

Paper:

# An Incremental Neural Network for Online Supervised Learning and Topology Learning

Youki Kamiya<sup>\*</sup>, Shen Furao<sup>\*\*</sup>, and Osamu Hasegawa<sup>\*\*,\*\*\*</sup>

<sup>\*</sup>Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology

R2-52, 4259 Nagatsuta, Midori-ku, Yokohama 226-8503, Japan

<sup>\*\*</sup>Imaging Science and Engineering Lab., Tokyo Institute of Technology

<sup>\*\*\*</sup>PRESTO, Japan Science and Technology Agency (JST)

E-mail: {youki, furaoshen, hasegawa}@isl.titech.ac.jp

[Received January 30, 2006; accepted May 19, 2006]

**A new self-organizing incremental network is designed for online supervised learning. During learning of the network, an adaptive similarity threshold is used to judge if new nodes are needed when online training data are introduced into the system. Nodes caused by noise are deleted to decrease the misclassification. The proposed network, which is robust to noisy training data, suits the following tasks: (1) online or even life-long supervised learning; (2) incremental learning, i.e., learning new information without destroying old learned information; (3) learning without any predefined optimal condition; (4) representing the topology structure of inputting online data; and (5) learning the number of nodes needed to represent every class. Experiments of artificial data and high-dimension real-world data show that the proposed method can achieve classification with a high recognition ratio, high speed, and low memory.**

**Keywords:** self-organizing, incremental, online supervised learning, topology learning

## 1. Introduction

Supervised learning is also referred to as discriminant analysis or supervised grouping. In supervised learning, a classifier is used to discriminate between members and non-members of a given class.

Many methods are suitable for supervised learning. Support Vector Machines (SVMs) produce stable and high-quality output [1], but they are strongly dependent on the selection of a kernel function [2]; ADA Boosting [3] requires advance knowledge of the base classifier. For classification tree methods such as CART [4], splits are increasingly created as the tree expands, and interpretation would necessarily be more complex. Linear Discriminant Analysis (LDA) [5] maximizes the ratio of between-class variance to the within-class variance in any particular dataset, thereby guaranteeing maximal separability, but it is only suitable to linear discriminant tasks. Learning vector quantization (LVQ) [6] is adopted

in a supervised learning technique for the self-organizing map (SOM) [7], but it requires advance definition of the number of prototype vectors for classes; furthermore, the number is the same for every class. Artificial neural networks (ANN) [8] are efficient for some supervised learning tasks. However, if we add one new class to the system, the whole network must be retrained. For that reason, ANN is unsuitable for incremental learning tasks, which are an important aspect of many applications.

The nearest neighbor [9] algorithm differs from other classification methods in that it does not build a classifier from the training data, but uses the original training vector in the discriminant process. Salient advantages of nearest neighbor are that: it can learn from a small set of examples; it can add new information incrementally at runtime; it requires no optimization, it is capable of modeling very complex target functions using a collection of less complex approximations, and it is robust to noisy training data. On the other hand, its major disadvantage is the high complexity of recognition for large datasets. Shibata et al. [10] proposed a K-D decision tree method to speed up the nearest neighbor algorithm. However, that method and other traditional methods such as condensing [11] and editing [12] techniques were unsuitable for incremental learning.

In [13], an incremental network for unsupervised learning that extended Growing Neural Gas (GNG) [14] was proposed and used to process some face recognition and vector quantization tasks. This method can judge whether new nodes are needed when online training data come into the system. The method can delete nodes caused by noise to thereby decrease misclassification [15]. In this paper, we adjust the network in [13] and design a self-organizing incremental network for online incremental supervised learning tasks. First, we use the online training samples to train the network corresponding to each class and output the nodes of the networks. Then, such nodes are used as prototype vectors of the different classes to assign test data to different classes. This method is suitable for incremental learning: it can learn new nodes as prototypes for new classes easily without destroying old learned information. In light of the nearest neighbor advantages described above, we adopt the nearest neighbor method as

the main benchmark for comparison with the proposed method.

## 2. Proposed Method

The targets of the proposed algorithm are: (1) Without prior conditions such as number of nodes or a good network structure or a kernel function to conduct online supervised learning, and represent the topological structure of input probability density. (2) To learn new information without destroying old learned information. (3) To learn the number of nodes needed to represent every class to avoid catastrophic allocation of new nodes. (4) To delete nodes caused by noise and thereby decrease misclassification.

