How to Use the SOINN Software: User's Guide (Version 1.0)

Kazuhiro Yamasaki* Naoya Makibuchi† Furao Shen‡
Osamu Hasegawa§

October 15, 2010

Abstract

The Self-Organizing Incremental Neural Network (SOINN) is an unsupervised classifier that is capable of online incremental learning. Studies have been performed not only for improving the SOINN, but also for applying it to various problems. Furthermore, using the SOINN, more intelligent functions are achieved, such as association, reasoning, and so on. In this paper, we show how to use the SOINN software and to apply it to the above problems.

Key Words: Self-Organizing Incremental Neural Network, SOINN software

1 Introduction

The Self-Organizing Incremental Neural Network (SOINN) [1] is an online unsupervised mechanism proposed by Shen and Hasegawa which is capable of incremental learning, that is, it can learn new knowledge without destroying the old learned knowledge. Because the neurons in the network are self-organized, it is not necessary to define the network structure and size in advance. In addition, this system is robust to noise.

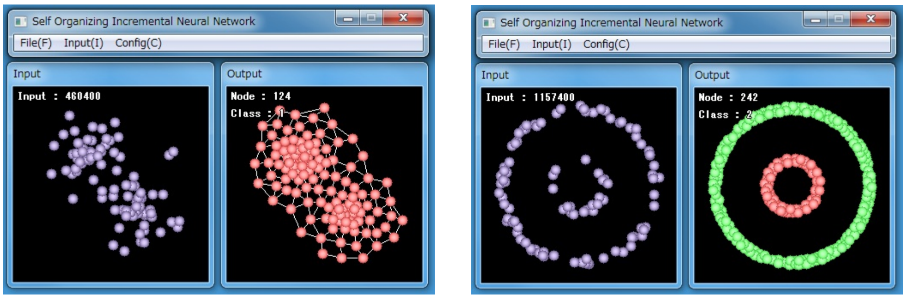
Studies have been performed not only for improving the SOINN, but also for applying it to various problems. For example, an Enhanced-SOINN (ESOINN) [2] has succeeded in reducing the number of parameters and layers of the original SOINN from two layers and eight parameters to one layer and four parameters. Furthermore, it is capable of separating clusters with a high-density overlap. An Adjusted SOINN Classifier (ASC) [3] automatically learns the number of prototypes needed to determine the decision boundary;, thus, very rapid classification is achieved.

*Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, `yamasaki.k.ac@m.titech.ac.jp`

†Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, `makibuchi.n.aa@m.titech.ac.jp`

‡The State Key Laboratory for Novel Software Technology, Nanjing University, `frshen@nju.edu.cn`

§Imaging Science and Engineering Laboratory, Tokyo Institute of Technology, `hasegawa.o.aa@m.titech.ac.jp`, `http://www.haselab.info/`



(a) Input : Gaussian (b) Input : Concentric Circle

Figure 1: Execution examples.

Using the SOINN, more intelligent functions can be achieved, such as association, reasoning, and so on. An associative memory (AM) system using the SOINN has been proposed [4], which is called the SOIAM. This system learns a pair of vectors (a key vector and an associative vector), which makes it possible to realize the association between them. On the other hand, a novel AM system consisting of a three-layer architecture has also been proposed [5]. This system realizes the association of both static information and temporal sequence information. In addition, a pattern-based reasoning system using the SOINN has been also proposed [6], which achieves reasoning with the pattern-based if-then rules of propositional logic.

The SOINN is also applied to fields such as robotics (e.g., language acquisition [7, 8], task planning [9, 10], the SLAM system [11], and robot navigation [12]).

The SOINN software is available here[1]. We provide an application and the source code written in the C++ language of the SOINN with a single layer and two parameters, which has been introduced by [3] and employed in [4, 6]. In this paper, we show how to use this software, and then describe briefly how to extend the source code into one of [4] or [6].

2 Application

You can acquire the SOINN software as a solution file of Microsoft Visual Studio 2005. To run the SOINN application, open and build the "SOINN.sln" file, then execute the resulting program. In doing so, windows containing the menu, the Input, and the Output are displayed, as shown in Figure 1. The menu window provides the functions for the designation of the data set, the definition of parameters and noise, and so on. The Input window shows how the data set is input continuously, specifically online, as well as the current number of input data items. The Output window shows how the network grows in a self-organizational way with each input data item, and the current number of nodes and classes in the network.

