Computer and Information Science; Vol. 9, No. 1; 2016
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Self-Organizing Map Learning with Momentum

Huang-Cheng Kuo[1] & Shih-Hao Chen[1]

[1] National Chiayi University, Taiwan

Correspondence: Huang-Cheng Kuo, Department of Computer Science and Information Engineering, National Chiayi University, Chia-Yi City 600, Taiwan. E-mail: hckuo@mail.ncyu.edu.tw

Received: December 18, 2015 Accepted: January 3, 2015 Online Published: January 31, 2016

doi:10.5539/cis.v9n1p136 URL: http://dx.doi.org/10.5539/cis.v9n1p136

Abstract

Self-organizing map (SOM) is a type of artificial neural network for cluster analysis. Each neuron in the map competes with others for the input data objects in order to learn the grouping of the input space. Besides competition, neighbor neurons of a winning neuron also learn. SOM has a natural propensity to cluster data into visually distinct clusters, which show the intrinsic grouping of data.

The self-organizing map algorithm is heuristic in nature and will almost always converge. Since self-organizing map may be trapped in a local optimum, so we introduce momentum into the learning process thus the movement of a neuron may jump over local optimum. We expect this will be similar to the learning of neurons in back-propagation with momentum. Like the learning process in back-propagation, the timing for updating the amount of movement of a neuron is either batch mode or incremental mode. However, due to the neighborhood function, the movement of a non-winner neuron is relatively small as compare to when it is a winner. So when deciding the momentum, the previous movement of a neuron needs special consideration. Experiment result show that adding momentum to self-organizing map considerably contributes to the acceleration of the convergence.

Keywords: self-organizing map, clustering, learning with momentum

1. Introduction

Cluster analysis (Richard et al., 1990) groups similar data objects into clusters. Clustering algorithms differ in the data type that they can handle and the result that they produce. Some clustering algorithms handle multidimensional data and some handle any data format as long as the similarity among data objects can be derived. For example, hierarchical clustering algorithms need only the similarities among the data objects, and density-based clustering algorithms requires the distances among data objects. Only multidimensional data type can be processed by self-organizing map (SOM) clustering algorithms. K-means clustering produces exactly the specified number of clusters. Clusters produced by density-based clustering algorithms can be in arbitrary shape. Hierarchical clustering algorithms give a dendrogram without specifying the number of clusters. SOM clustering algorithms produce a map of neurons where each neuron is associated with varying number of data objects.

Self-organizing map (Kohonen, 1982) is a competitive learning based clustering neural networks. Neurons on the map compete with each other for each input data. Generally, the self-organizing map (Kohonen, 2001) requires thousands of learning iterations to converge in order to search the optimal solution. Due to the nature of learning and searching, SOM has the possibility to get stuck in a local optimum. So, we incorporate the momentum term into the learning process of SOM (Masafumi, 1996). The idea appears in some other methods, such as the back-propagation algorithm (Vogl et al., 1988). The momentum term effectively accelerates the learning of the back-propagation algorithm (Hagiwara, 1992). In this paper, we discuss the timing for updating the momentum term in the self-organizing map which is not mentioned in the work by Masafumi. Analog to the batch mode and incremental mode for updating the weight vectors in back-propagation algorithm (Mitchell, 1997), we propose a batch mode algorithm and an incremental mode algorithm for updating the momentum term. Momentum term is the product of momentum coefficient and the previous updating amount of a neuron. However, in SOM, a non-winner neuron updates its weight vector. If the non-winner neuron is not so close to the winner, the amount of update is very tiny. In order to keep a reasonable amount of momentum, the definition of momentum term is revised as the product of momentum coefficient and the momentum mass of a neuron. Momentum mass is defined on the amount of weight vector updating of a neuron. We propose a batch mode

algorithm and an incremental mode algorithm to determine the momentum mass. We performed experiments and the result shows the quantization error decreases for SOM with momentum.

2. Related Work

Clustering (Kaufman, 1991) partitions a set of data objects into clusters. Objects in a cluster are similar to each another and are dissimilar to objects in other clusters. A good clustering method produces high quality clusters with high intra-class similarity and low inter-class similarity. Dissimilarity and similarity metric is expressed in terms of a distance function. The quality of clustering result depends on the similarity measure used by the method. Usually, for multidimensional data, Euclidean distance or Manhattan distance is adopted as the definition of distance function.

Major clustering methods can be classified into the following categories. The partitioning methods (Han, 2001) partition the given set of data objects into some groups. Then, the medoid of each group is computed. Each data object is re-assigned to a group whose medoid is closest to the data object. Then process iterates until there is no new re-assignment. A well-known partitioning method is k-means (Alsabti, 1988). The hierarchical methods (Carpinteiro, 2000) create a dendrogram of the given data objects. A hierarchical clustering method can be either agglomerative or divisive. Agglomerative hierarchical clustering algorithm initially treats a data object as a singleton cluster. Then, the algorithm iteratively merges two closest clusters into a cluster. There is a need to define the similarity between two clusters. The density-based method (Ankerst et al., 1999; Ester et al., 1996) is based on connectivity and density functions. A data object is called a core if there are enough neighboring data objects. Two parameters specified by the users are the minimum number of neighbors and the radius. Two data objects are called neighbors if the distance between them is within the radius. The grid-based method (Wang et al., 1997) quantizes the object space into a finite number of cells that form a gird structure. The model-based method (Dasgupta, 1998) hypothesizes a model for each cluster and finds the best fit of the data to the given model.

