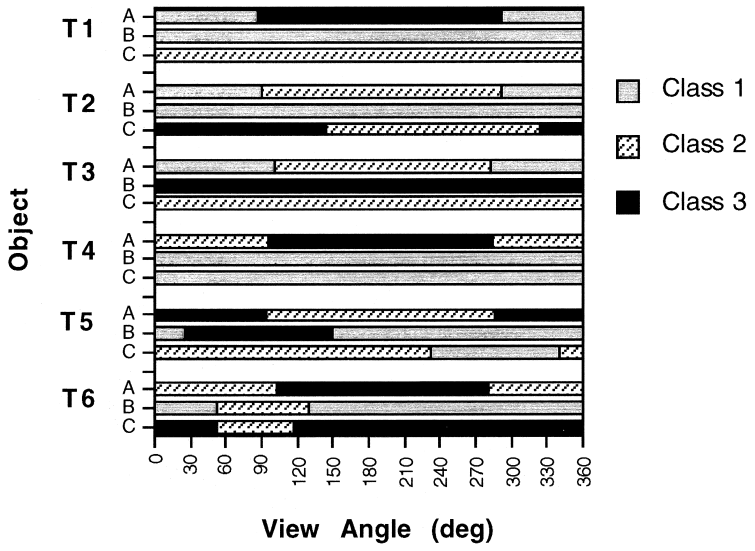

Fig. 12. Detailed description of the *K*-means clustering results. The figure indicates which class each view of each object was clustered into.

covered did not change any more during the iterative process.

We ran 33 trials with different initial values. Out of the 33 trials, three did not converge, i.e. the iterative process reached a state where at least one of the classes contained no data; therefore, its mean value could no longer be computed. The remaining 30 trials converged within 30 iterations. The classification rate averaged over the 30 trials was 75.50%. To compute the average classification rate, we selected the best classification rate among the six possible combinations of the three objects and the three classes. The lowest possible classification rate in this computation was therefore 33.3%.

The converged patterns were categorized into six clustering types indicated by Ti ($i = 1, 2, \ldots, 6$). The number of trials categorized into Ti ($i = 1, 2, \ldots, 6$) were 12, 11, 3, 2, 1, 1, respectively. Fig. 11 shows the number of views that were clustered into each class for each type. The views of Object A were split into two classes in all six types. The clustering of the views of Objects B and C varied, i.e. the views were successfully clustered into two classes in T1 and T3, but split into different classes in T2, T5, and T6, and all were clustered into one class in T4. The algorithm never reached a perfect state where each class contained only the views of a single object. When we ran the algorithm from an initial state where each mean value was set to the average of all the views of each object, the means gradually changed their values. We, therefore, confirmed that a perfectly clustered state is not a stable state of the *K*-means algorithm.

How the *K*-means algorithm clustered the views more or less reflected the similarity between the views. Fig. 12 shows the clustering results indicating which class each view of each object was clustered into. For the views of Object A, the front and back views were separated into two classes. This is a reasonable result when we consider the large Euclidean distance between these views as shown in Fig. 9. In T6, the views of Objects B and C around 90° were clustered together into Class 2, similar to the result using the mixture of autoencoders shown in Fig. 10, but Class 2 of the *K*-means algorithm also contained some views of Object A.

Experiments using the Gaussian mixture model were also conducted. Estimating a $d \times d$ full covariance matrix for each class was found to be computationally expensive, where $d = 240$ for the view data that we used. A covariance matrix was, therefore, assumed to be diagonal and to reduce the number of covariance components to be estimated to d. The EM algorithm described in Section 2.2 was used for estimating the parameters. However, we had difficulties getting the EM algorithm to converge. Twenty seven out of 30 trials did not converge, i.e. one of the covariance components became zero. Increasing the number of parameters may have also increased the number of inappropriate local minima. Therefore, we conducted experiments that assumed a constant variance for each Gaussian function.

Table 1
Clustering results on gray-level images of 3D objects

Conventional algorithms		Model network architectures	
K-means	Gaussian mixture	Three-layer autoencoders	Five-layer autoencoders
75.50%	75.37%	80.81%	92.26%

In this case, the EM algorithm converged in most trials. To compute the classification rate, the class that had the largest posterior probability was considered as the class that the view was classified into. The classification rate averaged over 30 trials with different initial values was 75.37%. Therefore, the Gaussian mixture model did not produce an improvement over the results of the *K*-means algorithm.

The main experimental results using the images of real 3D objects are summarized in Table 1. The performance of the mixture of autoencoders model was much better than the performance of the *K*-means algorithm and the Gaussian mixture model. The poorer performance of the conventional models can be ascribed to the inappropriate assumptions they imposed. These models assumed that each class has a center in the data space, but the views of each object are not distributed around a single center but rather forms a continuous manifold which has a complex shape in the data space. The better performance of the mixture of autoencoders, on the contrary, can be ascribed to its ability to find a sub-space that contains the data manifold of each object. In particular, the use of five-layer autoencoders outperformed the use of three-layer autoencoders, because they found non-linear sub-spaces which can more appropriately specify a complex shape of each view distribution.

4. Conclusions

To conclude, we have presented a modular network architecture that learns to cluster multiple views of multiple 3D objects without any identification of the object classes. An advantage of using the modular network architecture is that competition among the autoencoders allows the system to identify multiple non-linear sub-spaces. Consequently, the proposed model is more suited toward describing the complex distributions of the view data than the conventional clustering methods, which assume a center point for each data distribution. The superiority of the proposed model to the conventional clustering algorithms when applied to the view clustering problem was confirmed through experiments using both images of synthetic 3D wire-frame objects and gray-level images of real 3D objects.