If you select "Input > Synthetic data > Gaussian or Concentric circle," data generated from the two gaussian or concentric circles is input into the network

[1] `http://www.haselab.info/soinn-e.html`

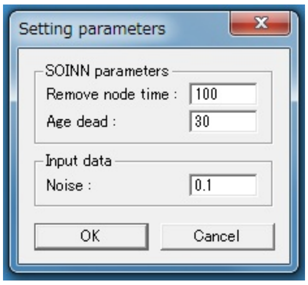


Figure 2: Screen of Setting parameters.

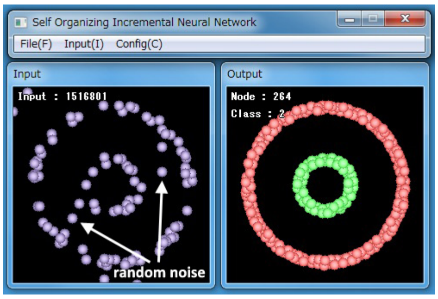


Figure 3: Data set including noise.

over time. The SOINN displays the topological structure of the input data, so that it regards a set of all nodes connected with edges as a cluster. This makes it possible to classify unsupervised data. For example, Figure 1(b) shows the classified data of each circle using different colors. At the same time, the SOINN shows the probabilistic density of the input data. For example, in Figure 1(a), it is found that nodes on the area around the center of the gaussian are distributed in high density and vice versa. In addition, the SOINN realizes the compression of the input data by finding typical prototypes for a large-scale data set. In fact, in Figures 1(a) and 1(b), the numbers of input data are 460,400 and 1,157,400, and the numbers of output data are 124 and 242.

The parameters necessary for the SOINN can be set by selecting "Config > Setting parameters," as shown in Figure 2. In this window, not only the SOINN parameters, but also the amount of noise added to the data set, can be defined. For example, as shown in Figure 2, if "Noise" is defined as 0.1, 10% random noise is added to the data set. However, as shown in Figure 3, it is found that the SOINN adequately displays the distribution of the original data set without being affected by noise.

3 SOINN API

A brief class diagram is depicted in Figure 4. The SOINN API consists of six core files, `CSOINN.cpp/.h`, `CNode.cpp/.h`, and `CEdge.cpp/.h`. The class `CSOINN` represents the entire SOINN system. The `CNode` is the node (i.e., neuron) in the network. You can retrieve the information of the node using the API functions of this class. The `CEdge` is the edge between the two nodes.

Following are explanations of the SOINN API functions:

- `CSOINN::InputSignal()` represents the algorithm[2] of the SOINN for one given input data.
- `CSOINN::Classify()` assigns the class ID to all the nodes according to the current network structure.

[2]This function includes processes such as the addition or deletion of nodes.

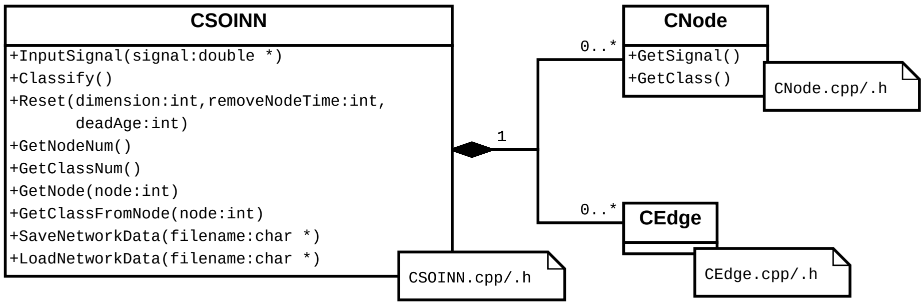


Figure 4: Class diagram of the SOINN software.

- `CSOINN::Reset()` resets the parameters of the SOINN (i.e., dimension, removeNodeTime, and deadAge). Note that, in the current implementation, all the nodes and edges in the network are also removed.
- `CSOINN::GetNodeNum()` returns the number of nodes in the current network.
- `CSOINN::GetClassNum()` returns the number of classes in the current network.
- `CSOINN::GetNode()` returns the `CNode` instance with the same ID[3] as the function's argument.
- `CSOINN::GetClassFromNode()` returns the class ID of the `CNode` instance with the same ID as the function's argument.
- `CSOINN::SaveNetworkData()` saves the current network to a given file.
- `CSOINN::LoadNetworkData()` loads any network from a given file.
- `CNode::GetSignal()` returns the weight vector of the `CNode` instance itself.
- `CNode::GetClass()` returns the class ID of the `CNode` instance itself.

