J. Herbert, J.T. Yao, GTSOM: Game Theoretic Self-Organizing Maps, K. Chen and L.P. Wang (eds.), Trends in Neural Computation, Springer-Verlag, 2007, pp199-224.

Chapter 8

GTSOM: GAME THEORETIC SELF-ORGANIZING MAPS

Joseph Herbert, JingTao Yao
Department of Computer Science
University of Regina, Saskatchewan, Canada, S4S 0A2
[herbertj,jtyao]@cs.uregina.ca

Abstract Self-Organizing Maps (SOM) is a powerful tool for clustering and discovering patterns in data. Input vectors are compared to neuron weight vectors to form the SOM structure. An update of a neuron only benefits part of the feature map, which can be thought of as a local optimization problem. A global optimization model could improve representation to data by a SOM. Game Theory is adopted to analyze multiple criteria instead of a single criteria distance measurement. A new training model GTSOM is introduced to take into account cluster quality measurements and dynamically modified learning rates to ensure improved quality.

Keywords: Game theory, competitive learning, self-organization, SOM, global optimization

1. Introduction

The material presented in this work is the culmination of research completed [1, 2] within the areas of competitive, unsupervised learning in SOM. The work is an attempt to move away from the local optimization process of traditional competitive learning in SOMs.

The problem with local optimization becomes apparent since only a single criterion is used to match neurons of a SOM to input vectors. That is, the choice and update of a neuron does not take into account the entire situation and configuration of the SOM. The goal was to build a new training model for a SOM that allows for a global optimization algorithm to be used in the training of the network of neurons. This new algorithm is introduced as GTSOM.

SOM, introduced by Kohonen [3], is an approach to discovering similar patterns found within data [4, 5]. Used primarily to cluster attribute data for pattern recognition, SOMs offer a robust model with many configurable aspects to suit many different applications. These applications include document organization using term usage and many other problems where classification is needed.

The traditional SOM method that updates neurons in the network was based only on similarity to individual input, presented in a discrete way. This is considered a problem in the long run as training iterations progress. Local optimization occurs when an input vector is presented to the network. The fact that the work done on manipulating weight vectors in order to represent similarities in input may be overwritten as iterations progress is disconcerting.

The process of training a system with subsets of data in order to acquire new knowledge with new data is similar to that of knowledge discovery in databases [6, 7], and has been used in other areas [8] apart from artificial intelligence.

Methods must be introduced that can measure the quality of SOM at any point in training. These methods will help us ensure that any new training techniques introduced into the model increases the quality of the SOM. Since learning during training is unsupervised, these methods must allow for automation for continued absence of user involvement. Efficient quality measures need to be created so that the state of the network can be acquired during the training process. Measures such as weight vector cluster density and distances between weight vector clusters can be considered adequate to assess whether or not the network represents training data accordingly.

The movement towards a global optimization model for training is necessary for three reasons. First, the final trained network may be stable in terms of input similarity, but the final network is biased towards input that is late in presentation. Second, the use of global optimization techniques could help in demystifying the process of SOM training since various idiosyncrasies are lost through the thousands of iterations and vector updates. Third, an infrastructure to govern the global optimization techniques will be able to help in governing variables that are used in performing weight vector updates, such as learning rates and neighbourhood sizes.

This work will use game theory as our underlying method in governing competitive learning in a SOM. Game theory allows us to organize and see cause-effect relationships between the multiple measures we use and the ways we can implement them. That is, if we find a certain strategy

can help a particular neuron improve quality, we are able to quickly determine which neuron should be chosen and which action to undertake.

A new algorithm GTSOM that utilizes aspects of game theory will be presented and thoroughly examined. This allows for global optimization of the feature map. This technique could be used to ensure that competitive learning results in the modification of neurons that are truly suitable for improving the training results. This research is beneficial because it improves the SOM training model by ensuring quality is improved every iteration of training. This may decrease the time required for training and creates better neuron clustering of input similarity.

The work is organized as follows: Section 2 contains background information regarding competitive learning, self-organizing maps, game theory, and clustering in general; Section 3 introduces the original contributions in the field of competitive learning in SOMs, beginning with a new SOM model and continuing with the introduction of the GTSOM algorithm; Section 4 contains the analysis of the findings acquired from the testings of our new model and algorithm; The summary of contributions and other conclusions are given in Section 5.

2. Background Information

This section acknowledges some of the background information used as foundational support for this work. A thorough overview of traditional SOMs is presented including: artificial neurons, weight and input vectors and neighbourhoods. An in-depth look at competitive learning and how it is used to train the SOM is provided as well, with explanations of the decaying learning rate α and neighbourhood sizes. Information regarding game theory and games is provided. This includes the notions of payoffs and payoff tables.

Traditional SOM Overview

This section details the underlying components used in SOM, including learning, artificial neurons, topologies, and input.

Self-Organizing Maps. SOMs were first introduced by Kohonen [3]. The SOM requires a set W of artificial neurons,

$$W = \{w_1, \ldots, w_n\}, \tag{8.1}$$

where w_i is the *i-th* neuron of the map. In neural network (NN) theory [10], artificial neurons act as a series of transformation functions that, given an input, a distinct output is presented as either input for

another neuron, or final output for the network. NNs emulate their biological neuron counterparts [11], with inputs representing synapses, outputs representing axoms that can be connected to other dendrites via synapses.

Artificial neurons in a SOM differ from those of NNs in terms of connectivity. Connectivity between neurons are not linear and fixed, as in NNs. Connectivity to a given neuron in the network is formulated by membership to the neighbourhood set of that particular neuron. Connectivity implies communication between neurons in terms of output of neurons act as input to others. This not the case for neurons in a SOM.

