A Multitask Learning Model for Online Pattern Recognition

Seiichi Ozawa, *Member, IEEE*, Asim Roy, *Senior Member, IEEE*, and Dmitri Roussinov

***Abstract*—This paper presents a new learning algorithm for multitask pattern recognition (MTPR) problems. We consider learning multiple multiclass classification tasks online where no information is ever provided about the task category of a training example. The algorithm thus needs an automated task recognition capability to properly learn the different classification tasks. The learning mode is "online" where training examples for different tasks are mixed in a random fashion and given sequentially one after another. We assume that the classification tasks are related to each other and that both the tasks and their training examples appear in random during "online training." Thus, the learning algorithm has to continually switch from learning one task to another whenever the training examples change to a different task. This also implies that the learning algorithm has to detect task changes automatically and utilize knowledge of previous tasks for learning new tasks fast. The performance of the algorithm is evaluated for ten MTPR problems using five University of California at Irvine (UCI) data sets. The experiments verify that the proposed algorithm can indeed acquire and accumulate task knowledge and that the transfer of knowledge from tasks already learned enhances the speed of knowledge acquisition on new tasks and the final classification accuracy. In addition, the task categorization accuracy is greatly improved for all MTPR problems by introducing the reorganization process even if the presentation order of class training examples is fairly biased.**

***Index Terms*—Automated task recognition, knowledge transfer, multitask learning, online learning, pattern recognition.**

I. Introduction

In machine learning and neural networks, we are obviously interested in understanding and replicating the knowledge acquisition processes of humans. A particular area of interest is learning of related tasks. We generally observe that humans can learn a new task quite quickly when the task is similar to the one they have learned before. For example, a person who plays tennis can quickly and easily learn a similar sport such as squash or racquetball. So the theoretical question is: In what ways is the human brain using knowledge of one task to enhance its learning of another similar task? The conjecture is that the human brain is using some form of knowledge transfer when it knows that a new task is similar to the one it had learned before. Thus, we are interested in using similar knowledge transfer in artificial systems to speed up the learning of similar tasks.

In general, the question of knowledge transfer arises when multiple tasks are learned, one after another or one in conjunction with another. This type of simultaneous or sequential learning of different tasks is called multitask learning and is well described in the literature (e.g., [1], [6], [25], and [26]). Caruana presents a multitask learning mechanism where a single multilayered perceptron is trained to perform multiple tasks [5], [6]. Ghosn and Bengio [9] have proposed an approach based on manifold learning. In these approaches, all tasks are learned simultaneously by the system. On the other hand, Silver *et al.* [22], [23] have proposed a type of multitask learning where the tasks are given sequentially. The difficulty with this approach, where a new task is learned in the same neural network that had learned another task before, is the problem of *catastrophic interference* or *forgetting*[4]. In catastrophic interference, the neural network forgets the knowledge of previous tasks when a new task is learned because its structural parameters change with the learning of the new task. To prevent catastrophic forgetting, Silver and Mercer [23], [24] save the networks for previously learned tasks and use them to generate virtual examples of the old tasks to learn the main network that includes the new task.

We consider a dynamic multitask learning environment where training examples for different tasks are given sequentially to the learning system, but we permit frequent switching of tasks. In addition, no particular order is required for the sequence of training examples for a given task or for the random switching of tasks. This continuous online task learning is very similar to human learning. In real life, humans are often provided with different descriptions of the same object and they learn to describe those objects with lots of descriptors. For example, a child initially learns to recognize people and faces. Then, over time, a child learns other descriptions of people – their sex (male or female), their approximate age (young or old), their approximate size (big or small, fat or skinny), their approximate height (short or tall), and so on. Learning each type of description is a separate task for the child. A child is generally taught these tasks in a trial-and-error process over time using a variety of examples.

The need for this type of learning in artificial systems is illustrated by the following example. Suppose a robot is to be trained to recognize different features on the face of a person, such as color of hair, hair style, shape of the nose, ears, eyes, and so

Manuscript received August 20, 2007; revised June 28, 2008; accepted September 12, 2008. First published February 02, 2009; current version published February 27, 2009. This work was supported in part by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C) 18500174 and 20500205.

S. Ozawa is with the Graduate School of Engineering, Kobe University, Kobe 657-8501, Japan (e-mail: ozawasei@kobe-u.ac.jp).

A. Roy is with the Department of Information Systems, Arizona State University, Tempe, AZ 85048 USA (e-mail: asim.roy@asu.edu).

D. Roussinov is with the Department of Computer and Information Sciences, the University of Strathclyde, Glasgow, Scotland G1 1XQ, U.K. (e-mail: dmitri.roussinov@cis.strath.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2008.2007961

on. Then, the robot is to be trained with pictures of different people with different features where each picture is labeled accordingly. For example, one picture might have the label "black hair," another might say "red hair," and so on. A sequence of pictures showing different hair colors might be shown to the robot so that it can learn to recognize hair color. That could be followed by a sequence of pictures showing different shapes of faces. Then, it could be another sequence of pictures of different hairstyles and so on.

More generally, the multitask recognition problem is as follows. An artificial robot is presented with various objects and their descriptions at different times and the task is to learn to describe those objects with an appropriate set of descriptors. So the robot has the following tasks to perform: 1) automatically group the various descriptors into appropriate tasks without external supervision (e.g., group different hair color descriptions for the hair color recognition task), 2) build and train models appropriate to each task, and 3) then apply those task models to describe new objects.

In this paper, we propose an approach to multitask pattern recognition (MTPR) where learning is online and the system learns to recognize tasks automatically. *Online automated task recognition* capability is important to build robots that collect data and learn on their own. This paper is organized as follows. Section II presents general concepts related to online learning of multiple tasks and the basic ideas of an automated task recognition system. Then, Section III presents the online learning algorithm composed of multiple neural classifiers. In Section V, we present experimental results for ten MTPR problems based on five University of California at Irvine (UCI) data sets. Finally, Section VI has our conclusions and directions for future work.

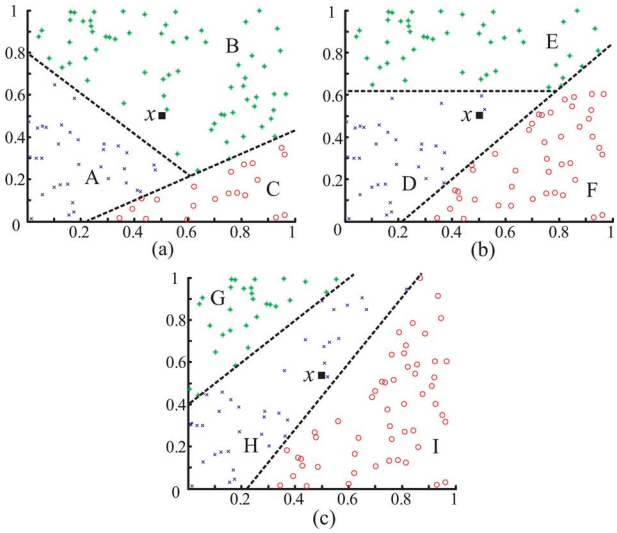


Fig. 1. Example of multitask learning problems in pattern recognition that is composed of three different three-class tasks. The points marked {o, x, *} represent the training examples in the 2-D input space; {o, x, *} are the class labels. This example is "Problem 2" in the later experiments. (a) Task 1. (b) Task 2. (c) Task 3.

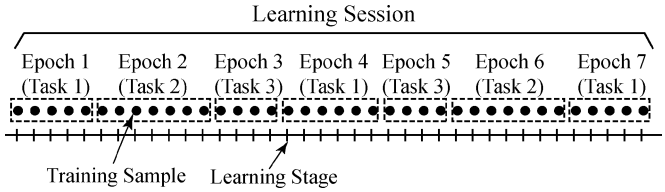


Fig. 2. Schematic diagram to show the sequential presentation of training examples.

II. The Multitask Learning Problem —Some General Concepts

A. The Multitask Learning Problem and Automated Online Learning of Tasks

MTPR problems [7], [15] consist of several learning tasks, each of which corresponds to a conventional multiclass classification problem. Fig. 1 shows an example of an MTPR problem in which three different classification tasks (tasks 1, 2, and 3) are defined. All classification tasks have the same input domain, but their class boundaries differ. Task 1 has three classes A, B, and C; task 2 has three classes D, E, and F; and task 3 consists of the classes G, H, and I. We assume that the tasks use different names for the classes. Thus, the class names A, B, and C are not used in tasks 2 and 3. But it is not known to the learning system that classes A, B, and C belong to task 1. During training, each training example comes with a class label, but not a task label. So the learning system does not know what task a particular training example belongs to.

We assume that the presentation of training examples to the system is sequential. The period of learning from a whole sequence of training examples is called a *learning session*. Fig. 2 shows an example of a learning session where the first five examples are from task 1 in Fig. 1, the second seven are from task 2, the next four are from task 3, followed by six from task 1, and so on. The period of learning from training examples of a single task is called an *epoch*. In Fig. 2, the first epoch consists of five examples of task 1 and the learning session consists of seven epochs for three different tasks. Thus, a training session is a sequence of epochs of different tasks where the same task can appear in different epochs in a session.

As mentioned above, the learning system is never told whether a particular training example is from task 1, 2, or 3. We require that after completion of a learning session, the learning system should say, for this problem, that *three tasks were learned* and that task 1 consists of classes A, B, and C, task 2 consists of classes D, E, and F, and task 3 consists of classes G, H, and I. So, in general, the learning system groups classes into tasks and makes appropriate inferences as to which classes belong to which tasks. Therefore, part of the proposed method performs unsupervised learning of tasks (assignment of classes to tasks and determining how many tasks there are) since training examples do not come with task labels.

In multitask problems, the question of knowledge transfer arises when the tasks are related to each other. In pattern classification tasks, task relatedness is measured by the amount of overlap between the classes of different tasks. For example, Fig. 1(a)–(c) shows that the region of class A has large overlaps with those of classes D and H, and class C also has large overlaps with classes F and I. So our proposed system is based on the following assumptions and requirements: 1) tasks are multiclass classification problems whose inputs are from the same

domain, but where class labels are unique to each task; 2) no task labels are provided in the training examples, just class labels; 3) training examples for a task are given sequentially one at a time and task switching can be random during online training; in other words, there is no particular order for the presentation of training examples or for the tasks; 4) classification tasks are related to one another; that is, there is some overlap between the class regions of different tasks; and 5) the learning system does automatic grouping of classes into tasks and builds classifiers for the tasks.

B. Stable Online Learning, Automated Task Recognition, and Knowledge Transfer

1) Stable Online Learning of Tasks Using Multiple Classifiers: To allow online learning of different tasks in a stable way, we use the classifier architecture called resource allocating network with long-term memory (RAN-LTM) [11], [12] that can build different classifiers for different tasks in an adaptive way. RAN-LTM is an extension of the RAN model [16] and the learning is carried out not only online but also in *one pass*[10] where training examples are presented to the learning system only once. RAN-LTM stores representative training examples in the long-term memory that are used for preventing catastrophic forgetting and for facilitating transfer of knowledge from one task to another. The RAN-LTM method is discussed in more detail in Section III-A.