3. Method

3.1 Self-Organizing Map

Self-organizing map is an extensively used method. It is a common type of artificial neural network in which the neurons learn iteratively through competing for the input data, and through the cooperation with the neighboring neurons. A particular feature of self-organizing map is that it typically reduces multi-dimensional data to a low-dimensional grid of nodes, making them easy to visualize the result. In the self-organizing map, each map node has a unique coordinate. It represents a cluster of input vector. Self-organizing map algorithm matches a given input vector to its best matching unit (Lippmann, 1987). The best matching unit acts on the optimal fit ant its neighbors using a neighborhood function. Self-organizing map uses Euclidean distance as the distance between a neuron and an input vector. For an input vector, the neuron having the shortest distance from the input vector among all the neurons is called the winning neuron.

The weight vector of the wining neuron is updated toward the input vector. And its topological neighbors also move toward the input vector in the input space. The weight vectors are updated according to the following equation:

$$W_j(t+1) = W_j(t) + \alpha(t) \times \Big(x(t) - W_j(t)\Big) \times h_{j^*}(t) \tag{1}$$

where t is the index of the current iteration and the regression is performed for each presentation of a sample of x, denoted x(t), and $0 < \alpha(t) < 1$ is the learning rate which decreases monotonically with the regression iterations. The $h_{j^*}(t)$ is the neighborhood function which is taken as the Gaussian

$$h_{j^*}(t) = exp\big(-d^2_{j,j^*}/2R^2(t)\big) \tag{2}$$

where d^2_{j,j^*} is the distance between the map locations j and j^*, R(t) is the neighborhood radius. During an iteration, both α(t) and R(t) decrease exponentially. The decay functions for the learning rate and neighborhood radius are $\alpha(t) = \alpha \times \exp(-t/T)$ and $R(t) = R \times \exp(-t/T)$, where α, R, and T are the initial learning rate, the initial radius, and the maximum number of the iterations, respectively.

Generally, the learning process of neural networks is very slow, therefore self-organizing map requires thousands of iterations to converge.

The quantization error (QE) of a training sample is defined as the distance from the input vector to its best

matching neuron in the self-organizing map. The convergence process measures the sum of quantization error. The sum of quantization error is following equation:

$$\sum_{x \in D} \left(\arg\min_j \sqrt{\sum_i (x_i - W_{ji})^2} \right), \tag{3}$$

where D is the set of input vectors, j is a neuron, i is the dimensional index of input vectors.

The momentum term effectively accelerates the learning of the back-propagation algorithm (Hagiwara, 1992). In order to effectively accelerate the reduction of quantization error, momentum term has been incorporated into the learning process of SOM (Masafumi, 1996). However, learning process was not thoroughly discussed. Analog to the batch mode and incremental mode for updating the weight vectors in back-propagation algorithm (Mitchell, 1997), we propose a batch mode algorithm and an incremental mode algorithm for updating the momentum term.

3.2 Learning with Momentum Term

Momentum term is the product of momentum coefficient and the previous movement of a neuron. The amount of movement of a neuron differs a lot, depending on whether the neuron is winner or not, and depending on the distance from the neuron to the winner. However, conventional self-organizing map with momentum term does not consider whether the previous movement is large or small. The movement of a neuron becomes previous movement immediately after the input vector is processed. We denote such conventional self-organizing map with momentum term as immediate mode momentum term algorithm. In the following subsections, we will propose a batch mode momentum term algorithm and an incremental mode momentum term algorithm.

Let D be the set of data objects. A grid size of $N_x \times N_y$ is considered as in standard self-organizing map. Let $N_x \times N_y = M$ be a set of neurons. Each neuron is associated with a weight vector which has the same dimensionality of input vectors. The elements in the weights are initialized with values between 0 and 1.

3.2.1 Momentum Term

The momentum term is used to speedup network training and to avoid local optimal. A neuron moves toward the location of an input sample that it wins. When the neuron or its neighbor wins an input sample, it moves along with its previous direction as well as toward the location of the new input sample. In other words, the previous moving direction has some effect on the movement of a neuron. The momentum term of such effect is determined by a coefficient and the previous amount of movement of the neuron. The coefficient is called momentum coefficient, which is between 0 and 1. The amount of movement of a neuron can be rewritten as the following equation:

$$\Delta W_j(t) = \alpha(t) \times \big(x(t) - W_j(t)\big) \times h_{j^*}(t) + m(t) \times \Delta W_j(t-1) \tag{4}$$

Update neuron's weight vector:

$$W_j(t+1) = W_j(t) + \Delta W_j(t) \tag{5}$$

where m(t) is the momentum coefficient at iteration t.

Typically neurons in a network use the same momentum coefficient. Momentum coefficient usually changes from iteration to iteration. In most cases, momentum coefficient decreases over time. Momentum coefficient decreases with the formula $m(t) = m \times \exp(-t/T)$, where m is the initial value of the momentum coefficient. The initial momentum coefficient affects the learning process, so we use two values, i.e., 0.5 and 0.8, in the experiments to observe the effect.

SOM with momentum term was proposed by Masafumi. The algorithm always uses the previous amount of update of a neuron to compute the momentum term, no matter the neuron was a winner or not. Also, the momentum coefficient is fixed. The algorithm is as follows:

1 Initialize grid size N_x and N_y, learning rate α, neighborhood radius R, momentum coefficient m and the number of iterations T.

2 for each iteration t in T do

3 $\alpha(t) = \alpha \times exp(-t/T), R(t) = R \times exp(-t/T), m(t) = m \times exp(-t/T)$

4 for each input data x in D do

5 find the winning neuron. Let the winner neuron be j*.