Neurons in a SOM. There are two ways in which connections between artificial neurons in a SOM can be gathered. First, immediate neuron adjacency in the network topology can be thought of as a connection, since influence (update) of a neuron's weight vector affect neurons in its immediate vicinity (neighbourhood). Second, the neighbourhood of a neuron can be acquired by finding neurons of growing adjacency that have weight vectors similar to that of the original. The neighbourhood method of finding neurons of logical physical adjacency to a particular neuron is crucial to the training of a SOM, as changes to a neuron weight vector should influence those weight vectors that are near and similar to the originally updated neuron.

Each artificial neuron in a SOM has a weight vector of equal dimensionality to the input vectors. Therefore, for a set of neurons, each neuron $w_i \in W$ has a weight vector $\vec{w}_i$ associated with it. A set W of neurons has a set $\vec{W}$ of weight vectors,

$$\vec{W} = \{\vec{w}_1, \ldots, \vec{w}_n\}. \tag{8.2}$$

For any $\vec{w}_i$ in $\vec{W}$, the dimensionality of $\vec{w}_i$ is equal to the dimensionality of any input vector $\vec{p}_j$ presented to the network during training, shown in Equation (8.3). Vector similarity between neuron weight vectors and input vectors are measured during training. Training is the process of modifying the SOM in order to create a map that adequately represents the training input. Thus, if dimensionality differs between the two, some components must be normalized - resulting in a loss of information.

$$\vec{w}_i = \begin{bmatrix} c_{i,1} \\ c_{i,2} \\ \vdots \\ c_{i,r} \end{bmatrix} \text{ and } \vec{p}_j = [p_{j,1}, p_{j,2}, \cdots, p_{j,r}] \tag{8.3}$$

Each neuron $w_i \in W$ has a set of neurons whose proximity is within that defined by d, a scalar whose value is changed according to an iteration q. Therefore, for each neuron w_i, the neighborhood $N_i(d) = \{w_r, \ldots, w_s\}$ consists of all neurons that have connectivity to w_i within distance d.

SOM Input Processing. Training of a SOM is typically done by using a set of input vectors,

$$P = [\vec{p}_1, \vec{p}_2, \cdots, \vec{p}_m], \tag{8.4}$$

where the i-th input vector corresponds to the i-th tuple in the original information table. During training, each input vector is presented to the network sequentially, which will be discussed in detail shortly.

SOM Training Model

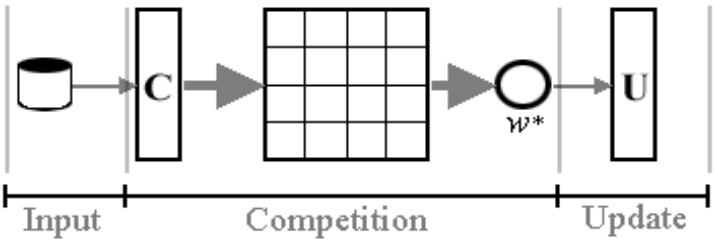
Weight vectors are adjusted according to a learning rate α that is decreased over time to allow for fast, vague training in the beginning and specific, accurate training during the remainder of the runtime. A SOM model contains three fundamental procedures that are required in order to discover clusters of data.

The first procedure consists of all preprocessing tasks that are required to be completed before training can take place. This includes initializing the weights vectors of each neuron either randomly or by some other method [13, 14] that determines suitable frequency of values. Another task to be performed is that of input vector creation.

The training process of the SOM can begin once preprocessing has been finished. The training model is divided into three layers: the input layer, competition layer, and the update layer. The input layer controls when each input vector is inserted into the SOM. The competition layer oversees competition between neurons where a suitable neuron is chosen that has a weight vector with highest degree of similarity. This neuron, as well as its neighbors, gets updated so that it becomes more similar to that of the input vector in the update layer. This process is repeated for every input, resulting in a completed iteration. The SOM Training model is presented visually in Figure 8.1.

The degree of change in which a weight vector becomes more similar to that of the input vector is manipulated through the learning rate α. Once many iterations have taken place, a suitable feature map will have been created.

Figure 8.1. The layers of a SOM during the training process.



This final trained map is now suitable for classifying additional data similar to that of the training set. This feature map is a collection of neurons with weight vectors assuming values corresponding to the training input distribution.

Local Optimization in SOM. The update of single neurons depend solely on their similarity to input, therefore a change in $\vec{w}_i(q)$ results in a modification of an original weight vector in the next iteration $\vec{w}_i(q+1)$.

One input vector $\vec{p}_j$ is presented into the network at any given time, and is most similar to that of some weight vector $\vec{w}_i$ associated with a neuron w_i. A winning neurons' weight vector is updated to become more similar to that of the current input. Therefore, $\vec{w}_i \mapsto w_i^*(q)$ is updated to $\vec{w}_i \mapsto w_i^*(q+1)$

The update mechanism for the training model only improves one neuron to each input to the full extent of the learning rate. This is considered a local optimization technique since erasure of previous work completed occurs as well as the inherent partial ordering of the input vectors. For example, the final weight vector configuration will be entirely different at the end of the training session, if a different ordering of input vector presentation is selected.

However, if a model was used in order to ensure as many deserving neurons as possible benefited from their similarity to one input, one may start moving towards a global optimization training procedure. However, if we decide to compare all input vectors to a single neuron, updating it's weight vector for each, we begin to see extraneous work being completed, not to mention a huge problem of overfitting.