2) Automated Task Recognition Mechanisms–A General Overview: The proposed algorithm groups classes into tasks to minimize prediction error. Since classes of different tasks have overlap, the grouping of classes into separate tasks is based on separating those classes that have substantial overlap and combining those that do not. For example, in Fig. 1, classes A, B, and C do not have substantial overlap between them, so they can be grouped into one task. Similarly, classes D, E, and F do not have substantial overlap, so they can be grouped into another task and so on. However, classes G, H, and I have substantial overlap with the classes A, B, C, D, E, and F, so all of those classes cannot be grouped together into a single task. The multitask learning system therefore has to recognize which classes do or do not overlap in a substantial way in order to assign classes to tasks and minimize the overall prediction error. At the end of training, the task recognition algorithm builds a separate classifier (a separate RAN-LTM neural network) for each task.

The proposed task recognition approach has some limitations as shown by the following example. The example in Fig. 3 has three tasks, but the tasks have classes that do not overlap. Since there is no overlap between the classes of different tasks, the algorithm will not recognize that there are three different tasks, although it will learn to perform all three tasks correctly using a single classifier.

3) Knowledge Transfer Mechanisms–A General Overview: The task recognition algorithm accumulates knowledge about tasks over time and the accumulated knowledge is transferred to a new task whenever a new task is encountered. Knowledge transfer starts when a new classifier (a new RAN-LTM network) is created for a new task. The extent of knowledge transfer depends on the amount of overlap of class regions between the old and new tasks. The actual transfer of knowledge to a new task occurs through the transfer of matching *memory items* of the previous classifier to the new classifier, where memory items are representative examples of the class regions in the previous task. Since training examples for the new task are given sequentially one at a time, the knowledge transfer itself also occurs on an ongoing basis.

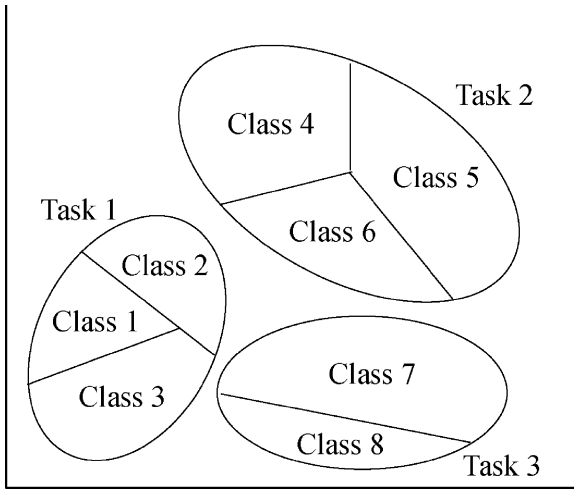


Fig. 3. Example of non-MTPR problems.

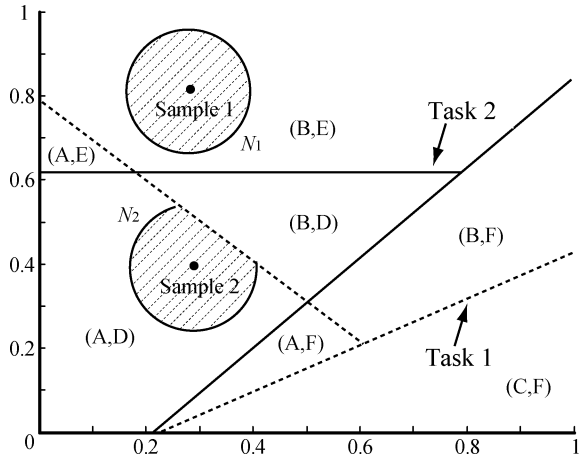


Fig. 4. Schematic diagram to explain the idea for knowledge transfer mechanism. The class boundaries of task 1 and task 2 are represented by dotted and solid lines, respectively. Now we consider a case that the task is changed from task 1 to task 2 and a training example (sample 1 or sample 2) is provided to a learning system.

Knowledge transfer works in the following way. In Fig. 4, the dotted and solid lines define the class boundaries for tasks 1 and 2, respectively. There are seven subregions bounded by these lines and each subregion is denoted by a pair of class labels such as (A, D) where, in this case, class A belongs to task 1 and class D belongs to task 2. Assume that the learning system was learning task 1, had constructed a RAN-LTM classifier for it, and also had stored some memory items (representative examples) for task 1. Then, the first training example from class E of task 2 is given to the learning system. The training example, shown in Fig. 4 as a solid dot and labeled "sample 1," is from the subregion (B, E). Let N_1 be the neighborhood of a radius around sample 1. For sample 1, the neighborhood N_1

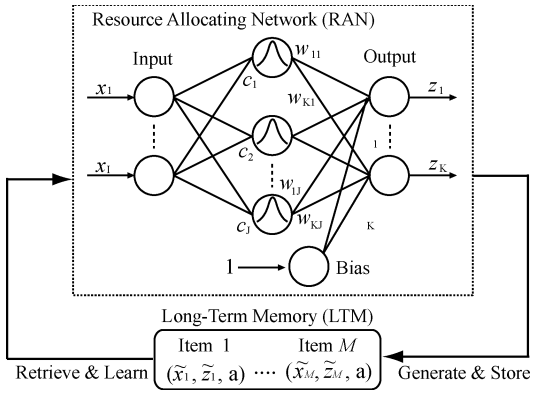


Fig. 5. Structure of resource allocating network with long-term memory (RAN-LTM).

is completely within the region of class B of task 1. So knowledge is transferred from task 1 to task 2 by transferring to the new classifier for task 2 all of the memory items (representative examples) from the task 1 classifier that are within the neighborhood N_1. During this transfer, the class labels of those memory items are changed from "B" to "E" to match the class label of sample 1. Next, suppose the second example of task 2 is "sample 2" in Fig. 4 from the subregion (A, D) and the neighborhood area is N_2. In this case, the neighborhood N_2 crosses over into the subregion (B, D). Since sample 2 is in the class A region of task 1, only class A memory items in the neighborhood N_2 are transferred to the new classifier for task 2 and their class labels changed to "D." Hence, the N_2 neighborhood is shown in a truncated form.

What knowledge transfer does is to add extra training examples from related tasks to facilitate quick and efficient learning of the new task. In the example above, the transfer of memory items added some extra training examples for learning task 2 in addition to the two examples, samples 1 and 2, which are provided to the classifier from the outside.

III. An Algorithm for Online Multitask Learning for Pattern Recognition

As mentioned in Section II-B, the proposed online learning algorithm has three main parts: 1) one-pass online learning of classifiers, 2) automated task recognition, and 3) knowledge transfer between tasks. These three parts are explained below in some more detail.

A. One-Pass Online Learning of Classifiers

1) Network Architecture of RAN-LTM: The RAN-LTM algorithm [11], [12] is the core classification algorithm in our learning system. The Appendix provides further details on the algorithm. Fig. 5 shows the architecture of RAN-LTM that consists of two parts: resource allocating network (RAN) and long-term memory (LTM). RAN uses a single-hidden-layer neural network structure and is similar to radial basis function (RBF) networks [18]–[21]. RAN allocates hidden units automatically during online learning. Let the number of input units, hidden units, and output units be I, J, and K, respectively. As shown in Fig. 5, let the input to the RAN be the vector $\boldsymbol{x} = \{x_1, \ldots, x_I\}^T$, the output of the hidden units be the vector $\boldsymbol{y} = \{y_1, \ldots, y_J\}^T$, and the output of the RAN be the vector $\boldsymbol{z} = \{z_1, \ldots, z_K\}^T$. The hidden and output unit values of the network are computed as follows:

$$y_j = \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2\sigma_j^2}\right), \qquad j = 1, \ldots, J \tag{1}$$

$$z_k = \sum_{j=1}^{J} w_{kj} y_j + \xi_k, \qquad k = 1, \ldots, K \tag{2}$$

where $\boldsymbol{c}_j = \{c_{j1}, \ldots, c_{jI}\}^T$ and σ_j^2 are the center and variance (width) of the jth hidden unit, w_{kj} is the connection weight from the jth hidden unit to the kth output unit, and ξ_k is the bias of the kth output unit. The variance or width σ_j^2 is generally fixed for all RAN networks.

In RAN, each output node represents a class in the classification problem based on the binary encoding scheme. So the kth output node represents the kth class $(k = 1, \ldots, K)$. Ideally, with a properly trained RAN network, if the input vector belongs to class k, then the kth output node should have a value of 1 and all other output nodes should have a value of 0. Less ideally, a point in class A might have an output vector of $(0.8, 0.1, 0.1)$, but the point is still considered to be from class A because the output of the class A node is quite high compared to the other ones.

The training examples stored in LTM are called *memory items*. Let L be the total number of memory items in LTM. Each memory item, which is a representative training example, stores the following information: an input vector $\tilde{\boldsymbol{x}}^{(l)}$, the class label $\tilde{d}^{(l)}$, and the status flag a. So the triplet $(\tilde{\boldsymbol{x}}^{(l)}, \tilde{d}^{(l)}, a)$ $(l = 1, \ldots, L)$ denotes the lth memory item. Define a target output vector $\tilde{\boldsymbol{d}}^{(l)} = \{\tilde{d}_1^{(l)}, \ldots, \tilde{d}_K^{(l)}\}^T$ for the class label $\tilde{d}^{(l)}$ based on the binary encoding scheme. For example, for a point in class A of task 1 in Fig. 1, the target vector is $(1, 0, 0)$, for a point in class B it is $(0, 1, 0)$, and so on. The status flag a has a value of 0 or 1 that represents the status of the memory item: 0 means it is inactive and 1 means it is active. If a memory item is active, then that memory item could be retrieved from LTM for use as a training example. If it is inactive, then it cannot be retrieved for training. In the simple version of RAN-LTM [12], a new memory item is created whenever a new hidden unit is created in the RAN network. That is, the new RBF center and the corresponding target output vector are stored as a memory item. To prevent catastrophic forgetting in RAN, some of these memory items are provided as training examples to the learning system along with the new ones.

2) Learning Algorithm of RAN-LTM: The RAN-LTM learning algorithm is divided into two phases: allocation of hidden units and updating of RBF centers and connection weights between the hidden and output units. The procedure for allocation of hidden units is the same as that of the original RAN [16], except that a new memory item is created whenever a new hidden unit is created. The RBF centers $\boldsymbol{c}_j$ and the connection weights w_{kj} are modified using the conventional steepest descent algorithm for error minimization. To prevent catastrophic forgetting, some of the memory items in LTM are

retrieved and provided as training examples along with the new training example $\boldsymbol{x}$.