6 for each neuron j

7 calculate $\Delta W_j(t)$ using equation (4)

8 update neuron's weights, using equation (5)

9 end

10 end

11 end

3.2.2 Batch Mode

When considering the momentum for the learning of a neuron, the momentum is the product of momentum coefficient and the previous movement of the neuron. In the learning of a neuron in back-propagation, the momentum is determined by the momentum coefficient and the previous movement of the neuron. The previous movement is somehow steady, so there is no special need to consider whether the previous movement is reasonable. However, in SOM, the previous movement of a neuron might be large if the neuron was the winner for the previous input vector, and might be small or even no movement due to neighborhood function if the neuron was not the winner. In average, the probability that a neuron wins two consecutive input vectors is 1/n, where n is the number of neurons. In other words, in most of the time, if there is no special treatment on this previous movement of a neuron, it will be near none, which results in no momentum effect.

We denote this previous movement of a neuron for computing momentum as movement mass. So, the momentum is the product of momentum coefficient and movement mass.

For discussing the batch mode momentum term, we need the concept of iteration of learning. An iteration of learning process is that each input vector in the dataset is processed once. Each neuron accumulates its own movement. The movement of a neuron is accumulated only when it is the winner for the input vector. At the end of an iteration, in other words, after all the input vectors have been processed once, the accumulated movement of each neuron is averaged by the number of input vectors won by the neuron. Then, the averaged movement of each neuron becomes the new movement mass of the neuron. Therefore the amount of weight values can be rewritten as the following equation

$$\Delta W_j(t) = \alpha(t) \times \left(x(t) - W_j(t)\right) \times h_{j^*}(t) + m(t) \times \widetilde{\Delta W_J}(t-1) \tag{6}$$

where $\widetilde{\Delta W_J}(t-1)$ is the average $\Delta W_j(t-1)$ when the neuron j won in the $(t-1)^{th}$ iteration. If a neuron did not win any input vector in previous iteration, this momentum term is 0.

Its algorithm is as follows:

1 Initialize grid size N_x and N_y, learning rate α, neighborhood radius R, momentum coefficient m and the number of iterations T.

2 for each iteration t in T do

3 $\alpha(t) = \alpha \times exp(-t/T), R(t) = R \times exp(-t/T), m(t) = m \times exp(-t/T)$

4 for each input data x in D do

5 find the winning neuron. Let the winner neuron be j*.

6 for each neuron j

7 calculate $\Delta W_j(t)$ using equation (6)

8 update neuron's weight, using equation (5)

9 if j is the winner, record $\Delta W_j(t)$ in order to compute $\widetilde{\Delta W_J}(t)$.

10 end

11 end

12 compute $\widetilde{\Delta W_J}(t)$ for each neuron

13 end

3.2.3 Incremental Mode

In this incremental mode algorithm, the amount of movement mass of a neuron is updated immediately after the neuron wins an input vector. Therefore the amount of weight values can be rewritten as the following equation:

$$\Delta W_j(t) = \alpha(t) \times \left(x(t) - W_j(t)\right) \times h_{j^*}(t) + m(t) \times \widetilde{\Delta W_J}(t-1), \tag{7}$$

where j^* is the winning neuron, $\widetilde{\Delta W_j}(t-1)$ is the movement mass. The movement mass of the winning neuron is updated, i.e., $\widetilde{\Delta W_j}(t) = \Delta W_j(t)$.

Its algorithm is as follows:

1 Initialize grid size N_x and N_y , learning rate α, neighborhood radius R, momentum coefficient m and the number of iterations T.

2 for each iteration t in T do

3 $\alpha(t) = \alpha \times exp(-t/T), \mathrm{R}(t) = R \times exp(-t/T), \mathrm{m}(t) = m \times exp(-t/T)$

4 for each input data x in D do

5 find the winning neuron. Let the winner neuron be j*.

6 for each neuron j

7 calculate $\Delta W_j(t)$ using equation (4)

8 if j is winner, update movement_mass for neuron j, using equation (7)

9 update neuron's weights, using equation (5)

10 end

11 end

12 end

4. Results

4.1 Measurement Method

We carry out the studies on the impact of momentum term. Two convergence measures are used. The first measure is to calculate the quantization error (QE) of an input vector. QE is the Euclidean distance of the input vector and the beast matching neuron. Sum of the squared QE is minimized for given training vectors.

Another convergence measure method assumes the number of clusters is known in advance. Let *k* be the number clusters in the following discussion. The size of a neuron is defined as the number of input vectors won by the neuron after the final iteration. The method selects the largest *k* neurons, called seed neuron. Each input vector won by the small neurons is assigned to the closest seed neuron. Input vectors are now partitioned into *k* clusters. Centroid of each cluster is computed. The convergence measure is sum of the squared error (SSE). The SSE as following equation:

$$\sum_{i=1}^{k} \sum_{x \in C_i} \|x - m_i\|^2 \tag{8}$$

where *k* is the number of clustering, C_i is the i-th cluster, m_i is the clustering center, x is the data.

The SSE is to calculate all data points corresponding to the sum of its distance from the cluster center and differences. We repeat all experiments 30 times and then take the average of the 30 runs.

4.2 Data Sets

In this paper, we use two data sets, Iris and Wine, from the UCI machine learning repository (Newman et al., 1988) to carry out experiments.