This logically leads to the fact that all input vectors have a possibility to be compared to all neurons, given the multiple iterations through the input vector set P. At first glance, this may seem like global optimization of the entire network to all input, but since order is important

when presenting input vectors to the network, this process is simply a truncation of local optimization procedures.

A Brief Introduction to Game Theory

In the past decade, game theory has been one of the core subjects of the decision sciences, specializing in the analysis of decision-making in an interactive environment. The disciplines utilizing game theory include economics [15, 16], networking [17], and cryptography [18, 19]. Game theory was founded by von Neumann. The first important book was The Theory of Games and Economic Behavior [9], which von Neumann wrote in collaboration with Morgenstern. Certainly Morgenstern brought ideas from neoclassical economics into the partnership, but von Neumann, too, was well aware of them and had made other contributions to neoclassical economics.

Game theory arose from the result of trying to mathematically express a simple game, including rules and actions a player of that game would perform. Game theory is an important domain since so many areas can use it. Many applications or problems can be expressed as a game between two or more players. If a problem or application can be expressed as a game, it can be expressed in a way that some aspects of game theory can be utilized. Therefore, the study of game theory can be thought of as an advanced problem solving technique that can be used in many domains.

The study of game theory is divided into three major areas: mathematical models of games, game theory and how it applies to economic applications, and game theory applications in other areas. The last major area - game theory for applications are studied by biologists, management, application mathematicians, legislators etc.

The basic assumption of game theory in terms of usage is that all participating players are rational in terms of attempting to maximize their expected payoffs. This presents problems when compared with neoclassical economics. It narrows the range of possibilities that a party can choose from. Rational behavior is much more predictable than irrational behavior, as opposing parties are able to determine other party's strategies on the basis that they will not do anything that makes their situation worse than before.

In a simple game put into formulation, a set of players $O = \{o_1, \ldots, o_n\}$, a set of actions $S = \{a_1, \ldots, a_m\}$ for each player, and the respective payoff functions for each action $F = \{\mu_1, \ldots, \mu_m\}$ are observed from the governing rules of the game. Each player chooses actions from S

to be performed according to expected payoff from F, usually some a_i maximizing payoff $\mu_i(a_i)$ while minimizing other player's payoff.

Further reading on game theory and applications can be found in *Game Theory* by D. Fudenberg *et al* [12].

3. A Game-Theoretic Approach to Competitive Learning

This section introduces the new material developed pertaining to creating a new model for self-organizing maps to facilitate global optimization. The first section will review some of the methods for competitive learning, measuring similarity, as well as present new quality measurements needed by the new model.

SOM Training and Competitive Learning

In this section, a review of existing training techniques, competitive learning processes, and similarity measures will be presented. This information helps in creating new ideas to further the progress in reaching the goals of this work.

Forming Trained Maps. In order for a SOM to cluster data, it must be trained with suitable data. Training a SOM requires the combination of three layers that work in tandem, where an output of one layer is treated as input to the next. This training model is shown in Figure 8.1.

The first layer, denoted as the input layer, consists of a data store to be formatted into a set of input vectors P. An input vector represents a *tuple* within the data set. Each input vector $\vec{p}_i \in P$ is used as input for the next layer of a SOM.

The second layer, denoted as the competition layer, manages the competitive learning methods [20] within the SOM. This layer determines which neuron w_i has a weight vector $\vec{w}_i$ with minimum distance (maximum similarity) to $\vec{p}_i$. From this layer, a winning neuron w_i^* is marked to be updated in the third and final layer.

The update layer updates the weight vector associated with the winning neuron that was used as input. After the updating of the neuron, it's weight vector is more attuned to that of the input vector. Transposing the values of both the input vector and the winning neurons weight vector onto a Cartesian plane, the distance between the vectors is smaller than it was at the beginning of the process. Once the weight vector of the winning neuron has been changed, the neighbourhood is changed, to

a lesser extent, to reflect similarity to the input, since it is adjacent to the winning neuron.

Each neuron $w_i \in W$ has a set of neurons, called its neighbourhood $N_i(d)$, where each neuron's proximity is within that defined by d, a scalar value that is changed according to an iteration q. A d of 1 would result in neurons within 1 unit to be added to the neighbourhood. Therefore, for each neuron w_i, the neighborhood $N_i(d) = \{w_r, \ldots, w_s\}$ consists of all neurons that have connectivity to w_i within distance d. An iteration q is completed when all input vectors have been introduced to the competition layer, a neuron has been selected as the winner, and the update layer has completed.

The learning rate α of the entire network is to be in the range $0 < \alpha < 1$. The learning rate is used as a modifier that determines how much a weight vector $\vec{w}_i$ is changed to become more similar to that of the current input vector.

Sufficient artificial neurons are created in order to adequately define clusters in our data. Too few neurons will result in closely-packed groupings, making it difficult to discern between clusters. Too many neurons will increase the runtime of the algorithm without any positive gain in representation [21].

As in the case of NNs, a SOM must be trained on a subset of data before the map is considered applicable.

Competitive Learning. To find the neuron $w_i \in W$ that has a weight vector closest to $\vec{p}_k$, similarity measures [22] are observed between each neuron and the input vector.

For example, a neuron w_i^* is marked as the winner (denoted by the asterisk) for input vector $\vec{p}_k$ if it has the smallest sum-of-squares value between its weight vector and the input vector.

Once a winning neuron has been identified, it's weight vector must be updated according to the learning rate α_q corresponding to iteration q. In addition, the neighborhood of that neuron must be updated so that neurons connected to the winner reflect continued similarity to the new information presented to the network. This process is done with functions Update_w and Update_N that update the winning neuron and its neighborhood respectively. The update of a winning neuron is completed by computing the Kohonen rule [11]. With α being used to determine how much of the distance between the original weight vector and current input vector is added to create a new weight vector allows the algorithm to specify how fast training can occur.