Let $\Omega(\boldsymbol{x})$ be the set of memory items in the neighborhood of $\boldsymbol{x}$, which are retrieved from LTM and provided as training examples along with $\boldsymbol{x}$. The total squared error E to be minimized consists of two parts: squared error $E(\boldsymbol{x})$ for the new example $\boldsymbol{x}$ and squared error $E(\tilde{\boldsymbol{x}})$ for the retrieved memory items $\tilde{\boldsymbol{x}}^{(l)}$ in the neighborhood of $\boldsymbol{x}$. These are defined as follows:

$$E = E(\boldsymbol{x}) + E(\tilde{\boldsymbol{x}}) \tag{3}$$

$$E(\boldsymbol{x}) = \sum_{k=1}^{K} \left(d_k - z_k(\boldsymbol{x})\right)^2 \tag{4}$$

$$E(\tilde{\boldsymbol{x}}) = \sum_{l \in \Omega(\boldsymbol{x})} \sum_{k=1}^{K} \left(\tilde{d}_k^{(l)} - z_k(\tilde{\boldsymbol{x}}^{(l)})\right)^2 \tag{5}$$

where $z_k(\boldsymbol{x})$ and $z_k(\tilde{\boldsymbol{x}}^{(l)})$ are the network outputs for $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}^{(l)}$. In the simple version of RAN-LTM, the set $\Omega(\boldsymbol{x})$ is selected based on hidden unit activations. Further details of the RAN-LTM algorithm are in the Appendix.

B. Automated Recognition of Tasks

The following functions are part of the algorithm that automatically recognizes tasks and builds corresponding classifiers for them: 1) detection of task change during online learning, 2) identification of known and unknown tasks and learning within a proper classifier, 3) reorganization of classifiers to correct misallocation of classes to tasks. In the following sections, we provide detailed descriptions of these functions.

1) Detection of Task Changes During Online Learning: There are a variety of methods that detect changes in data streams [8]. We use a heuristic approach [27] to detect task changes and it is based on the prediction error on incoming training examples. Consider any input to a RAN-LTM network. The output at each output node will be a value around between 0 and 1. The output value of a particular output node reflects the confidence with which that class is predicted for the given input, where each output node represents a particular class. Generally, if the output value is smaller than a certain threshold value θ, it means that the confidence is *low* in the prediction that the input is from the class represented by that output node. On the other hand, when the output value is larger than θ, it means that the confidence is *high* in the prediction.

Consider the problem in Fig. 1. Suppose the first RAN-LTM network has learned task 1 fairly well and for the input vector $\boldsymbol{x}$ (denoted as a black square in Fig. 1), the network output vector is $(0, \theta, 0)$. That means confidence is high in the prediction that $\boldsymbol{x}$ is from class B. Suppose the task now changes to task 2 and the first training example from task 2 is the same input $\boldsymbol{x}$, but its label is now class D. Whenever a new class is encountered, the RAN-LTM algorithm adds a new output node to the network. So a fourth output node is added to the network for class D and the target output vector for $\boldsymbol{x}$ is now $(0, 0, 0, 1)$. However, the current RAN-LTM network, trained for task 1, predicts $\boldsymbol{x}$ to be from class B and the output vector is $(0, \theta, 0, 0)$. From (4), the squared error for input $\boldsymbol{x}$ is $(1 + \theta^2)$. Therefore, if the output of the class B node is larger than θ and the error is larger than $(1+\theta^2)$, it signals a potential task change to the task recognition algorithm.

The prediction error of a network on a particular example could be large due to noise (due to overlapping classes within a task) and not due to task change. To avoid the potential error in detection of task changes, the task recognition algorithm looks for some more evidence for a task change. That is to say, when a training example signals a potential for a task change (large prediction error coupled with a new class not seen before), training of the current network is suspended and the current training example is temporally stored in a queue whose maximum size is defined by q. Then, the prediction errors and the class labels are tracked for the next $(q - 1)$ examples. If the average error for the examples in the queue becomes less than the threshold $(1 - \theta)^2$[1] or a known class is given before the queue reaches its size limit, the algorithm infers that there is no task change. Even when the queue reaches its size limit without encountering a training example that belongs to one of the known classes of the current RAN-LTM, no task change is detected by the algorithm if the average prediction error on the q stored examples is less than the threshold $(1 + \theta^2)$. In both of these cases, the current RAN-LTM is adjusted using the examples in the queue, and the queue is emptied. On the other hand, if the queue reaches its maximum size and the average error over q examples exceeds the threshold $(1 + \theta^2)$, it infers that a task change has indeed occurred.

In the above task change detection procedure, the algorithm resumes training when an example of a known class is given and the examples stored in the queue are used for training the current network. This heuristic is based on the two assumptions about how training examples are presented to the system. First, we assume that a certain number of training examples for a task are presented to the system before a switch to a different task. Second, we assume that examples of different classes within a task occur randomly in the training sequence. That also implies that the presentation of examples could be biased. For instance, training examples from a certain class might show up for the first time in the training sequence long after other class examples are presented. In such a case, our task change detection procedure infers that the training examples in the queue from an unknown class actually belong to the current task if an example from a known class appears in the training sequence before the queue reaches its maximum size.

It remains a question how parameter q, the maximum queue size, is determined. In general, it is difficult to determine q analytically before the start of online learning. It, therefore, has to be set using some heuristic. The main criterion that guides the setting of q is detection of a task change as quickly as possible. Setting of q, therefore, is a tradeoff between reliability and quickness of task change detection. However, not much is lost with task change detection failures because the algorithm has a backup method to recover from such failures (see Section III-B3). Section V-C discusses how q is set for empirical tests of the algorithm.

[1]The prediction accuracy is good if the squared error is *less* than $(1 - \theta)^2$, which means the largest output is larger than θ and the predicted class matches the target class.

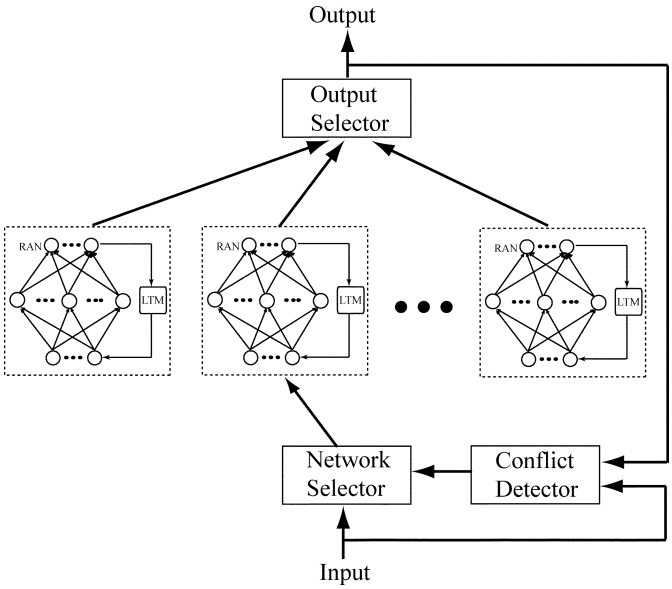


Fig. 6. Proposed model for MTPR problems that uses multiple RAN-LTMs for multiple tasks.

2) Identification of Known and Unknown Tasks and Learning Within a Proper Classifier: After detecting a task change, the task recognition algorithm has to identify whether the next task is known or unknown. This is done by evaluating the prediction errors for existing RAN-LTM networks.

Consider again the learning sequence in Fig. 2. After the system trains with the training examples of the first task through the third task, there will be three RAN-LTM networks in the system, one for each task. When the training examples of the fourth task (which are actually task 1 examples, as shown in Fig. 2) are presented next, the system should detect, using the procedure described above, that a task change has occurred. When a task change is detected, the system first tries to find a RAN-LTM network that is consistent with the q stored training examples in the queue. Consistency is checked on the basis of average prediction error for the q stored training examples. If a RAN-LTM whose average error is less than $(1 + \theta^2)$ is found, it indicates that the new task was encountered before. That particular RAN-LTM is then retrieved and trained with the set of q stored examples for that task. For the problem of Fig. 2, therefore, when training examples of the fourth epoch (which are actually task 1 examples) are presented to the system, it will recognize that there already is a RAN-LTM network for task 1. It will then retrieve the RAN-LTM network for task 1 and train it with the set of queued examples.

On the other hand, if no existing RAN-LTM matches the new task examples in the queue, a new RAN-LTM is created for the new task. To transfer knowledge to the new task, the newly created RAN-LTM inherits all of the memory items from the current RAN-LTM. Fig. 6 shows the structure of the proposed system that is composed of the conflict detector (to detect a new task as explained above) and multiple RAN-LTMs that are responsible for learning individual tasks. Note that the system must learn different tasks on a *continuous basis* and not wait until the end of training to group classes into tasks.

3) Reorganization of Classifiers to Correct Misallocation of Classes to Tasks: Since task change detection is based on a heuristic, there is no guarantee that the task recognition algorithm always detects task changes correctly. Misdetection of task changes can result in misallocation of classes to appropriate tasks (or networks). Thus, a backup measure is required to compensate for such failures and it should be carried out on an ongoing basis to maintain accurate task recognition.

Misallocation of classes to tasks is caused mainly by: 1) an inadequate number of examples to learn about class regions and 2) the lack of class examples to learn a task. The following situation illustrates the first case. For the problem of Fig. 1, classes A of task 1 and D of task 2 have an overlapped region in their input domains. Suppose that class A examples of task 1 are sequentially drawn from a part of the overlapped region and they are used for training during the first epoch. Then, suppose task 1 is changed to task 2, and several class D examples of task 2 are sequentially drawn from a different part of the overlapped region in the first several stages of the second epoch. In this case, the given examples of classes A and D are from the overlapped region between the two classes, but they happen to have no significant overlap with each other. In this case, the class D examples would be stored temporarily in the queue and then would be used to train the current task 1 network unless the average error becomes larger than the threshold (i.e., a task change is detected) when the queue reaches its maximum size. The end result could be that class D becomes part of task 1. Therefore, the algorithm could miss the task change when task 1 is changed to task 2.

The following situation illustrates the second case. Again, for the problem of Fig. 1, suppose that training examples in the first epoch are sequentially drawn from classes B and C, and the network is trained only for those two classes. Then, suppose task 1 is changed to task 2, and several class D examples, which are drawn from the overlapped region with class A of task 1, are presented for training in the first several stages of the second epoch. Since the above class D examples have no significant overlap with the classes B and C of task 1, the algorithm would miss the task change when task 1 is changed to task 2. Again, the end result could be that class D becomes part of task 1.

In the first case, where class D examples are misallocated to the task 1 network, the class A examples in that particular overlapped region with class D can be no longer allocated to task 1. Thus, class A appears in both networks of tasks 1 and 2. The same thing could happen in the second case. Suppose that class D happens to be allocated to the task 1 network along with classes B and C. If several class D examples in the overlapped region with class B are presented, a task change would be detected because the queued examples of class D would have a large prediction error. Thus, parts of class D could appear in both networks. In general, as a result of the flaw in the task change detection heuristic, the same class can exist in multiple networks.