4.2.1 Iris

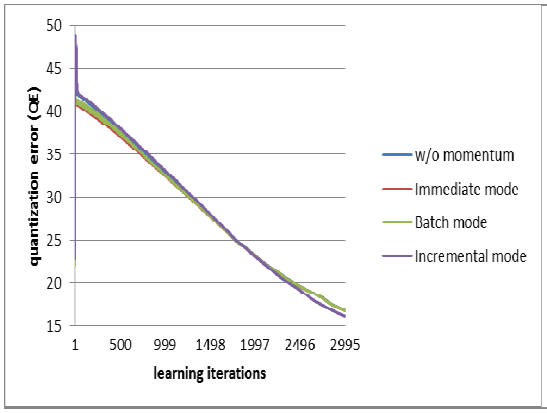
The iris data set consists of four dimensions with small variance. The dimensions are the petal length, the petal width, the sepal length, and the sepal width. Each object belongs to one of the three classes. There are 150 training instances in the data set. Each attribute is normalized with z-score method. The evaluation of QE was carried out for self-organizing map ($15 \times 15 = 225$ model and 3000 iterations, respectively.) The neighborhood function of the self-organizing map was a two-dimensional Gaussian. Figures1- 4 show the QE and SSE which are plotted as the function of iteration, where learning rate $\alpha = 0.01$, $\alpha = 0.001$, and $\alpha = 0.0001$, momentum $m = 0.5$ and $m = 0.8$, respectively. In each of the Figures, we compare the performance of the four methods, i.e., the original self-organizing map algorithm without momentum term, self-organizing map with immediate mode momentum term, self-organizing map with batch mode momentum term, and self-organizing map with incremental mode momentum term. In the Figures, the methods are denoted as w/o momentum, immediate mode, batch mode, and incremental mode.

4.2.2 Wine

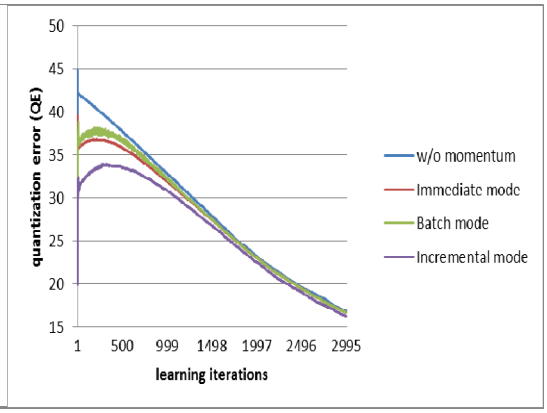
The measurements consist of thirteen dimensions. The class attribute has 3 values. There are 178 instances in the

data set. Each attribute is normalized with z-score method. The evaluation of QE was carried out for the self-organizing map ($10 \times 10 = 100$ model and 3000 iterations, respectively.) The neighborhood function is a Gaussian centered in the winner neuron. The neighborhood size, learning rate and momentum coefficient decreases exponentially during training. Experiment was conducted on original self-organizing map without momentum term, self-organizing map with immediate mode momentum term, self-organizing map with batch mode momentum term, and self-organizing map with incremental mode momentum term which were trained with 3000 iterations. After learing, Figures 5 to 8 show the QE and SSE which are plotted as the function of iteration, where learning rate $\alpha = 0.01$, $\alpha = 0.001$, and $\alpha = 0.0001$, momentum coefficient m = 0.5 and m = 0.8, respectively. In each of the Figures, we compare the performance of the four methods.