We wish to use a smaller learning rate to signify that although these neurons did not win the competition for the input vector, they do have

some connectivity to the neuron that did. This step preserves similarity between neurons adjacent to one another. Neighbourhood neurons are updated with a fractional part of α simply because they are not as similar to the input as is the winning neuron. Therefore, they should not be rewarded for similarity to the input, but be rewarded because of association with the winning neuron.

The process of updating a neuron and its neighbors to become more similar to that of the input vector can be thought of as a local optimization procedure. For any given input vector, the update layer in Figure 8.1 only adjusts neurons based on a single input, not the full data set. The competition layer does not take into account other information that could help in choosing a neuron better suited for distinguishing clusters of similar features. There is no way of adjusting the process if there happens to be too many neurons representing too few input vectors and vice versa [3]. Neurons representing completely dissimilar sets of input vectors should not be adjacent whereas separate groups of neurons representing similar sets of input vectors should not be far apart physically.

Therefore, a method of ensuring proper cluster density according to the related distribution within the data set should be used. Cluster density calculates that number of input vector associations per neuron in a particular cluster. A method of ensuring that dissimilar weight vectors representing dissimilar subsets of data should be as far apart on the feature map as physically possible should also be present. An additional competition layer must be added that can identify what actions should be performed in order to ensure that the above problems do not persist.

Similarity Measures. In the traditional SOM model, a neuron must be chosen as the winner in regards to a single input vector. The most common of these is the sum-of-squares similarity which computes the distance between vectors, in this case the input vector and a neuron weight vector, shown in Equation (8.5),

$$w_i^* = \left\{ w_i \mid \min \left(\sum_{j=1}^{m} (\vec{w}_i[j] - \vec{p}_k[j])^2 \right) \right\}. \tag{8.5}$$

This distance measure signifies similarity between the input vector and a neuron. The neuron whose weight vector is spatially closer to the input vector will have the smallest sum-of-squares result.

Given a SOM consisting of n neurons and an input vector set of size m, the training algorithm is expected to compute $m \times n$ sum-of-squares

calculations for an iteration, where an iteration consists of representing each input vector to the competition layer once.

Once a winner neuron has been chosen, denoted w_i^*, its weight vector must be updated to become more spatially closer, or more similar, to that of the current input vector. This is done by scaling the difference between the two vectors via the learning rate α and adding the result to the original weight vector, shown in Equation (8.6):

$$\vec{w}_i^*(q) = \vec{w}_i^*(q-1) + \alpha(\vec{p}_k(q) - \vec{w}_i^*(q-1)) \ . \tag{8.6}$$

The weight vector for the winning neuron w_i^* at iteration q is equal to the original weight vector at iteration $(q-1)$ plus the α-scaled difference between the current input vector $\vec{p}_k$ and the original weight vector $\vec{w}_k$.

The neighbourhood must then be updated. The neighbourhood set is calculated around w_i^* according to the decaying neighbourhood distance d. The update of a neighborhood is done via Equation (8.7):

$$\vec{w}_{N_{i^*}(d)}(q) = \vec{w}_{N_{i^*}(d)}(q-1) + \alpha\prime(\vec{p}_k(q) - \vec{w}_{N_{i^*}(d)}(q-1)) \ . \tag{8.7}$$

The modified learning rate $\alpha\prime$ denotes a smaller learning rate that is used on the neurons within the neighbourhood set $N_{i^*}(d)$.

The learning rate α in Equation (8.6) is derived from a decreasing polynomial formula [23]. The learning rate $\alpha\prime$ is a modified fractional scalar of α.

Measuring SOM Quality with Clusters

The competitive layer in the traditional SOM model does not have the ability to find a neuron which best represents the current input vector as well as having the ability to improve the quality of neuron placement and density. Improving quality in a SOM could include an increased ability to create and define better clusters.

Defining Clusters in a SOM. In order to determine the quality of a SOM, definitions on what is considered a high-quality cluster must be discovered. Clusters in a SOM are simply groupings of neurons that have strong weight vector similarities and physical adjacency. Clusters are the most visible organizational construct when viewing a trained SOM. Clusters can be defined in two ways: by the actual input data that was used to adjust the weight vectors or by the neurons associated with that data.

Cluster density is the ratio of neurons in a cluster with the number of input vectors it represents, shown in Equation (8.8). For example, two clusters K_1 and K_2 both have 8 neurons belonging to it. K_1 has 40 input vectors associated with it whereas K_2 has 70 input vectors. Using Equation (8.8), we find that the density $D_{K_1} = 5$ and $D_{K_2} = 8.75$ vectors/neuron. K_2 is more dense than K_1.

$$D_{K_i} = \frac{|\{\vec{p}_i | \vec{p}_i \mapsto w_i, w_i \in K_i\}|}{|K_i|} \quad . \tag{8.8}$$

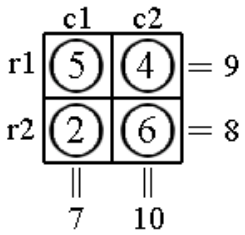
With the two criteria for defining clusters, two methods of representing clusters arise. First, a centroid vector can be used as a representation of the cluster. This vector could be calculated by taking the average of all weight vectors that the cluster includes. Representation of clusters are needed in order to give some pre-determined knowledge of SOM quality. Clusters will be used as tools to improve SOM quality. We are using cluster representation techniques in order to simplify calculations and decrease runtime. Second, a neuron whose weight vector is most similar to that of the average weight vector of all neurons could be given representation status. In addition to the two methods of representing clusters in a SOM, two methods can be used in order to find a neuron required in the latter method:

1 **Using vectors.** If a centroid input vector for a cluster is known, we can discover which neuron that centroid input vector is most similar to.