Although this paper shows the usefulness of the proposed modular architecture, there are more issues to investigate. Firstly, in the current formulation, the number of objects is assumed to be given, but in more natural conditions, the number of objects is unknown. In this more general case, modules may be added or deleted during the training based on the reconstruction error of each module. Such a split–merge technique is worth exploring for further elaboration of the model. Secondly, we did not optimize the training time in the experiments described in this paper. In order to speed up the convergence, more efficient algorithms for training the modules need to be studied. Thirdly, the generalization ability of the networks should be investigated. We are currently modeling how to use the generalization ability of the trained networks to analyze a complex cluttered scene. Finally, it is a great challenge for us to unveil how biological neural systems acquire and self-organize 3D object representations. The proposed modular network architecture may provide useful directions towards understanding flexible visual learning in the brain.

References

Ando, H. (1996). 3D object recognition using bidirectional modular networks. In S. Z. Li (Ed.), *Recent developments in computer vision (ACCV'95)*, (pp. 467–475). New York: Springer.

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, *2* (1), 53–58.

Bishop, C. M. (1995). *Neural networks for pattern recognition*, Oxford: Oxford University Press.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, *59*, 291–294.

Bülthoff, H. H., & Edelman, S. (1992). Psychophysical support for a two-dimensional view interpolation theory of object recognition. *Proceedings of the National Academy of Science USA*, *89*, 60–64.

DeMers, D., & Cottrell, G. (1993). Non-linear dimensionality reduction. In S. J. Hanson & J. D. Cowan & C. L. Giles (Eds.), (pp. 580–587). *Advances in neural information processing systems*, 5. San Mateo, CA: Morgan Kaufmann.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *B39* (1), 1–38.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*, New York: Wiley.

Edelman, S., & Bülthoff, H. H. (1992). Orientation dependence in the recognition of familiar and novel views of 3D objects. *Vision Research*, *32*, 2385–2400.

Fujita, T., Suzuki, S., & Ando, H. (1996). 3D object recognition by coupling mixtures of autoencoders and dynamic matching. *Proceedings of the International Conference on Neural Information Processing (Hong Kong)*, 377–382.

Fujita, I., Tanaka, K., Ito, M., & Cheng, K. (1992). Columns for visual features of objects in monkey inferotemporal cortex. *Nature*, *360*, 343–346.

Hinton, G. E., Revow, M., & Dayan, P. (1995). Recognizing handwritten digits using mixtures of linear models. In G. Tesauro & D. Touretzky & T. Leen (Eds.), (pp. 1015–1022). *Advances in neural information processing systems*, 7. Cambridge, MA: MIT Press.

Jacobs, R. A., & Jordan, M. I. (1993). Learning piecewise control strategies in a modular neural network architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, *23* (2), 337–345.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3* (1), 79–87.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6* (2), 181–214.

Linde, Y., Buzo, A., & Gray, R. M. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications*, *28* (1), 84–95.

Logothetis, N. K., & Pauls, J. (1995). Psychophysical and physiological evidence for viewer-centered object representations in the primate. *Cerebral Cortex*, *3*, 270–288.

Logothetis, N. K., Pauls, J., & Poggio, T. (1995). *Spatial reference frames for object recognition tuning for rotations in depth*, Cambridge, MA: MIT A.I. Memo No. 1533, C.B.C.L. Paper No. 120.

Marr, D., & Nishihara, H. K. (1978). Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London B*, *200*, 269–294.

Moghaddam, B., & Pentland, A. (1996). Probabilistic visual learning for object representation. In S. K. Nayar & T. Poggio (Eds.), *Early visual learning*, (pp. 99–130). Oxford: Oxford University Press.

Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation, 1* (2), 281–294.

Murase, H., & Nayar, S. K. (1995). Visual learning and recognition of 3D objects from appearance. *International Journal of Computer Vision, 14* (1), 5–24.

Oja, E. (1991). Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen & K. Mäkisara & O. Simula & J. Kangas (Eds.), *Artificial Neural Networks*, (pp. 737–745). North Holland: Elsevier.

Poggio, T., & Edelman, S. (1990). A network that learns to recognize three-dimensional objects. *Nature, 343*, 263–266.

Schwenk, H., & Milgram, M. (1995). Transformation invariant autoassociation with application to handwritten character recognition. In G. Tesauro & D. Touretzky & T. Leen (Eds.), (pp. 991–998). *Advances in neural information processing systems*, 7. Cambridge, MA: MIT Press.

Suzuki, S., & Ando, H. (1999). A modular network scheme for unsupervised 3D object recognition. *Neurocomputing* (in press).

Suzuki, S., & Ando, H. (1995). Unsupervised classification of 3D objects from 2D views. In G. Tesauro & D. Touretzky & T. Leen (Eds.), (pp. 949–956). *Advances in neural information processing systems*, 7. Cambridge, MA: MIT Press.

Tanaka, K. (1993). Neuronal mechanisms of object recognition. *Science, 262*, 685–688.

Tanaka, K., Saito, H., Fukada, Y., & Moriya, M. (1991). Coding visual images of objects in the inferotemporal cortex of the macaque monkey. *Journal of Neurophysiology, 66* (1), 170–189.

Uno, Y., Fukumura, N., Suzuki, R., & Kawato, M. (1995). A computational model for recognizing objects and planning hand shapes in grasping movements. *Neural Networks, 8* (6), 839–851.

Usui, S., Nakauchi, S., & Nakano, M. (1992). Reconstruction of Munsell color space by a five-layer neural network. *Journal of Optical Society of America A, 9* (4), 516–520.

Wang, G., Tanaka, K., & Tanifuji, M. (1996). Optical imaging of functional organization in the monkey inferotemporal cortex. *Science, 272*, 1665–1668.