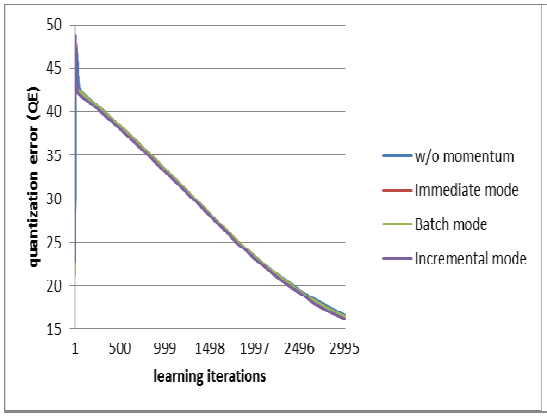
4.3 Statistics and Data Analysis



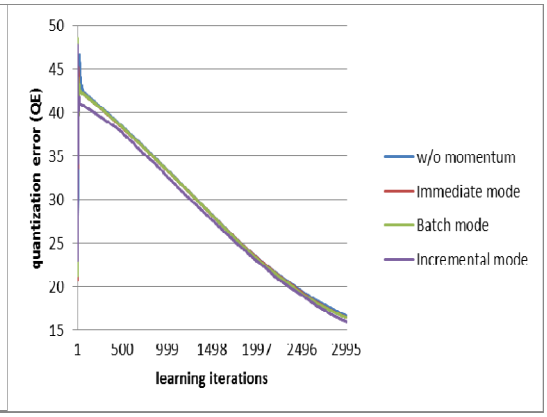
(1.a) learning rate = 0.01



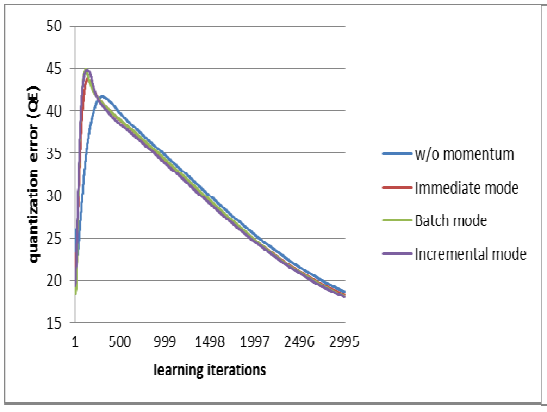
(2.a) learning rate = 0.01



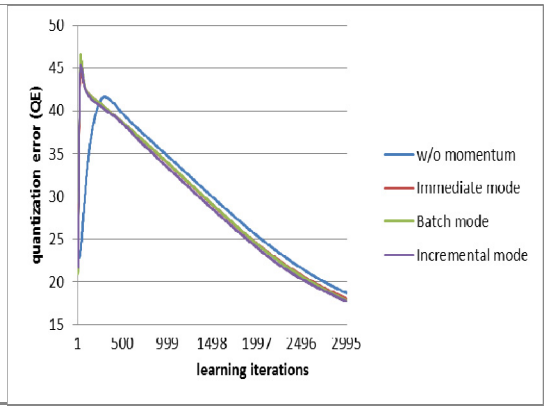
(1.b) learning rate = 0.001



(2.b) learning rate = 0.001



(1.c) learning rate = 0.0001

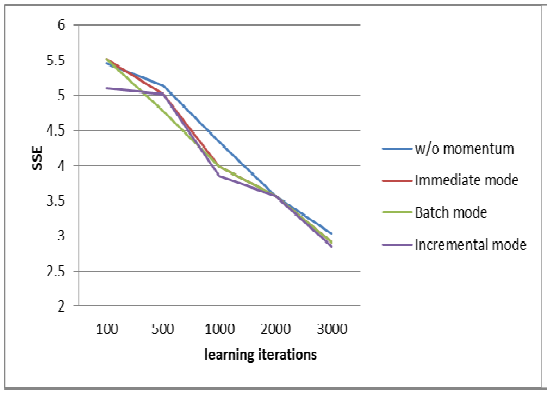


(2.c) learning rate = 0.0001

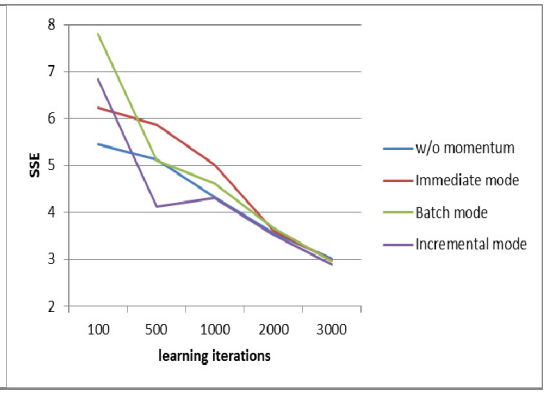
Figure 1. Quantization error of results against learning rate and momentum coefficient =0.5 in the Iris data

Figure 2. Quantization error of results against learning rate and momentum coefficient =0.8 in the Iris data

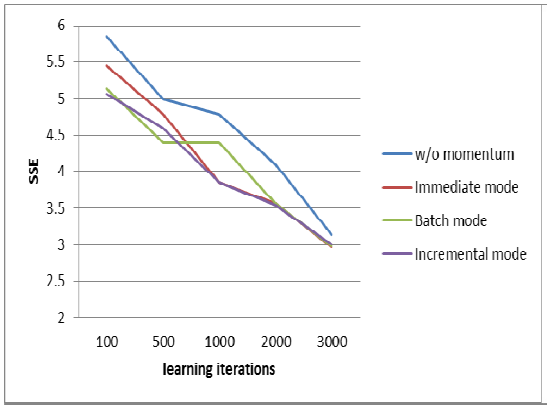
According to the experimental results shown in Figures 1 and 2, we observe that when learning rate is low, higher momentum coefficient results lower QE. From the Figure (1.a), after 250 iterations of learning, the QE value for momentum coefficient = 0.8 will be lower. For the self-organizing map method with batch mode momentum term, the QE value is less than 40. The QE value is less than 35 for the self-organizing map with incremental mode momentum term. From Figure (1.c) and (2.c), before 250 iterations of learning, the QE value of self-organizing map with batch mode momentum term and self-organizing map with incremental mode momentum term is higher. When the learning rate is small, the convergence process is slow. When momentum coefficient is larger, the QE value is higher. With the same learning rate and momentum coefficient, the incremental mode has the minimum QE value.



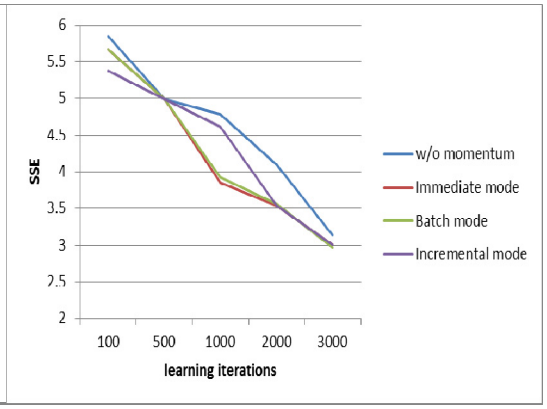
(3.a) learning rate = 0.01



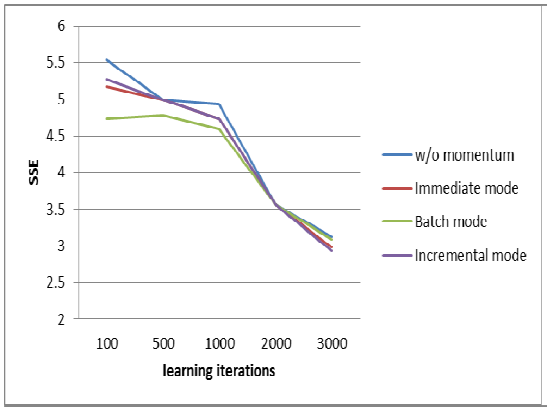
(4.a) learning rate = 0.01



(3.b) learning rate = 0.001

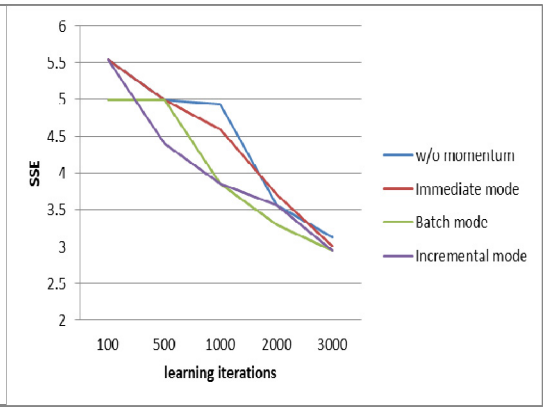


(4.b) learning rate = 0.001



(3.c) learning rate = 0.0001

Figure 3. SSE of results against learning rate and momentum coefficient =0.5 in the Iris data