2 **Using strength.** If we wish for the calculations of centroid to be strictly neuron based, we can find groups of neurons and determine which of those neurons have won more competitions.

Assuming a two-dimensional grid layout for neurons within a SOM, horizontal and vertical calculations can be performed on all numerical values associated with the sum of input vectors that have the closest similarity to each neuron. To further illustrate this process, a 2x2 subgrid of neurons is detected to be a cluster. A graphical representation of this cluster is shown in Figure 8.2. Performing horizontal calculation on rows $r1$ and $r2$, summing the number of victories those neurons have been awarded during the current iteration results in finding that the first row of neurons have a higher winner concentration. Therefore, we know that the centroid neuron for this cluster will be one of the two neurons on the top row. Doing the same process for the vertical results in finding that the second column has a higher winner concentration. Using these

Figure 8.2. A cluster of four neurons.



horizontal and vertical coordinates, we have found a good representing neuron to be our centroid for this cluster, namely neuron $w_{r1,c2}$.

With the calculation of known neuron clusters and methods of defining and representing them, a unique opportunity presents itself. As iterations progress and both the learning rate α and neighbourhood distance d decay, the possibility of significant change in neuron representation of input decreases. We can now decide to invoke cluster-based representation and visitation of the SOM network in the competition layers. This means that instead of searching and computing similarity measures for all n neurons, we can find the cluster K_i whose centroid neuron weight vector is closest to the input. Once this occurs, we can locally search the neurons in that particular cluster for the winner. This reduces the search space and thus computation time significantly. That is, the total distance calculations for an iteration is given by m input vectors $\times$ n neurons.

Invoking cluster-based representation means k cluster centroids (for all K clusters) will result in one cluster K_i^* to be chosen as the search space. The number of distance calculations for each input is then $k \times |K_i^*|$. For the entire iteration, it concludes that,

$$m \times k \times |K_i^*| \leq m \times n, \tag{8.9}$$

since $k \times |K_i^*|$ will always be less then or equal to n. If $n = k \times |K_i^*|$, then $k = n$ and $|K_i^*| = 1$. Each neuron is its own cluster. Hardly the correct decision to invoke cluster-based representation of a SOM.

A picture of how SOM quality (correct cluster density and cluster distances) can be gathered using the above methods. Using the ability to calculate physical distance between clusters on the feature map as well as the ability to calculate the density of a particular cluster can enable a new algorithm to determine which neuron is best suited to be

updated. These quality measures can be used together to see how much a particular neuron, if updated, can improve the overall quality of the feature map.

Measuring SOM Quality. A question that must be asked is how dense should a cluster be in a SOM? In order for a feature map to adequately represent the input, the density of a cluster should somehow be proportionate to the ratio of the input it represents and the entire input data set.

Let P_{K_i} be the set of input that is associated with the neurons present in K_i, the cluster to be measured,

$$P_{K_i} = \{\vec{p}_j | \vec{p}_j \mapsto w_i, w_i \in K_i\}, \tag{8.10}$$

where $\vec{p}_j \mapsto w_i$ implies $\vec{p}_j$ association by $\vec{w}_i$. w_i was winner in competition for $\vec{p}_j$ during the current iteration.

The number of input vectors in P_{K_i} is given as $|P_{K_i}|$. Therefore, Equation (8.11):

$$I_{K_i} = \frac{|P_{K_i}|}{|P|}, \tag{8.11}$$

is called the total input-to-cluster ratio for cluster K_i. If the ratio between the density of a cluster to total number of neurons approaches the total input-to-cluster ratio, the density and size of the cluster are correctly proportionate to the number of inputs it represents.

The density of a cluster is the number of input vector associations versus the number of neurons in the cluster shown in 8.8. This density over the number of neurons in the network will give us the correct input-to-cluster neurons-to-total neurons ratio shown in Equation (8.12).

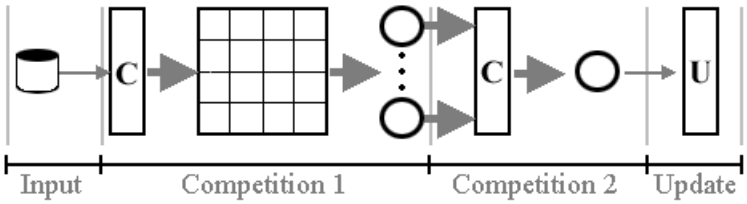
$$T_{K_i} = \frac{D_{K_i}}{|W|}, \tag{8.12}$$

where W is the set of neurons in our map. We wish to ensure Equation (8.12) is as close as possible to our total input-to-cluster ratio in Equation (8.11). It follows,

$$\frac{|P_{K_i}|}{|P|} \simeq \frac{D_{K_i}}{|W|} \tag{8.13}$$

$$\simeq \frac{|P_{K_i}|}{|K_i| \times |W|}. \tag{8.14}$$

Figure 8.3. The layers of GTSOM including the addition of another competition layer used during the training process.



Equation (3.23) has been dubbed *density unification* to show the target density of all clusters should be similar to the total input-to-cluster ratio.

Game-Theoretic Competitive Learning in SOM

Although individual neurons have the ability to improve their situation during each competition, a collective goal for the entire SOM is not considered. We have found that this is one problem in traditional SOM theory. Individual neurons that are updated have no indication on whether the entire network benefits from the process.