### 2.1. Overview of the Proposed Method

In this study, we adopt a self-organizing incremental neural network to realize our targets. Online training data are used to train the network for each class and generate a topological structure of the input pattern. Then the learned nodes of the networks are used as the prototype vectors to classify the test data.

To represent the topological structure in online or life-long learning tasks, growth is an important feature for decreasing task error and adapting to changing environments while preserving old prototype patterns. Insertion of new nodes is extremely useful to grow the network without interfering with previous learned knowledge. However, insertion must be stopped to prohibit a permanent increase in the number of nodes and to avoid overfitting.

We use a similarity threshold for every old node to judge if a new input pattern originates from a learned region or from a region that never happened. The similarity threshold  $T_i$  is defined as the distance (Euclidean distance) from the boundary to the center of Voronoi region  $V_i$  of node  $i$ . The Voronoi diagram is the nearest-neighbor map for a set of points. In this method, the set of points means the network. Each Voronoi region contains those points that are nearer one node than any other nodes. During the learning process, node  $i$  must change its position to meet the input pattern distribution. Therefore, the Voronoi region  $V_i$  of the node  $i$  will change and the similarity threshold  $T_i$  will also change. The similarity threshold is changed adaptively according to the change of position of node  $i$ . We set the input signal as a new node (the input signal comes from a never-happened area) when the distance between the signal and the nearest node (node  $k$ ) is greater than the similarity threshold  $T_k$  of node  $k$ . After some learning steps, we reject the insertion of the new node in the well learned area because the distance between the input and the nearest node is less than the similarity threshold  $T_j$  of node  $j$  (the input signal comes from the area where nodes are sufficiently numerous).

To build connections between neural nodes, we adopt the competitive Hebbian rule [16]. The competitive Hebbian rule can be described as follows: for each input sig-

nal, connect the two closest nodes (measured using Euclidean distance) by an edge. In online or life-long learning, the nodes change their locations slowly but permanently; nodes that are neighboring at an early stage might not be neighboring at a more advanced stage. Therefore, the competitive Hebbian rule removes connections that have not been refreshed for a while.

In general, noise exists in real world data. To delete the nodes caused by noise and thereby reduce misidentification, we propose the following strategy: if the number of generated input signals up to that time is an integer multiple of a parameter, remove those nodes with no topological neighbor or which have only one. The idea is based on the behavior of adding and removing edges from the network. We adopt the competitive Hebbian rule that connects the two closest nodes for each input signal. Thereby, nodes having many edges produce a network in the area where many input signals come. After numerous input signals, if the node has no neighbor or only one, then during a period, this node has a very low chance to be selected as one of the two closest nodes and the insertion of new nodes near this node is difficult. As a result, we infer that the probability that the node is caused by noise is very high.

### 2.2. Complete Algorithm

Along with the analysis of Section 2.1, we present the complete algorithm here.

*Notations to be used in the algorithm*

$A$	Node set, used to store nodes.
$C$	Connection set (or edge set), used to store connections (edges) between nodes.
$W_i$	$n$ -dimension weight vector of node $i$ .
$T_i$	Similarity threshold. If the distance between an input pattern and node $i$ is greater than $T_i$ , the input pattern is a new node.
$N_i$	Set of direct topological neighbors of node $i$ . If a node links with node $i$ by an edge, we say the node is the neighbor of node $i$ .
$M_i$	Local accumulated number of signals of node $i$ .
$age_{(i,j)}$	Age of the edge that connects node $i$ and node $j$ .

*Algorithm 2.1:* The proposed algorithm for generating a self-organizing incremental network

1. Initialize node set  $A$  to contain two nodes,  $c_1$  and  $c_2$  with weight vectors selected randomly from the input pattern. Initialize connection set  $C$ ,  $C \subset A \times A$ , to the empty set.
2. Input new pattern  $\xi \in R^n$ .
3. Search the nearest node (winner)  $s_1$ , and the second-nearest node (second winner)  $s_2$  by

$$s_1 = \arg \min_{c \in A} \|\xi - W_c\| \quad . . . . . (1)$$

$$s_2 = \arg \min_{c \in A \setminus \{s_1\}} \|\xi - W_c\| \quad . . . . . (2)$$