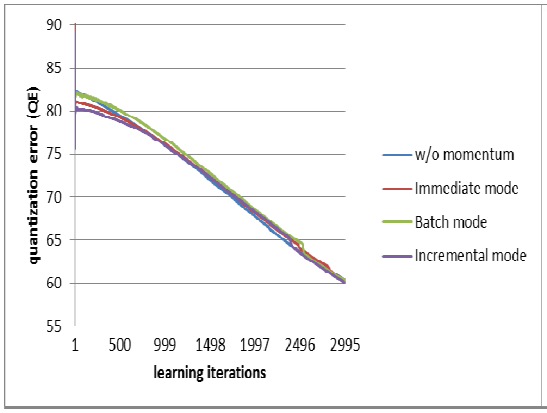


(4.c) learning rate = 0.0001

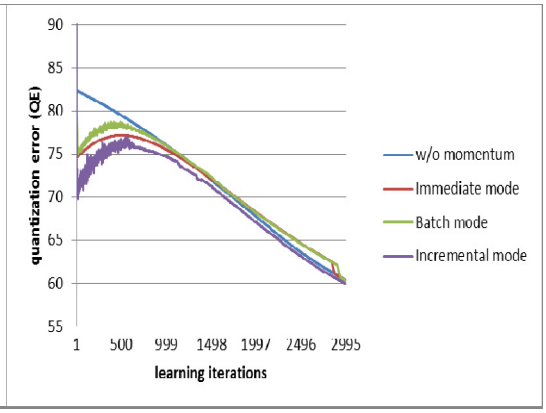
Figure 4. SSE of results against learning rate and momentum coefficient =0.8 in the Iris data

According to the experimental results shown in Figures 3 and 4, we discuss the relationship between momentum coefficient and learning rate. From Figure (3.a) and (4.a), before 250 iterations of learning, self-organizing map with any mode of momentum term has relatively high value of SSE. This is because the momentum term of such effect is determined by a coefficient and the previous amount of movement of the neuron. The higher momentum coefficient is, the higher SSE values are. After 250 iterations of learning, the SSE values for self-organizing map with any mode of momentum term are significantly lower.

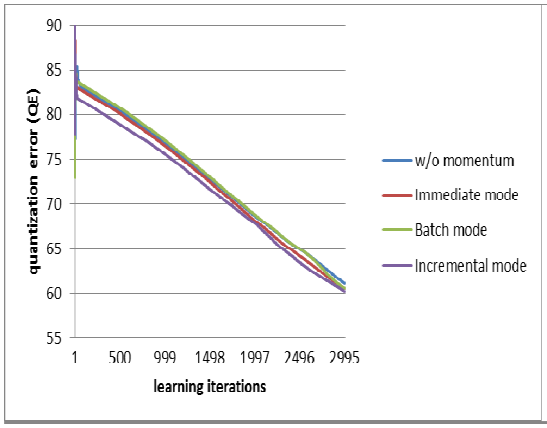
From Figure (3.c) and (4.c), when learning rate = 0.0001, adding momentum term significantly accelerates convergence.