The transition between local optimization techniques to those of global optimization must occur in order to solve problems of density mismatch and physical adjacency errors. The concept of overall SOM quality must be defined in order to progress to a state in which properties between overall neuron relationships and input vectors can be measured.

The GTSOM Training Model. With the ability to precisely define neuron clusters within a SOM, measures can be used in order to define overall quality of the network. These measures, such as the size of clusters, the distance between clusters, and the appropriate cluster size to represent input can be combined to give a certain payoff value to a particular neuron, if chosen as a winner. The new training model is called GTSOM, or Game-Theoretic Self-Organizing Maps. This new model architecture is shown in Figure 8.3.

The GTSOM model consists of four layers, one more than the traditional model found in Figure 8.1 on page 6: the input layer, the first competition layer, the second competition layer using game-theoretic concepts, and the update layer. No changes were made to the input layer which still governs the presentation of input vectors in the compe-

tition layer(s). We added an additional competition layer to handle the game-theoretic aspects of the training.

When the competitive phase begins, a ranking can be associated with each neuron according to its distance from the input vector. Using the ranked list of neurons, a new competition layer is constructed in order to determine which neuron and which strategy or action should be taken.

The first Competition layer is modified so that instead of determining which neuron is most similar to the current input vector, the layer now ranks neurons according to each similarity measure obtained. There is an opportunity here to include a dynamic, user-defined threshold value t_1 that can deter any neurons that are beyond a certain similarity measure to be included in the ranked set as shown in Equation (8.15) and Equation (8.16):

$$W' = \{n_1^*(q), \ldots, n_n^*(q)\} \ , \tag{8.15}$$

where $\forall n_i^*(q) \in W$,

$$|\vec{w}_i^*(q) - \vec{p}_i| \leq t_1 \ , \tag{8.16}$$

and $1 \leq i \leq n$. This allows the user to specify a degree of minimum similarity desired when having the first competition layer computing which neurons should enter the second competition layer.

This ranked set neurons is the main gateway of communication between the two competition layers. The second competition layer uses this set to perform its special instructions.

We are starting to see competitive learning aspects of SOM being expanded into a more complex, multi-tiered competition system. The use of game theory for added decision making ability is needed because of the increase of complexity of the competition between neurons.

The Game-Theoretic Competition Layer. Once a ranked set of neurons has been created, the second competition layer starts to create competition tables of the form shown in Table 8.1. A neuron n_i^* with possible actions $S = \{a_{i,1}, \ldots, a_{i,r}\}$ and payoffs calculated from corresponding utility functions $U = \{\mu_{i,1}, \ldots, \mu_{i,r}\}$ competes against neuron n_j^* with the same action and utility sets. The neuron whose specific action $a_{i,k}$ results in the greatest overall SOM quality is chosen to be the winner. Table 8.1 shows a payoff result for each neuron using respective actions. For example, $< \mu_{i,1}, \mu_{j,1} >$ is the payoff for neuron n_i^* using action $a_{i,1}$ versus the payoff of neuron n_j^* using action $a_{j,1}$. We wish to

Table 8.1. Payoff table created by second Competition layer.

		$n_j^*(q)$		
		$a_{j,1}$	...	$a_{j,r}$
$n_i^*(q)$	$a_{i,1}$	$< \mu_{i,1}, \mu_{j,1} >$	...	$< \mu_{i,1}, \mu_{j,r} >$
	$\vdots$	$\vdots$	...	$\vdots$
	$a_{i,r}$	$< \mu_{i,r}, \mu_{j,1} >$	...	$< \mu_{i,r}, \mu_{j,r} >$

look at this table and find the neuron whose payoff or increase in SOM quality is largest.

With the addition of quality measures, neurons are now ranked in partial order. For example, a particular neuron n_i^* could have a higher ranking than n_j^* in terms of a particular similarity measure between itself and the input vector, but the neuron may not have that same ranking when additional quality measures are taken into account.

A ranked list of neurons created with input similarity as a focus could be different then a ranked list of neurons created with both similarity and cluster size taken into account. Likewise for lists created with similarity to input and cluster density. The second competition layer must take into consideration not only similarity to input, but also how much each neuron can increase or decrease feature map quality. Many different ranking of neurons in W' can occur when more than one measure is used.

There are two possible ways of creating tables to govern the second phase of competition. First, neurons can be initially paired randomly with each other. Victors of each "round" move on to the next round, where new tables are created for the neurons that have been awarded victories. This process proceeds until a total victory is declared for one neuron. Second, for a set $W = \{n_1^*(q), \ldots, n_n^*(q)\}$ of ranked neurons, an *n-dimensional* payoff table can be created. With n neurons ranked and entering competition, each with r possible actions, a total of r^n cells must be observed to determine which neuron gives the best quality or utility value for this iteration.

SOM Update Strategies. Actions performed by a particular neuron could possibly include parameters such as adjustable learning rates or adjustable neighborhood size. Such actions can be called *strategies* to describe an action that can be modified in order to create new actions.

A strategy of adjusting the learning rate α can be modified so that there is an action for an increased adjustment, decreased adjustment, and a no-change scenario. This strategy can improve clusters by forcing

subsequent input vectors that are similar to the current input to have a greater possibility to be more similar to a different neuron than it did on a previous iteration in the case of an increased learning rate. That is, the input vector will have an increased likelihood to be closer to a different neuron next iteration. A decreased learning rate will result in a diminished similarity adjustment between the victor and the current input vector, resulting in negligible change from subsequent iterations.