To rectify this misallocation of classes to multiple networks, we use a procedure to reorganize and consolidate classifiers. Assume that class A is assigned to or exists in two RAN-LTMs denoted by RAN-LTM(m) and RAN-LTM(n). Let $T(m)$ and $T(n)$ be the set of memory items for that class

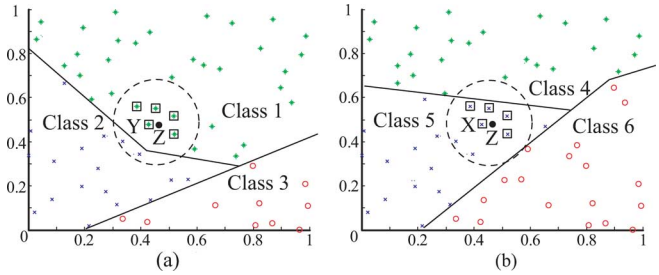


Fig. 7. Schematic diagram to explain knowledge transfer that is actually realized by replacing the class labels of memory times marked by squares with the label of X in task 2. The solid dot denoted by Z represents a given training example; X and Y are the closest memory items to Z for task 2 and task 1, respectively. Note that the boxed points in the figures represent memory items to be transferred. (a) Task 1. (b) Task 2.

in RAN-LTM(m) and RAN-LTM(n), respectively. To consolidate class A, either $T(m)$ should be moved to RAN-LTM(n) or $T(n)$ should be moved to RAN-LTM(m). To determine which set of memory items is to be moved, the prediction error $E(m)$ of RAN-LTM(m) is estimated for $T(n)$; in the same way, the prediction error $E(n)$ of RAN-LTM(n) is estimated for $T(m)$. If $E(m) < E(n)$, class A is consolidated into RAN-LTM(m); otherwise, it is consolidated into RAN-LTM(n). To minimize the overall error and the number of classifiers, the reorganization procedure checks the error for every shared class between two RAN-LTMs, and this procedure is applied to every combination of two RAN-LTMs. After the reorganization operation, every class is allocated to one and only one task (or network).

IV. Transfer of Knowledge Between Tasks

If multiple tasks are related to one another, then the learning of a particular task among them can be speeded up if knowledge is transferred from another related task that has already been learned. Baxter [3] has shown the theoretical effectiveness of such knowledge transfer. When there are several learned tasks that are related to the new task, it is important to choose the one most closely related to the knowledge transfer. There is considerable work on selecting the right source for such transfer of knowledge [2], [17], [26]. Silver and Mercer [22] have proposed several ways to measure the relatedness of tasks.

The knowledge transfer procedure in our system works in the following way. For the problem of Fig. 7, suppose the system so far has been trained with task 1 examples and the task now is changed to task 2. After detection of a task change, the algorithm transfers all memory items in the current RAN-LTM to the new RAN-LTM for the new task, but they remain inactive (i.e., the status flag a is set to 0 for all memory items in the new RAN-LTM). That means that none of the transferred memory items can be used yet for learning task 2. In Fig. 7(a) and (b), all of the points are memory items. Now suppose that a new "class 5" training example Z (solid dot) of task 2, around the point $(0.45, 0.5)$, is provided to the system. The closest memory item to Z in Fig. 7(b) is X, and Y in Fig. 7(a) is the memory item corresponding to X in Fig. 7(b). The knowledge transfer is achieved through activation of a set of the transferred memory items. In this case, it is the set of Y and memory items around Z. In the neighborhood of Z (dotted circle), there are seven memory items. Based on the knowledge transfer ideas of Section II-B2, five of the seven memory items including Y itself, those that have the same class label as Y, are activated for task 2 [see the five squares in Fig. 7(a)]. Therefore, the class labels of the corresponding five transferred memory items in Fig. 7(b), denoted by the squares, are set to "class 5," which is the label of Z, and these memory items are then *activated*. That means that they can be used for learning task 2 from now on. The procedure illustrated here is the basic method of knowledge transfer for a single new training example of the new task. However, this knowledge transfer occurs on an ongoing basis for every new training example of the new task. Thus, whenever a new training example for the new task is given, the class labels of the *inactive memory items* in its neighborhood are changed as per this procedure and then activated for training purposes. Note that the status flag a of activated memory items is set to "1" to distinguish them from inactive ones.

The effectiveness of this knowledge transfer mechanism depends on the size of the neighborhood. A large neighborhood transfer can bias a new task in a detrimental way, whereas a small neighborhood transfer may not transfer enough knowledge. On the other hand, a large neighborhood transfer can be useful if the two tasks are highly related. In general, it is difficult to determine the size of the neighborhood transfer analytically either before or during learning. So the algorithm has a mechanism to minimize as much as possible the problem of too much or too little neighborhood transfers. In this algorithm, incorrect knowledge transfer is caused by the assignment of incorrect class labels to the transferred memory items in the new RAN-LTM. So it uses the following mechanism to correct such incorrect knowledge transfers: every time a new training example is given, the algorithm first finds the closest active memory item and checks the class label of the memory item against that of the training example. If there is discrepancy between them, the class label of the active memory item is changed to that of the training example.

A. Automated Task Recognition–A Summary of the Algorithm

Here is a step-by-step description of the automated task recognition algorithm.

[Automated Task Recognition Algorithm]

Step 0: Initialize: For the first training example $(\boldsymbol{x}, d_1)$, create a classifier (RAN-LTM) with a single hidden unit and a single output unit, set the center vector $\boldsymbol{c}_1$ to x, and initialize the weight vector $\boldsymbol{w}$ to w_{11} (i.e., $\boldsymbol{w} = \{w_{11}\}$). Set the number of output units K to 1. Define an empty queue Q where the last q training examples can be stored.

Step 1: New class encountered, create a new output unit: If a training sample $(\boldsymbol{x}, d)$ is given and d is an unknown class label, add a new output unit $(K \leftarrow K + 1)$. Define the output variable and the corresponding class label by z_K and d_K, respectively.

Step 2: Calculate average error for a new training example and for those in the queue: Store a new training example $(\boldsymbol{x}, d)$ in Q. Obtain the output $\boldsymbol{z} = \{z_1, \ldots, z_K\}^T$ of the

current classifier (RAN-LTM) for all the training examples in Q, and calculate the average squared error $\bar{E}_Q$.

Step 3: Check if the prediction of RAN-LTM is consistent with a given target signal—if so, update current RAN-LTM; otherwise, check if enough evidence for a task change has been collected: If $\bar{E}_Q < (1-\theta)^2$ or the class of the new training example is already trained by the current RAN-LTM, then go to Step 7. Else if the number of training examples in Q is less than q, go back to Step 1.

Step 4: Reorganize Classifiers: Perform the *Reorganize Classifiers* procedure described below.

Step 5: Check if there is an existing RAN-LTM for this task—if so, switch to it: Search all existing RAN-LTMs to check if the average error for the training examples in Q is less than $(1+\theta^2)$. If one exists, switch to it as the current classifier and go to Step 7.

Step 6: Consistent high error for last q examples—new task encountered, create new RAN-LTM: Perform the following procedure to create a new RAN-LTM for the new task:

1) Create a new RAN-LTM, and copy all memory items of the current RAN-LTM to it;
2) deactivate all the transferred memory items in the new RAN-LTM.

Step 7: Update current RAN-LTM: Perform the following procedure for all training examples in Q:

1) Perform the *Update Memory Item* operation.
2) Perform the *Do Knowledge Transfer* operation.
3) Update the current RAN-LTM using the learning algorithm in the Appendix.
4) Empty Q and go back to Step 1.

[Update Memory Item]

Step 1: Search for the nearest memory item: Find the active memory item $(\tilde{\boldsymbol{x}}^*, \tilde{d}^*, a)$ closest to $(\boldsymbol{x}, d)$.

Step 2: Class label update: If $\tilde{\boldsymbol{x}}^* \approx \boldsymbol{x}$ and $\tilde{d}^* \neq d$, update the class label by setting $\tilde{d}^* = d$.

[Do Knowledge Transfer]

Step 1: Define the neighborhood of the next training example in Q: Define the neighborhood $\Omega(\boldsymbol{x})$ of the next training example $(\boldsymbol{x}, d)$ in Q by a hypersphere whose center and radius are $\boldsymbol{x}$ and η, respectively.

Step 2: Search for the closest nonactive memory item: Find the nonactive memory item $(\tilde{\boldsymbol{x}}^*, \tilde{d}^*, a=0)$ closest to $(\boldsymbol{x}, d)$ within $\Omega(\boldsymbol{x})$. If there is no nonactive memory item found, go to Step 5.

Step 3: Search for other nonactive memory items to be transferred: Find all nonactive memory items $(\tilde{\boldsymbol{x}}^{(l)}, \tilde{d}^{(l)}, a=0)$ within $\Omega(\boldsymbol{x})$ whose class label is $\tilde{d}^*$ (i.e., $\tilde{d}^{(l)} = \tilde{d}^*$).

Step 4: Activate the memory items: For the nearest nonactive memory item $(\tilde{\boldsymbol{x}}^*, \tilde{d}^*, a=0)$ and all the memory items $(\tilde{\boldsymbol{x}}^{(l)}, \tilde{d}^{(l)}, a=0)$ found in Step 3, replace the class labels with d (i.e., $\tilde{d}^* = d$ and $\tilde{d}^{(l)} = d$), and activate the memory items (i.e., set the status flag $a = 1$).

Step 5: If there is no training example left in Q, terminate this procedure. Otherwise, go back to Step 1 for the next example in Q.

[Reorganize Classifiers]

Do the following procedure for all combinations of (n, m) where n and m $(n < m)$ are the classifier numbers.

Step 1: Find class labels shared between two RAN-LTMs: Find the class labels that occur in both the nth RAN-LTM(n) and the mth RAN-LTM(m). Define the set of shared class labels by L_1. If $L_1 = \phi$, terminate the reorganize procedure for these two RAN-LTMs.

Step 2: Find the active memory items for the shared classes: Let $T(m)$ and $T(n)$ be the set of active memory items in RAN-LTM(m) and RAN-LTM(n), respectively, whose class labels are in L_1.

Step 3: Define the direction of consolidation: Calculate the average error $E(n)$ of RAN-LTM(n) for $T(m)$, and the average error $E(m)$ of RAN-LTM(m) for $T(n)$. If $E(m) > E(n)$, set $\mathrm{src} = m$ and $\mathrm{trg} = n$. Otherwise, set $\mathrm{src} = n$ and $\mathrm{trg} = m$.

Step 4: Consolidate the shared classes in RAN-LTM(src) into RAN-LTM(trg): Train RAN-LTM(trg) using the memory items in $T(\mathrm{src})$. For RAN-LTM(src), delete the memory items in $T(\mathrm{src})$, the output units for the classes in L_1, and their weight connections.