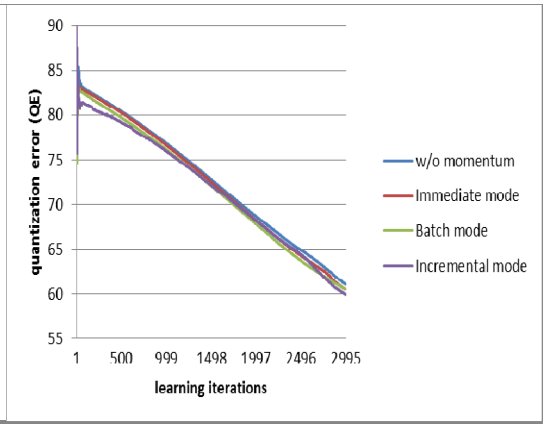
(5.a) learning rate = 0.01



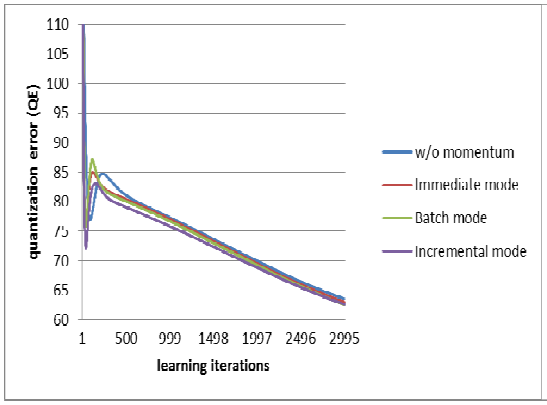
(6.a) learning rate = 0.01



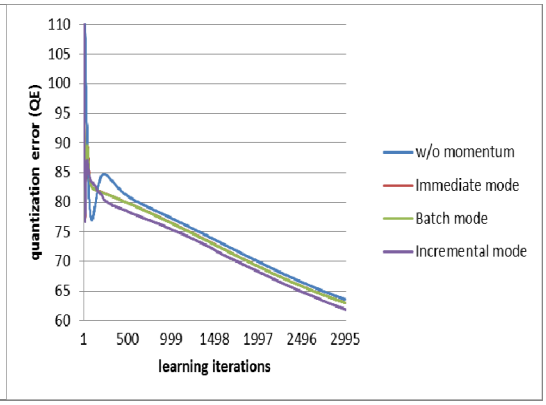
(5.b) learning rate = 0.001



(6.b) learning rate = 0.001



(5.c) learning rate = 0.0001

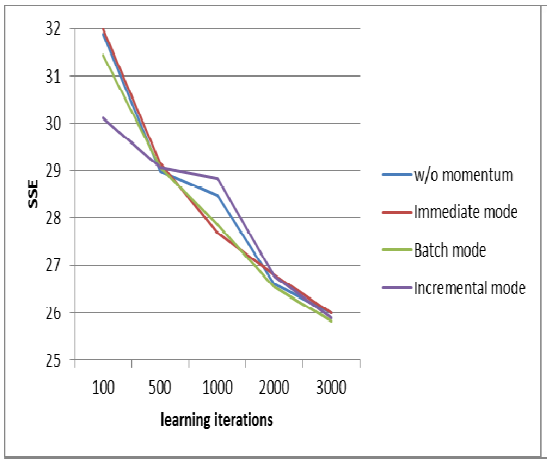


(6.c) learning rate = 0.0001

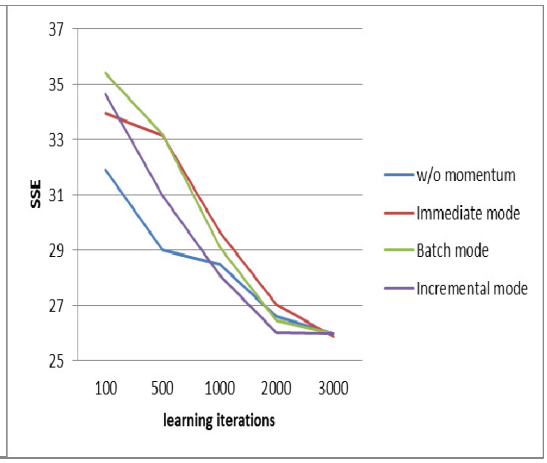
Figure 5. Quantization error of results against learning rate and momentum coefficient =0.5 in the Wine data

Figure 6. Quantization error of results against learning rate and momentum coefficient =0.8 in the Wine data

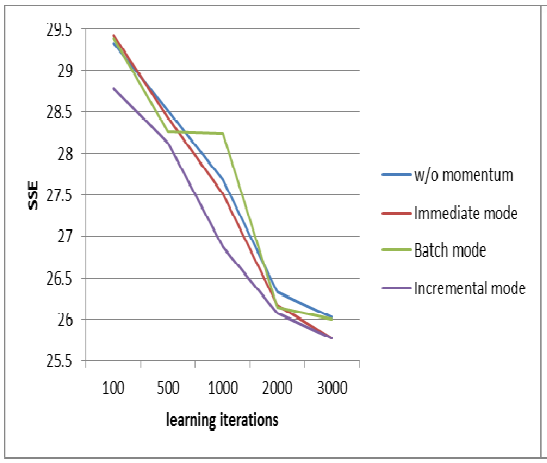
According to the experimental results shown in Figures 5 and 6, when the learning rate is low, there is slow convergence value. Whereas, when the learning rate is high, the learning process converges faster. At the same learning rate, the convergence effect with momentum coefficient =0.8 is significantly higher than that with momentum coefficient =0.5. With the same momentum and learning rate, the quantization error in incremental momentum has the lowest value.