A set of actions detailing neighborhood size for a particular neuron is useful when cluster sizes are desired to either grow or diminish. An increased neighborhood size will modify a larger number of neurons to become more similar to the current input vector. This will result in less dense clusters if desired. In contrast, a decreased neighborhood size could have an exact opposite effect, decreasing the size and increasing the density of clusters. If clusters are too far apart, the density of a particular cluster could be diminished so that cluster boundaries become closer. Also, if clusters are too compact, the density of some clusters could be increased in order to increase distance between centroids.

GTSOM Implementation

The process of ranking neurons according to similarity, creating payoff tables, and determining winning neurons is introduced. Training will stop when either of the following three conditions is met.

1 If a maximum number of specified iterations have been performed.

2 If no neurons have won competitions for new input vectors that were not won before during previous iterations.

3 If the overall quality of the SOM has reached or moved beyond that of a user-defined threshold.

A traditional SOM stops training when either conditions of the first two conditions is met. With the addition of the third condition, training time can be reduced if a certain quality has been reached. A lower threshold will most likely result in a lower number of iterations performed. As precision increases with respect to the number of iterations performed (smaller learning rate), a lower number of iterations will result in the algorithm completing with a learning rate above that of the final desired learning rate.

A large value for t_1 will result in increased computation time as it will result in a larger W'. Since tables are created and observed for each distinct pair of neurons within W', the similarity threshold must

be considered carefully. If t_1 is too small, it will result in incomplete competition, where neurons that may offer valuable actions could be ignored based on their dissimilarity to the current input vector.

The threshold t_2 gives the option of stopping the training process when a certain overall SOM quality has been reached. If t_2 is too high, representing a high quality preference, will result in no computational efficiency improvement. This threshold may never be reached before maximum iterations have occurred.

If t_2 is too low, it could result in too few iterations being performed. Since the learning rate α is adjusted during each iteration, it will not get an opportunity to become sufficiently small for precise weight vector updating.

---

for each neuron $n_i \in W$
{
  Initialize $\vec{w}_i$ randomly ;
}
while $(q \leq q_m)$ or $(\forall\, \vec{p}_i \in P, n_i^*(q) \neq n_i^*(q-1))$ or $(\mu(A) \leq t_2)$
{
  $\alpha_q =$ adjusted α_{q-1} for iteration q ;
  $d_q =$ adjusted d_{q-1} for iteration q // neighborhood distance ;
  for each $\vec{p}_k \in P$
  {
    Find set $W' = \{n_1^*(q), \ldots, n_n^*(q)\}$;
    for each $< n_i^*(q), n_j^*(q) >$ pair in W'
    {
      $T_{i,j} = (N, S_{i,j}, F_{i,j})$, where
      $N = \{n_i^*(q), n_j^*(q)\}$,
      $S_{i,j} =$set of actions for $n_i^*(q)$ and $n_j^*(q)$,
      $F_{i,j} =$set of utility functions returning quality of A.
      $\alpha_q = \pm a_i^*$, where $a_i^* =$the action that best improves A. ;
    }
    Choose $n_q^*(\vec{p}_i)$ whose utility μ_i has maximum payoff action ;
    Update_w($n_i^*(q), \vec{p}_k, \alpha_q$) // update winning neuron ;
    Update_N($N_{n_i^*(q)}(d_q), \vec{p}_k, \alpha_q$) // update neighborhood of n^* ;
  }
}

---

4. GTSOM Algorithm Analysis

This section analyzes the GTSOM algorithm presented in Section 3 on page 19. The main disadvantage to the algorithm is the decreased running time, due to additional complexity required for finding the similarity ordering and calculating payoff matrices.

SOM and GTSOM Comparison

The training of the traditional SOM and new GTSOM was performed on a color dataset [2]. We will be looking at two different types of results from our training experiments when determining whether the GTSOM algorithm is performing to our expectations. These two comparisons will be runtime between SOM and GTSOM and the quality of the map during the training of SOM and GTSOM. There are four different scenarios to discuss when looking at findings, these are:

1 SOM vs non-cluster-based GTSOM over maximum iterations.

2 SOM vs cluster-based GTSOM over maximum iterations.

3 SOM vs non-cluster-based GTSOM over with user-defined quality threshold.

4 SOM vs cluster-based GTSOM over with user-defined quality threshold.

Runtime Comparison. Runtime finding demonstrate the total length of time taken to train a SOM. Scenario 1 is shown in Table 8.2. 1000 iterations are performed for each method on a SOM with 100 neurons.

Table 8.2. Scenario 1, maximum iterations, runtime in seconds (lower is better), 100 neurons.

Method	q_m	$\|W\|$	runtime(seconds)
SOM	1000	100	392
GTSOM	1000	100	491 (+20.1%)

Without cluster-based representation and quality thresholds defined by the user, the traditional SOM training algorithm outperforms the new GTSOM method by 20%. This is mainly due to the creation of payoff tables between neurons and the added complexity of the algorithm.

Scenario 2 is shown in Table 8.3. 1000 iterations are performed for each method on a SOM with 100 neurons.

Table 8.3. Scenario 2, maximum iterations, runtime in seconds (lower is better), 100 neurons.

Method	q_m	$\|W\|$	runtime(seconds)
SOM	1000	100	385
GTSOM	1000	100	362 (-5.9%)

With cluster-based representation, the amount of neurons visited during competition layer 2 in GTSOM is dramatically reduced. This improves performance over SOM by 5.9%, plus or minus 1.8%. This improvement in performance will only increase with an increase of iterations.

For scenario 3, the user-defined quality threshold will be the quality reached at the maximum last iteration performed by SOM. That is, if a quality measure of $\mu(q_m)$ is achieved by SOM at the end of training, GTSOM will stop once it reaches that threshold. Findings are shown in Table 8.4. 1000 iterations are performed for the SOM method. The GTSOM method reached the target quality $\mu(q_m)$ in 823 iterations. The SOM had 100 neurons in the network.