Step 5: Consolidate the unshared classes in RAN-LTM(src) into RAN-LTM (trg) if necessary: Define the set L_2 of all class labels in RAN-LTM(src) that are not included in L_1. Perform the following procedure for each class label $l \in L_2$:

1) define the set $T(\mathrm{src})$ of active memory items whose labels are class l;
2) calculate the average error of RAN-LTM(trg) for $T(\mathrm{src})$;
3) if the error is smaller than $(1+\theta^2)$, train RAN-LTM(trg) using the memory items in $T(\mathrm{src})$. For RAN-LTM(src), delete the memory items in $T(\mathrm{src})$, the output unit for the class l, and its weight connections.

Step 6: If there is no class left in RAN-LTM(src), delete RAN-LTM(src).

V. Experimental Evaluation of the Automated Online Task Learning System

A. Evaluation Method and Performance Measures

Since the performance of the system could potentially be influenced by the order of presentation of tasks and training examples, the algorithm is evaluated using the average results from

TABLE I
SOME RANDOM TASK SEQUENCES FOR LEARNING THREE DIFFERENT TASKS. HERE "T_1," "T_2," AND "T_3" STAND FOR AN EPOCH OF TASK 1, TASK 2, AND TASK 3, RESPECTIVELY, WHERE EACH EPOCH CONSISTS OF SOME NUMBER OF TRAINING EXAMPLES OF THE CORRESPONDING TASK

Session	Task Sequences
1	$T_1, T_2, T_1, T_2, T_1, T_2, T_1, T_3, T_2, T_3, T_2, T_3, T_1, T_3, T_2, T_3, T_1, T_3$
2	$T_1, T_2, T_1, T_3, T_2, T_3, T_2, T_1, T_2, T_3, T_1, T_2, T_3, T_2, T_1, T_3, T_1, T_3$
3	$T_2, T_3, T_2, T_1, T_3, T_2, T_1, T_2, T_1, T_3, T_1, T_3, T_1, T_3, T_2, T_3, T_1, T_2$
4	$T_3, T_1, T_3, T_1, T_2, T_1, T_2, T_1, T_2, T_3, T_2, T_1, T_2, T_3, T_1, T_3, T_2, T_3$
5	$T_1, T_3, T_1, T_2, T_3, T_1, T_2, T_3, T_2, T_1, T_3, T_2, T_3, T_1, T_2, T_1, T_2, T_3$

TABLE II
INFORMATION ON THE FIVE UCI DATASETS

	#Attributes	#Classes	#Train. Samples	#Test Samples
Segmentation	19	7	210	2,100
Thyroid	21	3	3,772	3,428
Vehicle	18	4	188	658
Vowel	10	11	528	462
Yeast	8	10	744	740

50 different sequences for each problem that are generated such that both tasks and training samples are presented in random order. This section defines the performance evaluation measures and related concepts.

In our experiments, the total number of training examples (all tasks combined) in a training session is fixed at 720 and each session has three tasks to learn. Although all epochs within a training session have a fixed number of training examples, the number of training examples per epoch is varied across the 50 different training sessions for a problem to remove any bias from using a fixed epoch size. To illustrate, when the number of training examples per epoch is 40, the number of epochs in the training session is 18 $(= 720/40)$ and 18 tasks (or epochs) are presented sequentially to the learning system. Table I shows some sample task sequences for different training sessions. In our experiments, the number of training examples per epoch is varied from ten to 80 and the number of epochs consequently varies from nine to 72.

We test the algorithm on a set of problems to evaluate: 1) whether new tasks are automatically recognized on an ongoing basis and training examples are assigned to the appropriate classifier (RAN-LTM) for the task, 2) whether its knowledge transfer mechanism enhances the classification accuracy when an unknown new task is presented, and 3) whether the mechanism of reorganizing classifiers works well and properly assigns classes to tasks (task categorization). So the following specific measures are used to evaluate the performance of the algorithm: a) the accuracy of task-change detection, b) the task categorization accuracy (accurate assignment of classes to tasks), c) the classification accuracy after a complete training session, and d) the classification accuracy after the first epoch of a completely unknown task. Next we define these measures in detail.

a) *Accuracy of task-change detection* measures the percentage of times the system correctly detects an actual task change. It is the ratio of the number of successful task-change detections over the total number of actual task changes in a training session. The total number of actual task changes is always known in advance. For example, there are 18 task changes in a training session with 720 examples when the number of examples per epoch is 40. They occur when the following training examples are presented: the 41st, 81st, 121st, and so on. So it is easy to verify how many of these actual task changes are correctly detected.

b) *Task categorization accuracy* measures the success rate in assigning classes to the right classifiers for tasks, and this accuracy is evaluated only after training is complete. For the example of Fig. 1, task categorization would be successful if, after training, classes A, B, and C are in one classifier, classes D, E, and F are in a second classifier, and classes G, H, and I are in a third classifier. Hence, task categorization is successful if the system assigns classes of a particular task to the corresponding classifier for that task. On the other hand, task categorization fails if a classifier has classes of different tasks and/or the class labels of a particular task separately allocated into more than one classifier. Redundancy of classifiers, where the system creates more classifiers than tasks, does not by itself result in any performance degradation. So multiple classifiers for a single task could still be successful in separating classes into tasks as long as the classes are properly assigned to each and every task. However, since this leads to inefficiency in computations and memory usage, we do not count it as successful task categorization when more than one classifier is created for a single task. We should also note here that the system cannot predict the task of a particular test example. Although it builds a separate classifier for each task, predicting the task of a particular test example is difficult without some additional information because the same input belongs to different classes of different tasks.

c) *Classification accuracy* measures the percentage of times the class of a test example is predicted correctly. Like training examples, test examples do not have task labels either. So the basic prediction is at the class level. Since all tasks are from the same input space, the system makes a class prediction for each task. For example, for the problem of Fig. 1, suppose a test example $\boldsymbol{x}$ is from class B of task 1. During testing, the system will predict that $\boldsymbol{x}$ is from one of the following classes: class B of task 1, class D of task 2, or class H of task 3. Since $\boldsymbol{x}$ is from class B of task 1, the prediction is actually correct and the system treats this as a correct prediction for the purposes of measuring classification accuracy. The basic idea of this system is to group and learn various descriptors of similar objects and expect one of the predicted descriptors on a test object to be correct. The classification accuracy is estimated on all test data after

TABLE III
TWO MTPR PROBLEMS CREATED FROM THE UCI SEGMENTATION DATA

(a) Problem 1

Original	Task 1	Task 2	Task 3
1	1	8	10
2	2	8	10
3	3	8	11
4	4	9	11
5	5	9	12
6	6	9	12
7	7	9	12

(b) Problem 2

Original	Task 1	Task 2	Task 3
1	1	3	5
2	2	3	5
3	1	3	6
4	2	4	6
5	1	4	7
6	2	4	7
7	1	4	7

TABLE IV
TWO MTPR PROBLEMS CREATED FROM THE THYROID DATA

(a) Problem 1

Original	Task 1	Task 2	Task 3
1	1	4	6
2	2	4	7
3	3	5	7

(b) Problem 2

Original	Task 1	Task 2	Task 3
1	1	3	5
2	2	3	6
3	1	4	6

TABLE V
TWO MTPR PROBLEMS CREATED FROM THE VEHICLE DATA

(a) Problem 1

Original	Task 1	Task 2	Task 3
1	1	5	7
2	2	5	8
3	3	6	7
4	4	6	8

(b) Problem 2

Original	Task 1	Task 2	Task 3
1	1	3	5
2	2	3	6
3	2	4	5
4	1	4	6

the training is successfully completed and is called the *final classification accuracy*. Note that the final accuracy is measured only when task categorization is successful. This is because failure in task categorization can degrade the classification accuracy. Task categorization failure cases are, therefore, ignored so that classification accuracies can be compared on a fair basis.

To observe the evolution of learning by the system, classification accuracy is also evaluated on an ongoing basis at every learning step using a randomly selected subset of test examples. This ongoing estimation of accuracy is not only for tasks learned up to a certain point in a training session, but also for tasks not yet learned, and it helps to verify whether knowledge transfer is effective in online task learning. In general, knowledge transfer from one task to another occurs mainly when an unknown task is first encountered, and if the transfer works well, the classification accuracy of the system should be higher for the epoch right after such a new task appears compared to when there is no such knowledge transfer. Therefore, the acceleration in learning can be measured by the average accuracy for the epoch when an unknown task first appears.

B. Data Sets for Experimental Evaluation

To evaluate the robustness of the task learning system, ten multitask MTPR problems with noisy, overlapping classes are defined based on the following five data sets from the UCI Machine Learning Repository: Segmentation, Thyroid, Vehicle, Vowel, and Yeast. Information on these five data sets is summarized in Table II. Although Segmentation, Vehicle, and Vowel data sets have fairly even distribution of class examples, that is not true for Thyroid and Yeast data sets. The Yeast data set in particular has a very skewed class distribution; two classes have more than 200 training examples, but six other classes have less than 30 examples. The largest and smallest numbers of class examples are 232 and 3, respectively. So the smallest class examples are rarely presented in a training session. The Yeast problem, therefore, is the hardest problem of the five.

Since the UCI data sets are for a single classification task, we combined the original classes in different ways to create additional tasks such that every task has some relatedness to each other. Tables III–VII show how the original classes are combined to form new classes for the new tasks. Here is how the seven Segmentation classes of Table III(a) are combined to form new tasks. In Problem 1 in Table III(a), the classes of task 1 are the same as those of the original problem; therefore, task 1 is a seven-class classification problem. Task 2 is defined to be a two-class problem with class labels 8 and 9. As shown in the table, class 8 of task 2 combines classes 1–3 of the original problem and class 9 of task 2 combines classes 4–7. Task 3 is defined in a similar way. The other MTPR problems shown in Tables IV–VII are defined in a similar manner. Since the difficulty of MTPR problems could depend on the grouping of classes, we defined two MTPR problems for each data set as shown in the tables. For the two MTPR problems for Yeast data, we combined small classes with large ones to define new classes so that the new class distributions are more balanced than the original ones.

C. Setting Parameters of the Algorithm

In this learning system, several parameters have to be set. As shown below, those parameters can be divided into three groups: (1) the ones related to the underlying RAN-LTM classifier, (2) the ones associated with task change detection, and (3) the one related to knowledge transfer.

