Abstract

This paper presents a novel growing self-organizing algorithm for dynamic clustering. Controlling a signal counter of each cell, the network can grow. Also, if there exists undesired cell for the clustering, the cell can be deleted virtually. Our algorithm can reinforce the clustering function and the learned network can adapt flexibly to time-variant input space.

1 Introduction

In this paper, we consider a novel unsupervised learning algorithm for neural networks having dynamic clustering function. As a typical unsupervised learning system, Kohonen’s Self-Organizing Map (SOM) is known. The SOM is simple and can extract characteristics of the input space. The SOM has been applied to speech recognition, image compression, data mining, and so on [1]-[4]. However, for the clustering problems, the SOM has some bottlenecks: the network are distorted for complicated inputs and the learning algorithm can not adapt to time-variant input space.

In this paper, we present a novel growing self-organizing algorithm for dynamic clustering. The algorithm has the following important basic functions:

• Classification of the input space without class information before the learning.

• Flexible adaptation to time-variant input space.

The algorithm has the following properties.

• The network has a growing cell structures [5]: a novel cell is inserted depending on a value of a signal counter [6]. Hereby initial setting of the network size is not needed.

• The counter also controls the learning rate and we can avoid distortion of the network in the learning process.

• If the counter has lower value than a threshold, the cell is deleted virtually: it contribute to reinforce the clustering function.

Our algorithm can realize a mapping with efficient clustering function that can adapt flexibly time-variant input space. The efficiency is confirmed in fundamental numerical experiments.

2 Learning algorithm

Figure 1 shows basic network structures of SOM where \( A \) denote discrete space, \( X \) denote input space. Our network has 1-dimensional topology and each cell \( i \) is connected to the closest neighbor \( N_c \). The cell \( i \) is characterized by a 2-dimensional weight vector \( w_i \) and a counter of the input. The counter \( C_i \) is memory of input signals. The outline of our learning algorithm is as the following.

Step 1 : Initialization

Let \( t \) denote discrete time, \( k \) be an initial number of...
cells, and let 2-dimensional weight vectors \( w_i (i = 1 \sim k) \) be initialized randomly.

**Step 2 : Input space**

Apply input vector \( x \) according to some distribution.

**Step 3 : Determination of winner cell**

Let \( c \) be a winner and let \( w_c \) be a corresponding weight vector. The winner is selected by

\[
\| x(t) - w_c(t) \| = \min_{i \in A} \| x(t) - w_i(t) \| \quad (1)
\]

where \( \| \cdot \| \) denote the Euclidean distance.

**Step 4: Update of the counters**

The counter \( C_i \) memorizes the winning number of cell \( i \). The counter is updated according to Equation (2).

\[
C_i(t+1) = \begin{cases} 
C_i(t)+1 & \text{if } i = c \\
C_i(t) - 0.005 & \text{if } N_c \\
C_i(t) - 0.007 & \text{Otherwise}
\end{cases} \quad (2)
\]

Note that the counter decreases for neighbors and the others. Then if the \( i \)-th cell is out of the input space, its counter \( C_i \) must decrease.

**Step 5 : Update of weight vectors**

If \( C_i > Th \), the cells are updated as the following:

\[
w_i(t+1) = \begin{cases} 
\beta_c(x(t) - w_i(t)) & \text{if } i = c \\
\beta_n(x(t) - w_i(t)) & i \in N_c \\
w_i(t) & \text{Otherwise},
\end{cases} \quad (3)
\]

If \( C_i < Th \), the cells are updated as the following:

\[
w_i(t+1) = \begin{cases} 
\beta_c(x(t) - w_i(t)) & \text{if } i = c \\
w_i(t) & \text{Otherwise},
\end{cases} \quad (4)
\]

where \( \beta_c \) and \( \beta_n \) denote learning late parameters.

**Step 6 : Insertion of a cell**

Figure 2 shows an insertion method of a cell. At every \( M \) step, we find the cell \( q \) with the maximum counter value: \( C_q = \max C_i(i \in \{1, \ldots, n\}) \). We also find the farthest direct neighbor cell \( f \). New cell \( r \) is inserted between cells \( q \) and \( f \) with the following conditions:

\[
d = \| w_f(t) - w_{i+1}(t) \|, \quad (5)
\]

\[
a < d < b, \quad (6)
\]

\[
w_r = 0.5(w_q + w_f), \quad (7)
\]

where \( a \) and \( b \) denote parameters. Also, the counter value changes as the following:

\[
C_r(t+1) = 0.5C_q(t), \quad C_q(t+1) = 0.5C_q(t). \quad (8)
\]

Hereby, insertion of the cell can be controlled and the maps can be grown without distortion.

Also, the cells are not distributed too dense if the value of parameter \( a \) is appropriate.

**Step 7 : Deletion of cells**

Figure 3 shows deletion of cells. If the counter value \( C_i \) is smaller than threshold \( Th \), we delete the cells and its connection(s) . We declare the deleted cells to be an isolated cell. Then if the counter of the isolated cell exceeds a threshold \( Th \), the cell revives as a usual cell.

**Step 8 :** Let \( t = t + 1 \). If \( t < T \), then go to **Step 2**.

The efficiency of the algorithm have been confirmed by some numerical experiments. In the experiments, we apply the algorithm to the input space governed by some distribution. Let initial number of cells \( k = 10 \). Also, the other parameters are selected as the following:

\( \beta_c = 0.03, \beta_n = 0.006, a = 0.5, b = 3.0, Th = 0.0, \)
M = 100. After the learning is terminated, we apply the input data to the learned network and evaluate the learning result by normalized variation of the winning numbers (standard deviation divided by average).

3.1 Experiment 1
The input space is characterized by uniform distribution on four circles in Figure 4. Let black points denote weight vectors and black lines denote connections. We have confirmed that the network was grown without distortion and obtained an efficient clustering function in Experiment 1. After the learning, the normalized variation of the winning numbers is 0.20.

3.2 Experiment 2
We consider a time-variant input space: the number of circles is changed from 3 to 4 at t = 10000 (see Figure 5). Let black points denote weight vectors and black lines denote connections. The algorithm could adapt flexible to the temporal change of the input space. The maps can be grown without distortion. After the learning, the normalized variation of the winning numbers is 0.23.

3.3 Experiment 3
We consider another time-variant input space: the number of circles is changed from 4 to 3 at t = 10000 (see Figure 6). Let black points denote weight vectors and black lines denote connections. The algorithm could adapt flexibly to the temporal change of the input space. The maps can be grown without distortion. After the learning, the normalized variation of the winning numbers is 0.18.

4 Conclusion
We have considered a growing self-organizing algorithm for dynamic clustering. It is show that the algorithm can adapt flexibly to the temporal change of the input space. Our algorithm can reinforce the clustering function. Now we are trying to apply the algorithm to practical problems, e.g., data mining problems [4].

References
Figure 4: Experiment 1

Figure 5: Experiment 2
Figure 6: Experiment 3