4 Usage Example

In this section, we show how to use the API of the SOINN software through the sample code (Listing 1) of the 1-Nearest Neighbor (1-NN) algorithm. Note that it is necessary to define the functions `Distance()` and `LoadSignal()`.

First, to use the SOINN, call the `CSOINN::CSOINN()` function. This constructor returns the `CSOINN` instance. Next, to perform the learning process of the SOINN, call the `CSOINN::InputSignal()` function. The argument of this function represents one training data vector. You then call the `CSOINN::Classify()` function to assign all nodes in the current network with the class ID.

Next, to run the 1-NN algorithm, calculate the distance between the test data and the weight vector of each of the nodes in the current network, where each

[3]The SOINN software assigns an ID to all the nodes in the current network.

weight vector is returned by the `CNode::GetSignal()` function. Finally, use the functions `CNode::GetClass()` or `CSOINN::GetClassFromNode()` to retrieve the class ID.

Listing 1: Sample code of the 1-NN using the SOINN API.

```
// DIMENSION : the number of dimension of the input vector
// REMOVE_NODE_TIME : the predefined threshold
//                    to remove edges with an old age
// DEAD_AGE : the predefined parameter
//            to delete nodes having only one neighbor
// TRAINING_DATA_FILE : the name of training data file
// TEST_DATA_FILE : the name of test data file
// *signal : the training data vector
// *targetSignal : the test data vector
//
// Distance() : the distance function (typically Euclidean distance)
// LoadSignal() : the function to get one training/test data vector

CSOINN *pSOINN;
pSOINN = new CSOINN( DIMENSION, REMOVE_NODE_TIME, DEAD_AGE );

double *signal;
while ( ( signal = LoadSignal( TRAINING_DATA_FILE ) ) != NULL ) {
  pSOINN->InputSignal( signal );
}
pSOINN->Classify();

// 1-NN algorithm.
double minDist       = CSOINN::INFINITY;
int    nearestID     = CSOINN::NOT_FOUND;

double *targetSignal = LoadSignal( TEST_DATA_FILE );
for ( int i = 0; i < pSOINN->GetNodeNum(); i++ ) {

  double *nodeSignal = pSOINN->GetNode( i )->GetSignal();
  double dist        = Distance( targetSignal, nodeSignal );

  if ( minDist > dist ) {
    minDist   = dist;
    nearestID = i;
  }

}

int   nearestClassID = pSOINN->GetNode( nearestID )->GetClass();
printf( "Nearest Node ID : %d, Class ID : %d.\n", nearestID, nearestClassID );

delete pSOINN;
```

5 Extensions

In this section, we briefly describe how to extend the SOINN to [2, 3, 4, 5, 6]. In the SOIAM [4], the dimension of the association pair must be defined in `CSOINN` and `CNode`. Furthermore, it is necessary to define an appropriate distance function, a recall function, and so on. Listing 2 shows the sample code of the additional implementation for the SOIAM. In the other AM system consisting of a three-layer architecture [5], the new class representing an input layer, a memory layer, and an associative layer must be defined. In a pattern-based reasoning system [6], as well as in the SOIAM, some functions, such as a distance function, also need to be defined. In the ESOINN [2], it is necessary to modify the `CSOINN::InputSignal()` function to enable this system to separate clusters with high-density overlap. In addition, two new parameters to determine the noise node must be added in `CSOINN`. In the ASC [3], the multiple `CSOINN` instances that correspond to the classes of the data set are required, and the new class to perform the *k*-means algorithm for the stabilization of the ASC must be defined. The functions to reduce noisy and unnecessary prototype nodes must also be defined.