TABLE VI
TWO MTPR PROBLEMS CREATED FROM THE VOWEL DATA

(a) Problem 1

Original	Task 1	Task 2	Task 3
1	1	12	14
2	2	12	15
3	3	12	14
4	4	12	15
5	5	12	14
6	6	12	15
7	7	13	14
8	8	13	15
9	9	13	14
10	10	13	15
11	11	13	14

(b) Problem 2

Original	Task 1	Task 2	Task 3
1	1	3	5
2	2	3	5
3	1	3	6
4	2	3	6
5	1	3	7
6	2	4	7
7	1	4	8
8	2	4	8
9	1	4	9
10	2	4	9
11	1	4	9

TABLE VII
TWO MTPR PROBLEMS CREATED FROM THE YEAST DATA

(a) Problem 1

Original	Task 1	Task 2	Task 3
1	1	11	13
2	2	12	14
3	3	11	15
4	4	12	16
5	5	12	16
6	6	12	16
7	7	11	16
8	8	12	16
9	9	11	16
10	10	12	16

(b) Problem 2

Original	Task 1	Task 2	Task 3
1	1	3	5
2	2	3	5
3	1	3	6
4	2	3	6
5	1	4	7
6	2	4	7
7	1	4	8
8	2	4	8
9	1	4	9
10	2	4	9

[RAN-LTM classifier]
— σ^2 (width of radial basis functions)
— δ (minimum distance between two nearest RBF centers)
— ε (tolerance for approximation error)
[Task change detection]
— θ (threshold for class prediction at an output node)
— q (minimum length of queue Q)
[Knowledge transfer]
— η (radius of knowledge transfer region)

Other than the RAN-LTM parameters, the parameters θ, q, and η are the new ones introduced by this task learning system. Since MTPR problems are classification problems, the threshold for class prediction θ is set to 0.5 in all test problems because the output of an output node reflects the likelihood of the corresponding class.

The main problem is with setting the three inter-related RAN-LTM parameters: σ^2, the width of the RBFs, δ, the minimum distance between them, and ε, the tolerance for approximation error. This problem is inherent to many online RBF algorithms that cannot adaptively define these parameters during learning; therefore, this is not a problem created by the multitask learning concept presented here. Roy *et al.* [18]–[21] have developed a version of the RBF algorithm where the algorithm itself determines the widths and centers of the RBFs and therefore the problem of presetting these parameters goes away. But currently there is no online version of that algorithm. We hope to develop an online version of RAN-LTM that incorporates the ideas of Roy *et al.* For this paper, however, we report our experience with the current RAN-LTM algorithm of Ozawa *et al.* [12].

Table VIII(a) shows the final classification, task-change detection, and task categorization accuracies for different RBF widths (σ^2) for the Segmentation data set. Table VIII(b) shows the same accuracies for different values of q and Table VIII(c) shows the effect of the parameter η on these accuracies. As seen from Table VIII(a), task categorization and classification accuracies fall drastically when the width σ^2 is less than 0.5. This is caused by poor generalization ability of RAN-LTM with small RBF widths, which results in incorrect prediction on known or unknown tasks.

From Table VIII(b), one can see that task-change detection and task categorization accuracies decrease significantly when q is equal to 1. This is easily expected since task change decisions are being made with single evidence; therefore, the system tends to be deceived into detecting task changes at wrong places

TABLE VIII
THE ACCURACIES (IN PERCENT) OF FINAL CLASSIFICATION, TASK CHANGE DETECTION, AND TASK CATEGORIZATION FOR DIFFERENT MODEL PARAMETERS: (a) WIDTH OF RBF σ^2, (b) MINIMUM SIZE OF QUEUE q, AND (c) RADIUS OF KNOWLEDGE TRANSFER REGION η. THE DATA SET USED IS "SEGMENTATION"

(a)

	0.1	0.5	1	2	3	5	10
Classification	40.1	71.1	81.3	85.0	85.4	84.8	80.0
Detection	94.7	98.5	99.4	99.4	99.3	99.2	99.2
Task categ.	85	98	100	100	97	96	92

(b)

	1	2	3	5	7	10
Classification	84.7	83.8	84.4	85.0	85.2	85.4
Detection	24.9	85.9	96.4	99.4	99.9	100
Task categ.	32	90	93	100	100	100

(c)

	0.01	0.05	0.1	0.2	0.5	1
Classification	85	85	85	85	83.9	83.4
Detection	99.4	99.4	99.4	99.4	99.4	99.4
Task categ.	100	100	100	100	99	98

TABLE IX
AVERAGE ACCURACIES (IN PERCENT) OVER ALL PARAMETER COMBINATIONS FOR TEN DIFFERENT TASK SEQUENCES AND THE AVERAGES FOR THE STANDARD PARAMETER SETTING USED IN THE EXPERIMENTS. THE DATA SET USED IS "SEGMENTATION"

	Standard parameter	All parameter combinations
Classification	85	82.5
Task categ.	100	88.3

by noisy data such as outliers and data within overlapped class regions. On the other hand, the task categorization accuracy for $q = 2$ is greatly improved from the case of $q = 1$ in spite of keeping one more evidence in the queue. Considering that the task categorization accuracy for $q = 2$ is better than the accuracy of task-change detection, this improvement results not only from the improvement in task-change detection but also from the introduction of the reorganization process. On the contrary, however, the classification accuracy for $q = 2$ is a little worse than that for $q = 1$. This is because the classification accuracy is measured only when task recognition succeeds and because the accuracy is generally deteriorated through the reorganization process. As seen from the algorithm of *Reorganize Classifier*, when two classifiers are consolidated, the training data to recalculate the weight connections of a consolidated classifier are the memory items, which correspond to a small subset of training data given so far. Therefore, it is inevitable to lose certain classification accuracy by this retraining especially when the reorganization process is often invoked. However, the degradation in the classification accuracy is not serious as long as we see the results in Table VIII(b).

Overall, these results show that the algorithm works quite well for $q \geq 5$. So we set q to 5 in our experiments across all problems. As seen from Table VIII(c), the various accuracies are also not affected greatly by η, so we used $\eta = 0.1$ in our experiments. The RAN-LTM parameters are set to the following values: $\sigma^2 = 2$, $\delta = 1$, and $\varepsilon = 0.05$.

The important parameters in this algorithm are: the RBF width σ^2, the queue length q, and the radius of knowledge transfer region η. We evaluated the final classification and task categorization accuracies for the following parameter combinations on the Segmentation data set: $\sigma^2 = 1, 2, 3, 5, 10$, $q = 2, 3, 5, 7, 10$, and $\eta = 0.01, 0.05, 0.1, 0.2, 0.5, 1$. Average accuracies over all parameter combinations for ten different task sequences are shown in Table IX along with the averages for the standard parameter setting used in our experiments: $\sigma^2 = 2$, $q = 5$, and $\eta = 0.1$. From these results, one can see that good performance can be obtained from a broad range of parameter combinations and that there never is a need to fine-tune the parameters to a given problem. In general, the algorithm is fairly robust for a broad range of reasonable parameter values.

D. Performance Evaluation

The proposed multitask learning algorithm includes two distinctive functions: 1) knowledge transfer and 2) reorganization of classifiers. In this section, we verify the effectiveness of these two functions through experimental results. As per our discussion in the previous section, the following parameters are used in the experiments: $\sigma^2 = 2$, $\delta = 1$, $\varepsilon = 0.05$, $\theta = 0.5$, $q = 5$, and $\eta = 0.1$.

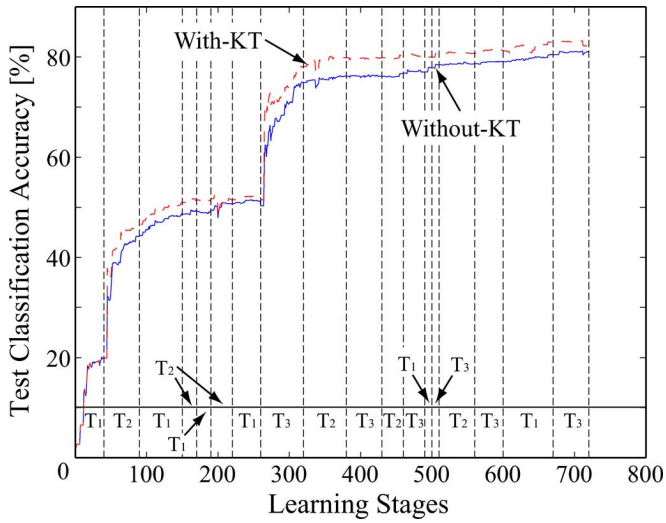


Fig. 8. Time evolution of test classification accuracy (in percent) in the proposed system with knowledge transfer (with-KT) and without knowledge transfer (without-KT). The tasks are provided in the sequence: $T_1 \rightarrow T_2 \rightarrow T_1 \rightarrow T_2 \rightarrow T_1 \rightarrow T_2 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_3 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_3$ where "T_1," "T_2," and "T_3" stand for task 1, task 2, and task 3, respectively. The learning stage (X-axis) is a count of the number of examples presented to the system. The data set used is "Segmentation."

1) Evaluation of Knowledge Transfer: The basic idea behind transferring knowledge from an old task to a new task is to improve the generalization performance and to speed up the learning of new tasks [6], [22]. To measure the effectiveness of knowledge transfer, the task recognition system is evaluated in two ways—with and without knowledge transfer. Fig. 8, for the segmentation data set, shows the overall classification accuracy over time with knowledge transfer (with-KT) and without knowledge transfer (without-KT) for the particular task sequence $T_1(40) \rightarrow T_2(50) \rightarrow T_1(60) \rightarrow T_2(20) \rightarrow T_1(20) \rightarrow T_2(30) \rightarrow T_1(40) \rightarrow T_3(60) \rightarrow T_2(60) \rightarrow T_3(50) \rightarrow T_2(30) \rightarrow T_3(30) \rightarrow T_1(10) \rightarrow T_3(10) \rightarrow T_2(50) \rightarrow T_3(40) \rightarrow T_1(70) \rightarrow T_3(50)$ where the values in parenthesis are the number of training samples in that epoch. The new tasks T_2 and T_3 first appear when the 41st and 261st training examples are presented (i.e., at the learning stage 41 and 261), respectively. As seen in Fig. 8, when these new tasks are first presented, the classification accuracy increases rapidly, but the improvement for "with-KT" is higher than that for "without-KT." Thus, knowledge transfer seems to be quite effective in this particular problem. Note that the proposed system works well even when the number of training examples for an individual task varies at every learning epoch.