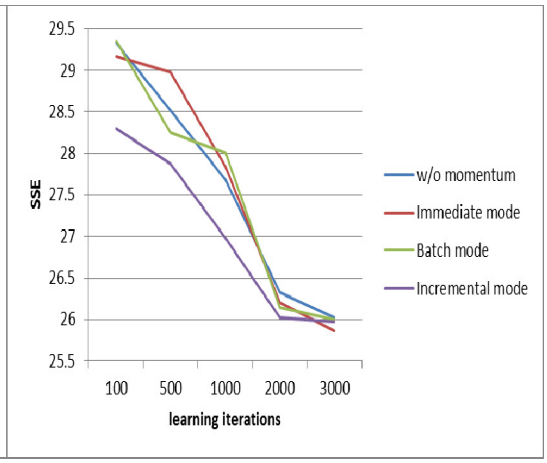
(7.a) learning rate = 0.01



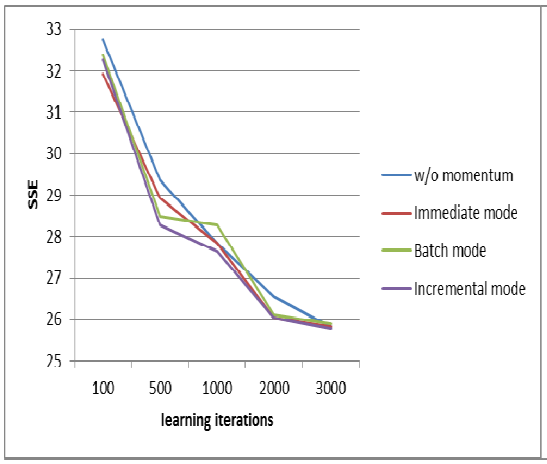
(8.a) learning rate = 0.01



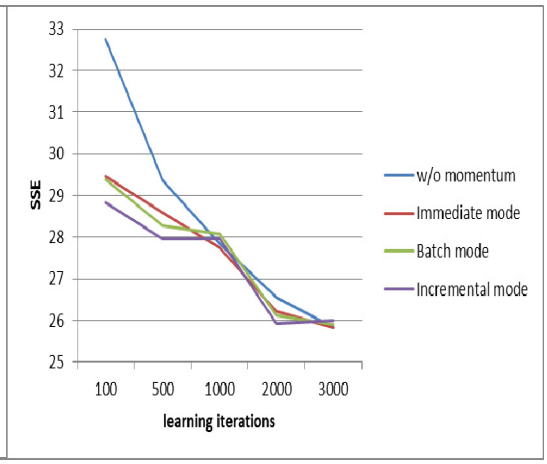
(7.b) learning rate = 0.001



(8.b) learning rate = 0.001



(7.c) learning rate = 0.0001



(8.c) learning rate = 0.0001

Figure 7. Quantization error of results against learning rate and momentum coefficient =0.5 in the Wine data

Figure 8. Quantization error of results against learning rate and momentum coefficient =0.8 in the Wine data

According to the experimental results shown in Figures 7 and 8, we discuss the relationship between momentum coefficient and learning rate. Since Wine contains more complex thirteen dimensions, from the Figure (a), the learning rate = 0.01, when added the momentum term, there is a higher SSE value. From Figure (8.c), the learning rate = 0.0001, the added momentum term can effectively reduce the value of SSE.

In the experiment, we compared the original self-organizing map without momentum term, self-organizing map with immediate mode momentum term, self-organizing map with batch mode momentum term, and self-organizing map with incremental mode momentum term. The results suggest that the learning process by adding the momentum term can accelerate the convergence, especially in the incremental mode.

5. Conclusion

Momentum term has been incorporated into the learning process of SOM. Momentum term of a neuron is the product of mememtum coefficient and the previous movement of the neuron. However, the previous movement of a neuron may be very small if (1) the neuron was not the winner of the previous input vector, and (2) the neuron is far away from the winning neuron of the previous input vector. Thus, the effect of momentum term in learning process will be lost. In this paper, we discuss this phenomenon and propose two modes for updating the prvious movement of a neuron when computing the momemntum term.

For discussion convenience, we temporarily coined a term, *momentum mass*, to denote the previous movement of a neuron. Thus, momentum term of a neuron is the product of mememtum coefficient and momentum mass of the neuron. The batch mode momentum term algorithm updates momentum mass of all neurons after an iteration of all input vectors. The new momentum mass of a neuron is the average movement of the neuron when it wins the input vectors in the iteration. The incremental mode momentum term algorithm updates momentum mass of a neuron after it wins an input vector.

Experimental results show that SOM with momentum term can accelerate convergence. Incremental mode momentum algorithm has the most obvious effect.

References

Alsabti, K., Ranka, S., & Singh, V. (1988). *An Efficient K-means Clustering Algorithm.* First Workshop on High Performance Data Ming.

Ankerst, M., Breunig, M., Kreigel, H. P., & Sander, J. (1999). *OPTICS: Ordering Points to Identify the Clustering Structure.* ACM SIGMOD Conference, 49-60.

Carpinteiro, O. A. S. (2000). *A Hierarchical Self-Organizing Map Model for Sequence Recognition*. International Conference on Artificial Neural Networks, 98, 815-820. http://dx.doi.org/10.1007/s100440070012

Dasgupta, A., & Raftery, A. E. (1998). Detecting Features in Spatial Point Processes with Clutter via Model-based Clustering. *Journal of the American Statistical Association, 93*, 294–302. http://dx.doi.org/10.2307/2669625

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* International Conference on Knowledge Discovery and Data Mining, 226–231.

Hagiwara, M. (1992). Theoretical Derivation of Momentum Term in Back-propagation. *Neural Networks, 1*, 692-686. http://dx.doi.org/10.1109/IJCNN.1992.287108

Han, J., & Kamber, M. (2001). *Data Mining: Concepts and Techniques.* Academic Press.

Kaufman, L., & Rousseeuw, P. J. (1991). Finding Groups in Data: an Introduction to Cluster Analysis. *Biometrics, 47*(2), 788. http://dx.doi.org/10.2307/2532178

Kohonen, T. (1982). Self-organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics, 43*, 59–69. http://dx.doi.org/10.1007/BF00337288

Kohonen, T. (2001). *Self-organizing Maps, Springer Series in Information Sciences.* Springer.

Lippmann, R. (1987). Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, 4-22. http://dx.doi.org/10.1109/MASSP.1987.1165576

Masafumi, H. (1996). Self-organizing Feature Map with a Momentum Term. *International Joint Neurocomputing, 10,* 71-81. http://dx.doi.org/10.1016/0925-2312(94)00056-5

Mitchell, T. (1997). Artificial Neural Networks. Chapter 4 in *Machine Learning*. McGraw Hill.

Newman, D. J., Hettich, S., Blake, C. L., & Merz, C. J. (1988). *UCI Repository of Machine Learning Database.*

Richard C. D., & Anil K. Jain, (1990). *Algorithms for Clustering Data. Technometrics, 32*, 227. http://dx.doi.org/10.2307/1268876

Voegtlin, T. (2002). *Recursive Self-Organizing Map*. Neural Networks (Vol. 15, pp. 8-9).

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., & Alkon, D. L. (1988). Accelerating the Convergence of the Backpropagation Method. *Biological Cybernetics, 59*, 257-263.

Wang, W., Yang, J., & Muntz, R. (1997). *STING: A Statistical Information Grid Approach to Spatial Data Mining.* International Conference on Very Large Data Bases, 186–195.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/).