Listing 2: Sample code of the additional implementation of the SOIAM.

```
class CSOIAM : public CSOINN {

  public :
    std::vector<CNode *> *Recall( double *signal, bool isDirect );

  private :
    int keyDim;
    int assocDim;

    double Distance( double *signal1, double *signal2, bool isDirect );

}
/* =============================================== */
std::vector<CNode *> *CSOIAM::Recall( double *signal, bool isDirect )
{
  int classNum = GetClassNum();
  int nodeNum  = GetNodeNum();

  std::vector<std::pair<double, int>> minPair( classNum );
  for ( int i = 0; i < nodeNum; i++ ) {

    CNode  *node    = GetNode( i );
    int    classID  = node->GetClass();

    double distance = Distance( signal, node->GetSignal(), isDirect );
    if ( minPair[classID].first > distance ) {
      minPair[classID].first  = distance;
      minPair[classID].second = i;
    }

  }

  std::vector<CNode *> *recalledNodes = new std::vector<CNode *>( classNum );
  for ( int i = 0; i < classNum; i++ ) {
    recalledNodes->at( i ) = GetNode( minPair[i].second );
  }

  return recalledNodes;

}
/* =============================================== */
double Distance( double *signal1, double *signal2, bool isDirect )
{
  int beginPtr  = ( isDirect ) ? 0 : keyDime;
  int endPtr    = ( isDirect ) ? keyDim : ( keyDim + assocDim );
  int normConst = ( isDirect ) ? keyDim : assocDim;

  double distance  = 0.0;
  for ( int i = beginPtr; i < endPtr; i++ ) {
    double tmp = ( signal1[i] - signal2[i] );
    distance += ( tmp * tmp );
  }
  distance = sqrt( distance / normConst );

  return distance;

}
```

Acknowledgement

This study was supported by the Industrial Technology Research Grant Program in 2010 from the New Energy and Industrial Technology Development Organization (NEDO) of Japan.

References

[1] Shen, F., Hasegawa, O.: An incremental network for on-line unsupervised classification and topology learning. Neural Networks, vol.19, No.1, pp.90–106 (2005)

[2] Shen, F., Ogura, T., Hasegawa, O.: An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Networks, vol.20, No.8, pp.893–903 (2007)

[3] Shen, F., Hasegawa, O.: A fast nearest neighbor classifier based on self-organizing incremental neural network. Neural Networks, vol.21, No.10, pp.1537–1547 (2008)

[4] Sudo, A., Sato, A., Hasegawa, O.: Associative memory for online learning in noisy environments using self-organizing incremental neural network. IEEE Trans. on Neural Networks, vol.20, No.6, pp.964–972 (2009)

[5] Shen, F., Yu, H., Kasai, W., Hasegawa, O.: An Associative Memory System for Incremental Learning and Temporal Sequence. Proc. of the 2010 International Joint Conference on Neural Networks. to appear (2010)

[6] Shen, F., Sudo, A., Hasegawa, O.: An online incremental learning pattern-based reasoning system. Neural Networks, vol.23, No.1, pp.135–143 (2010)

[7] He, X., Kojima, R., Hasegawa, O.: Developmental Word Grounding through A Growing Neural Network with A Humanoid Robot. IEEE Trans. SMC-Part B, vol.37, No.2, pp.451–462 (2007)

[8] He, X., Ogura, T., Satou, A., Hasegawa, O.: Developmental Word Acquisition And Grammar Learning by Humanoid Robots through A Self-Organizing Incremental Neural Network. IEEE Trans. SMC-Part B, vol.37, No.5, pp.1357–1372 (2007)

[9] Makibuchi, N., Shen, F., Hasegawa, O.: Online Knowledge Acquisition and General Problem Solving in a Real World by Humanoid Robots. 1st SOINN Workshop, in conjunction with ICANN, pp.551–556 (2010)

[10] Makibuchi, N., Shen, F., Hasegawa, O.: General Problem Solving System in a Real World and Its Implementation on a Humanoid Robot, IEICE Trans. Inf. & Sys. [in Japanese], volJ93-D, No.6, pp.960–977 (2010)

[11] Kawewong, A., Honda, Y., Tsuboyama, M., Hasegawa, O.: Reasoning on the Self-Organizing Incremental Associative Memory for Online Robot Path Planning. IEICE Trans. Inf. & Sys., vol.E93-D, No.3, pp.569–582 (2010)

[12] Tangruamsub, S., Tsuboyama, M., Kawewong, A., Hasegawa, O.: Unguided Robot Navigation using Continuous Action Space. 1st SOINN Workshop, in conjunction with ICANN, pp.528–534 (2010)