The effectiveness of knowledge transfer is further verified by the final classification accuracies. Table X shows the average final classification accuracies over 50 learning sessions with 720 training examples in each session for the five UCI data sets. The two values in each cell of Table X are the mean and standard deviation of the final classification accuracies. The single asterisk (*) and double asterisks (**) mean that the average difference between "with-KT" and "without-KT" is statistically

TABLE X
FINAL CLASSIFICATION ACCURACIES (IN PERCENT) OF THE TASK RECOGNITION SYSTEM WITH KNOWLEDGE TRANSFER (WITH-KT) AND WITHOUT KNOWLEDGE TRANSFER (WITHOUT-KT) FOR THE FIVE UCI DATA SETS. THE ACCURACY IS MEASURED AFTER COMPLETION OF TRAINING WITH A SEQUENCE OF 720 TRAINING SAMPLES. THE TWO VALUES IN EACH CELL ARE THE AVERAGE ACCURACY AND THE STANDARD DEVIATION IN THE FORM OF (AVERAGE) ± (STANDARD DEVIATION). THESE ACCURACIES ARE AVERAGED OVER 50 DIFFERENT LEARNING SESSIONS. THE SINGLE ASTERISK (*) AND THE DOUBLE ASTERISKS (**) MEAN THAT THE AVERAGE DIFFERENCE BETWEEN WITH-KT AND WITHOUT-KT IS SIGNIFICANT AT 5% AND 1% LEVEL, RESPECTIVELY

(a) Segmentation

	Problem 1	Problem 2	Average
without-KT	81.7±1.0	85.1±1.2	83.4±2.1
with-KT	82.9±1.4**	87.1±1.1**	85.0±2.4**

(b) Thyroid

	Problem 1	Problem 2	Average
without-KT	81.2±0.7	81.3±0.7	81.2±0.7
with-KT	83.7±1.9**	83.9±1.9**	83.8±1.9**

(c) Vehicle

	Problem 1	Problem 2	Average
without-KT	77.0±0.8	78.6±0.8	77.8±1.1
with-KT	77.9±1.2**	79.6±1.1**	78.8±1.4**

(d) Vowel

	Problem 1	Problem 2	Average
without-KT	56.8±2.1	60.6±2.0	58.7±2.8
with-KT	57.7±2.5	61.9±2.1**	59.9±3.1*

(e) Yeast

	Problem 1	Problem 2	Average
without-KT	48.0±2.8	53.5±2.4	51.9±3.5
with-KT	49.0±2.1	53.9±2.9	52.5±3.5

significant at the 5% and 1% levels, respectively. As seen from Table X, the average final classification accuracy across all problems is better with knowledge transfer (with-KT) than without (without-KT), and their average differences are statistically significant except for the Yeast problem. The results suggest that knowledge transfer enhances the generalization performance on most problems.

Note that Fig. 8 shows the effectiveness of knowledge transfer through acceleration of learning (in terms of classification accuracy) over time, after each new training example is presented, for one of the UCI problems. Table X also reflects acceleration in learning, but after a certain time, because this accuracy is measured after a fixed number of training examples are presented.

2) Evaluation of Reorganization: An important feature of the proposed algorithm is the dynamic reorganization of classifiers to improve task categorization; that is, reorganization to properly allocate classes to tasks. If enough training examples for all of the classes are provided during the first epoch of a new task, it is expected that the system can construct a complete classifier for the new task without misallocation of classes to tasks. However, if only a small number of training samples are provided or training samples of all the classes are not provided in an epoch, the system might fail to detect a new task in the subsequent epoch and, as a result, classes of different tasks can get misallocated to a classifier. Dynamic reorganization of classifiers tries to correct this misallocation of classes to tasks (or classifiers), and this subsection examines the effectiveness of this procedure.

To evaluate the effectiveness of reorganization, we computed the average accuracies of final classification, task-change detection, and task categorization on 50 different task sequences for the ten MTPR problems shown in Tables III–VII. Table XI shows the results of these simulations with reorganization (with-RO) and without reorganization (without-RO) for different numbers of training examples per epoch (10, 20, 40, 60, and 80). As seen from Table XI, if training samples per epoch are less than 40, task categorization accuracy with reorganization (with-RO) is significantly better than without reorganization (without-RO) for all data sets although there is no significant difference in task-change detection accuracy. This means that the reorganization process makes up for the task-change detection failures. This is much clearer when the number of training samples per epoch is smaller. In addition, we find a significant improvement for Thyroid and Yeast data sets whose class distributions are strongly biased. Table XI also shows that the final classification accuracy with reorganization (with-RO) is almost the same as that of without reorganization (without-RO). Since the final accuracy is measured only when the task categorization succeeds, the results of without-RO reflect the accuracies only for easy cases that the reorganization is not necessary for correct task recognition. Therefore, the above fact indicates that the classification accuracies are not significantly degraded by the reorganization. Overall, these results show that the reorganization mechanism helps to correct misallocation of classes to tasks.

VI. CONCLUSION AND FUTURE WORK

We addressed the following multitask recognition problem in this paper. An artificial robot is presented with various objects and with various descriptions of those objects at different times and the task is to learn to describe those objects with an appropriate set of descriptors. So the robot has to do the following in terms of learning: 1) automatically group the various descriptors into appropriate tasks without external supervision, 2) build and train models appropriate to each task, and 3) then apply those task models to describe new objects of similar type. It is anticipated that future robots would need these kinds of algorithms to recognize and learn new tasks on their own without outside intervention. In fact, similar learning assumptions have been considered in automated mental development, which gives a new concept to the robot learning [29].

At the algorithmic level, the paper considered learning of multiclass classification tasks where no information was provided to the learning algorithm about task categories of training examples. The learning mode of the proposed algorithm is "online" where training examples for the tasks are given sequentially one after another. The learning algorithm, therefore, acquires task knowledge incrementally as training examples are given sequentially. Since classification tasks are related to each other in a multitask learning environment, the task learning algorithm is required to detect task changes automatically and utilize knowledge of previous tasks for learning new tasks. Overall, automated task recognition falls in the category of unsupervised learning since no information about task categories of training examples is ever provided to the algorithm. Thus, in this algorithm, supervised learning is performed at the lower

TABLE XI
THE ACCURACIES (IN PERCENT) OF CLASSIFICATION, TASK-CHANGE DETECTION, AND TASK CATEGORIZATION FOR THE PROPOSED MODEL WITH REORGANIZATION (WITH-RO) AND WITHOUT REORGANIZATION (WITHOUT-RO) FOR DIFFERENT NUMBERS OF TRAINING EXAMPLES PER EPOCH. ALTHOUGH TWO MTPR PROBLEMS ARE DEFINED FOR EACH DATA SET, THE AVERAGE PERFORMANCE FOR THE TWO PROBLEMS IS SHOWN DUE TO THE SPACE LIMITATION

(a) Segmentation

	without-RO			with-RO		
	Classification	Detection	Task categ.	Classification	Detection	Task categ.
10	84.3	77.7	82	84.0	79.5	100
20	84.6	99.2	92	84.4	99.4	100
40	85.1	99.2	94	85.0	99.4	100
60	85.0	99.0	97	85.0	99.1	100
80	85.6	98.2	94	85.5	98.3	97

(b) Thyroid

	without-RO			with-RO		
	Classification	Detection	Task categ.	Classification	Detection	Task categ.
10	82.8	76.8	72	82.6	78.1	95
20	82.7	97.2	75	82.7	98.4	92
40	83.8	98.5	84	83.8	98.8	96
60	84.0	99.4	92	84.2	99.5	96
80	84.2	99.5	100	84.2	99.5	100

(c) Vehicle

	without-RO			with-RO		
	Classification	Detection	Task categ.	Classification	Detection	Task categ.
10	77.8	76.9	66	77.2	78.6	97
20	78.1	98.9	88	77.8	99.5	100
40	78.8	99.3	98	78.8	99.4	98
60	79.5	99.5	99	79.5	99.6	100
80	79.9	98.9	97	79.9	99.0	99

(d) Vowel

	without-RO			with-RO		
	Classification	Detection	Task categ.	Classification	Detection	Task categ.
10	59.0	73.9	51	58.3	75.8	81
20	59.6	95.8	66	59.0	96.5	90
40	60.2	97.7	84	59.9	97.6	96
60	61.3	97.3	81	60.9	97.1	89
80	60.8	96.9	88	60.7	96.9	96

(e) Yeast

	without-RO			with-RO		
	Classification	Detection	Task categ.	Classification	Detection	Task categ.
10	52.4	75.9	9	52.9	75.8	46
20	53.4	97.9	13	52.2	98.5	58
40	52.6	97.6	40	52.5	98.3	58
60	51.9	97.9	58	52.3	97.7	66
80	52.3	97.1	53	52.5	97.4	61

level where classification problems are framed, whereas unsupervised learning is performed at the task recognition level. In "online learning," catastrophic forgetting is a serious problem. The new algorithm addressed catastrophic forgetting at both levels of learning—at the supervised learning level and at the unsupervised task recognition level. It is able to do that by combining neural classifiers and long-term memories (called RAN-LTM), which had originally been proposed by one of the authors.

Overall, the proposed algorithm has the following properties: 1) the ability to transfer knowledge from one task to another to speedup learning, 2) automated recognition of tasks, and 3) incremental, one-pass online learning. Although the first capability has been considered in prior works in machine learning, the last two capabilities are new. The last two capabilities are required in an environment where task category information is not provided explicitly during learning and almost all training samples are discarded after learning (that is, there is no long-term storage of all training examples).

The algorithm was tested on ten multitask problems—they were defined from five UCI database problems (Image Segmentation, Thyroid, Vehicle, Vowel, and Yeast). The following performance measures were used to evaluate the algorithm: a) accuracy of task-change detection, b) classification accuracy on test data, and c) task categorization accuracy. The experimental results demonstrated that the proposed algorithm can learn new tasks successfully in an incremental, online environment, and that knowledge transfer enhances both the learning speed and the final classification accuracy. In addition, it was verified that task recognition works fairly well unless the number of training examples per epoch is too small, and that classifier reorganization greatly improves task categorization accuracy for all MTPR problems even when the presentation order of class training examples is fairly biased.

This algorithm is a first step towards developing a completely automated multitask learning system for artificial robots. There still remain several problems in the current algorithm that have to be addressed in future works. The most important challenge is

complete automation of this algorithm so that an artificial robot can learn on its own without external help. For that, we have to resolve the problem of having to predefine different parameters of this algorithm, in particular, those of the RAN-LTM classifier. We plan to develop a new online version of RAN-LTM that incorporates the ideas of Roy *et al.* [18]–[21] where the algorithm itself adaptively defines the widths and centers of the RBFs and therefore gets away from the problem of presetting those parameters. In addition to this problem, we still need more robustness for the task-change detection part and the reorganization process. If the presentation order of classes is strongly biased, the possibility of misdetection of task changes would be increased. In an extreme case that only one class of a particular task is presented at every epoch, there is a high possibility that the class region of a particular task has no overlap with another class region of a subsequent task. Since the task-change detection in the proposed model relies on large prediction error, this case would cause the misdetection of a task change, and the two classes of different tasks are allocated in the same classifier. In this case, it is quite difficult for the current reorganization algorithm to recover this misallocation. Although the misallocation of classes could be resolved through the reorganization process to some extent, there is no guarantee for the proposed system to work well for any biased cases at the present state. This problem is also left for our future work. Finally, pattern recognition systems perform feature extraction when inputs have large dimensions. Therefore, for practical purposes, the proposed multitask learning model should also have the ability to extract features on an ongoing basis. This can be done with online feature extraction algorithms such as incremental principal component analysis (PCA) [12], [13] and incremental linear discriminant analysis (LDA) [14].

APPENDIX
THE RAN-LTM ALGORITHM

The RAN-LTM learning algorithm is summarized below.

[Learn RAN-LTM]

For each training example $\boldsymbol{x}_p$ $(p = 1, \ldots, P)$ in the given training set with an output $\boldsymbol{d}_p$, perform the following procedure. Here, P is the number of training examples.

Step 1: Output Calculation: Calculate the outputs $\boldsymbol{z}(\boldsymbol{x}_p)$ from (1) and (2).

Step 2: Network Learning: Calculate the squared error $E(\boldsymbol{x}_p) = \|\boldsymbol{d}_p - \boldsymbol{z}(\boldsymbol{x}_p)\|^2$. If $E(\boldsymbol{x}_p) > \varepsilon$ and $\|\boldsymbol{x}_p - \boldsymbol{c}^*\| > \delta$ ($\boldsymbol{c}^*$: the nearest center to $\boldsymbol{x}_p$), then the following procedure is carried out.

1) **Hidden Unit Allocation**: Add a hidden unit (i.e., $J \leftarrow J + 1$). For this hidden unit, the center vector $\boldsymbol{c}_J$ is set to $\boldsymbol{c}_J = \boldsymbol{x}_p$ and the connection weight w_{kJ} from the Jth hidden unit to the kth output unit are set to $w_{kJ} = d_{pk} - z_k(\boldsymbol{x}_p)$.
2) **Memory Item Creation**: If $\min_l \|\boldsymbol{x}_p - \tilde{\boldsymbol{x}}^{(l)}\| > \delta$, increment the number of memory items: $L \leftarrow L + 1$ and set the Lth memory item as follows: $(\tilde{\boldsymbol{x}}^{(L)}, \tilde{\boldsymbol{d}}^{(L)}, a) = (\boldsymbol{x}_p, \boldsymbol{d}_p, 1)$. Then, store this memory item into LTM.

Otherwise, the following procedure is carried out:

1) **Memory Item Retrieval**: Find all *active* memory items $(\tilde{\boldsymbol{x}}^{(l)}, \tilde{\boldsymbol{d}}^{(l)}, a = 1)$ where $\boldsymbol{x}_p \approx \tilde{\boldsymbol{x}}^{(l)}$. Define an index set $\mathcal{M}$ for these memory items.
2) **Output Calculation**: For the training example $(\boldsymbol{x}_p, \boldsymbol{d}_p)$ and all the retrieved active memory items $(\tilde{\boldsymbol{x}}^{(l)}, \tilde{\boldsymbol{d}}^{(l)}, a = 1)$ $(l \in \mathcal{M})$, calculate RAN's outputs $\boldsymbol{z}(\boldsymbol{x}_p)$ and $\boldsymbol{z}(\tilde{\boldsymbol{x}}^{(l)})$, and calculate the total squared error E in (3).
3) **Modification**: Update the weight connection w_{kj}, RBF center c_{ji}, and bias ξ_k as follows:

$$w_{kj}^{\mathrm{NEW}} = w_{kj}^{\mathrm{OLD}} + \alpha \tilde{e}_k y_j$$

$$c_{ji}^{\mathrm{NEW}} = c_{ji}^{\mathrm{OLD}} + \frac{\alpha}{\sigma^2}(x_i - c_{ji}^{\mathrm{OLD}}) y_j \sum_{k=1}^{K} \tilde{e}_k w_{kj}$$

$$\xi_k^{\mathrm{NEW}} = \xi_k^{\mathrm{OLD}} + \alpha \tilde{e}_k$$

where

$$\tilde{e}_k = (d_k - z_k(\boldsymbol{x}_p)) + \sum_{l \in \mathcal{M}} \left(\tilde{d}_k^{(l)} - z_k(\tilde{\boldsymbol{x}}^{(l)}) \right).$$

Step 3: Reevaluation and Retraining: Calculate the square error $E(\boldsymbol{x}_p) = \|\boldsymbol{d}_p - \boldsymbol{z}(\boldsymbol{x}_p)\|^2$ again. If $E(\boldsymbol{x}_p) > \varepsilon$, perform *Hidden Unit Allocation* and *Memory Item Creation* in Step 2.

ACKNOWLEDGMENT

The authors would like to thank Prof. S. Abe, Kobe University, Japan, and the anonymous reviewers for helpful discussions and useful suggestions.

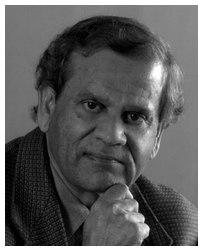
REFERENCES

[1] Y. S. Abu-Mostafa, "Learning from hints in neural networks," *J. Complexity*, vol. 6, no. 2, pp. 192–198, 1989.
[2] B. Bakker and T. Heskes, "Task clustering and gating for Bayesian multitask learning," *J. Mach. Learn. Res.*, vol. 4, pp. 83–99, 2003.
[3] J. Baxter, "A Bayesian/information theoretic model of learning to learn via multiple task sampling," *Mach. Learn.*, vol. 28, no. 1, pp. 7–39, 1997.
[4] G. A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *IEEE Computer*, vol. 21, no. 3, pp. 77–88, Mar. 1988.
[5] R. Caruana, "Learning many related tasks at the same time with backpropagation," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995, vol. 7, pp. 657–664.
[6] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 1, pp. 41–75, 1997.
[7] R. Caruana and J. O'Sullivan, "Multitask pattern recognition for autonomous robots," in *Proc. IEEE Int. Conf. Intell. Robot. Syst.*, 1998, vol. 1, pp. 13–18.
[8] S. Ben-David, J. Gehrke, and D. Kifer, "Detecting change in data streams," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2004, pp. 180–191.
[9] J. Ghosn and Y. Bengio, "Bias learning, knowledge sharing," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 748–765, Jul. 2003.
[10] N. Kasabov, *Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines*. New York: Springer-Verlag, 2002.
[11] M. Kobayashi, A. Zamani, S. Ozawa, and S. Abe, "Reducing computations in incremental learning for feedforward neural network with long-term memory," in *Proc. Int. Joint Conf. Neural Netw.*, 2001, vol. 3, pp. 1989–1994.

[12] S. Ozawa, S. L. Toh, S. Abe, S. Pang, and N. Kasabov, "Incremental learning of feature space and classifier for face recognition," *Neural Netw.*, vol. 18, no. 5-6, pp. 575–584, 2005.
[13] S. Ozawa, S. Pang, and N. Kasabov, "Incremental learning of chunk data for online pattern classification systems," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 1061–1074, Jun. 2008.
[14] S. Pang, S. Ozawa, and N. Kasabov, "Incremental linear discriminant analysis for classification of data streams," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 35, no. 5, pp. 905–914, Oct. 2005.
[15] S. Parveen and P. Green, "Multitask learning in connectionist robust ASR using recurrent neural networks," in *Proc. EUROSPEECH*, 2003, pp. 1813–1816.
[16] J. Platt, "A resource allocating network for function interpolation," *Neural Comput.*, vol. 3, pp. 213–225, 1991.
[17] L. Y. Pratt, J. Mostow, and C. A. Kamm, "Direct transfer of learned information among neural networks," in *Proc. 9th Nat. Conf. Artif. Intell.*, 1991, pp. 584–589.
[18] A. Roy, L. S. Kim, and S. Mukhopadhyay, "A polynomial time algorithm for the construction and training of a class of multilayer perceptrons," *Neural Netw.*, vol. 6, no. 4, pp. 535–545, 1993.
[19] A. Roy, S. Govil, and R. Miranda, "An algorithm to generate radial basis function (RBF)-like nets for classification problems," *Neural Netw.*, vol. 8, no. 2, pp. 179–202, 1995.
[20] A. Roy and S. Mukhopadhyay, "Iterative generation of higher-order nets in polynomial time using linear programming," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 402–412, Mar. 1997.
[21] A. Roy, S. Govil, and R. Miranda, "A neural network learning theory and a polynomial time RBF algorithm," *IEEE Trans. Neural Netw.*, vol. 8, no. 6, pp. 1301–1313, Nov. 1997.
[22] D. Silver and R. Mercer, "Selective functional transfer: Inductive bias from related tasks," in *Proc. IASTED Int. Conf. Artif. Intell. Soft Comput.*, 2001, pp. 182–189.
[23] D. Silver and R. Mercer, "The task rehearsal method of life-long learning: Overcoming impoverished data," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2002, vol. 2338, pp. 90–101.
[24] D. Silver and R. Poirier, *The Sequential Consolidation of Learned Task Knowledge*, ser. Lecture Notes in Artificial Intelligence, A. Y. Tawfik and S. D. Goodwin, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 217–232.
[25] S. Thrun and T. Mitchell, "Learning one more thing," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CS-94-184, 1994.
[26] S. Thrun and L. Pratt, *Learning to Learn*. Norwell, MA: Kluwer, 1998.
[27] K. Tsumori and S. Ozawa, "Incremental learning in dynamic environments using neural network with long-term memory," in *Proc. Int. Joint Conf. Neural Netw.*, 2003, pp. 2583–2588.
[28] Univ. California at Irvine, UCI Machine Learning Repository, Irvine, CA [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
[29] 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, pp. 599–600, 2001.

Seiichi Ozawa (M'02) received the B.E. and M.E. degrees in instrumentation engineering and the Ph.D. degree in computer science from Kobe University, Kobe, Japan, in 1987, 1989, and 1998, respectively.

Currently, he is an Associate Professor with Graduate School of Engineering, Kobe University. From 2005 to 2006, he was a visiting researcher at Arizona State University, Tempe, and this work was mainly conducted during the visit. His primary research interests include neural networks, machine learning, intelligent data processing, and pattern recognition.



Asim Roy (S'78–M'82–SM'89) received the M.S. degree in operations research from Case Western Reserve University, Cleveland, OH, in 1977 and the Ph.D. degree in operations research from University of Texas at Austin, in 1979.

Currently, he is a Professor of Information Systems at Arizona State University, Tempe. He has been a Visiting Scholar at Stanford University, visiting Prof. David Rumelhart in the Psychology Department, and a Visiting Scientist at the Robotics and Intelligent Systems Group at Oak Ridge National Laboratory, Oak Ridge, TN. His research has been published in *Management Science*, *Mathematical Programming*, *Neural Networks*, *Neural Computation*, various IEEE Transactions (IEEE TRANSACTIONS ON NEURAL NETWORKS, IEEE TRANSACTIONS ON FUZZY SYSTEMS, IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS) and other journals. He has been invited to many national and international conferences for plenary talks and for tutorials, workshops and short courses on his new learning theory and methods. His research interests are in brain-like learning, neural networks, machine learning, data mining, intelligent systems, and nonlinear multiple objective optimization.

Dmitri Roussinov received the diploma with honors from the Moscow Institute of Physics and Technology, Moscow, Russia, in 1991, the M.S. degree in economics from Indiana University, in 1995, and the Ph.D. degree in management information systems from the University of Arizona, Tucson, in 1999.

He holds a Senior Lecturer position in the Department of Computer and Information Sciences, University of Strathclyde, Glasgow, Scotland. He was involved in the study described here when he was an Assistant Professor in the Department of Information Systems, W.P. Carey School of Business, Arizona State University, Tempe. He is interested in applications of neural networks and artificial intelligence in general to information retrieval, text mining, web search engines, and automated question answering. He has published in various IEEE and ACM transactions and conference proceedings, as well as in the leading business information systems outlets.