Table 8.4. Scenario 3, user-defined quality threshold, runtime in seconds (lower is better), 100 neurons.

Method	q	$\|W\|$	$\mu(q_m)$	runtime(seconds)
SOM	1000	100	73	386
GTSOM	823	100	73	319 (-17.4%)

A significant improvement of 17.4% is seen for the GTSOM method. This is due to the fact that the method reached $\mu(q_m)$ 19.4% faster than the traditional SOM approach. A decreased number of iterations performed will decrease runtime. It is worth mentioning that the map made by GTSOM was not the same as the map made by SOM.

Scenario 4 is shown in Table 8.5. 1000 iterations are performed for the SOM method. The GTSOM method reached the target quality $\mu(q_m)$ in 778 iterations. The SOM had 100 neurons in the network.

A 22.8% decrease in runtime was measured when using cluster-based GTSOM and a user-defined threshold. The combination of fewer iterations and few neuron visits have given a significant performance increase versus SOM.

GTSOM has increased performance in all but one test in regards to runtime. These results show that even though complexity was added

Table 8.5. Scenario 4, user-defined quality threshold, runtime in seconds (lower is better), 100 neurons.

Method	q	$\|W\|$	$\mu(q_m)$	runtime(seconds)
SOM	1000	100	73	390
GTSOM	778	100	73	301 (-22.8%)

to the methods of training, opportunities now exist to decrease runtime while remaining at a consistent quality.

Quality Comparison. We will be looking at three main quality characteristics findings: final distance between clusters, final cluster density and resulting a difference in unification, and overall quality of the final trained map.

First, the distance between clusters should be at the largest level possible since the main clusters signify primary colour groupings (red, green, and blue). Results are shown in Table 8.6.

Table 8.6. SOM vs GTSOM, $|k|$ = number of clusters, average distance (higher is better).

Method	$\|k\|$	Average Distance
SOM	4	4.9
GTSOM	4	5.2 (+5.8%)

The GTSOM algorithm results in a 5.8% increase in distance between clusters, a marginal improvement.

Second, the density of clusters should be as close to unification as possible with the actual input classification. Results are shown in Table 8.7.

Table 8.7. SOM vs GTSOM, density is inputs/neuron, difference from unification (lower is better).

Method	Average neurons per cluster	Average Density	Difference from Unification
SOM	18	14.8	-18.4%
GTSOM	16	16.6	-11.2%

The GTSOM algorithm results in improved density of the map, as unification is 39% closer to optimal. More iterations would improve map density.

Third, the overall quality of the map is taken into consideration. The results are shown in Table 8.8.

Table 8.8. SOM vs GTSOM overall quality (higher is better).

Method	Average Distance	Difference from Unification	Overall Quality
SOM	4.9	-18.4	73.0%
GTSOM	5.2	-11.2	114.9 (+63.5%)

A 5.8% increase in distance between clusters and a 39% improvement in unification results in a significant 63.5% increase of overall quality of the map.

Results show that runtime is decreased for three of the four scenarios, with a maximum improvement of 22.8% in scenario 4.

5. Conclusion

A SOM is a proven method for discovering similarities within a data set. By performing distance measures between neuron weight vectors and input vectors, a SOM can manipulate itself in order to represent some of the patterns it finds. Although the method works well with many types of data, local optimization occurs when having a one-to-one comparison (one weight vector compared to one input vector).

In order to create a new type of SOM that is globally optimized to input, multiple criteria is used to find a neuron that not only represent the current input, but also ensures that the entire network improves in representation. Cluster sizes, densities, and distances are used in conjunction with input similarity to improve SOM quality. The added measures remove the local optimization problem by looking at the entire map and how it is performing.

The new competitive learning routines make use of game theory to decide which neuron is chosen to represent current input. This required the creation of additional metrics in order to measure map quality during training. The notion of *density unification* ensures that neuron clusters accurately portray the actual data. The use of game theory facilitated the demonstration of possible strategies to improve SOM quality, including dynamic decaying learning rates and neighbourhood sizes. This use of game theory as an infrastructure ensures global optimization during training by picking neurons that can improve overall SOM quality.

We have proposed a new approach to competitive learning in SOMs called GTSOM. The opportunity to create a model to facilitate global

optimization of the feature map requires methods to acquire the overall quality of the feature map. These methods take the form of measuring distance between clusters, cluster density and cluster size. GTSOM allows for global optimization using multiple criteria for choosing neurons. The modification of the first competition layer to sort neurons according to similarity to input and the addition of the second competition layer for the game-theoretic aspects of training was presented.

The usefulness of the new GTSOM training model for classification purposes was demonstrated using colour vector data. Results show that runtime is decreased by a factor of 22.8% and a 63.5% increase of overall quality of the map. This is a simple example of how our approach can improve classification applications such as image recognition. This made use of neuron cluster detection and representation within the SOM that was introduced in this work.

References

[1] Herbert, J., Yao, J.T.: A game-theoretic approach to competitive learning in self-organizing maps. In: Proceedings of the First International Conference on Natural Computation . Volume 1. (2005) 129–138

[2] Herbert, J.: A new approach to competitive learning in self-organizing maps. Master's thesis, University of Regina (2006)

[3] Kohonen, T.: Automatic formation of topological maps of patterns in a self-organizing system. In: Proceedings of the Scandinavian Conference on Image Analysis. (1981) 214–220

[4] Huntsberger, T.L., Ajjimarangsee, P.: Parallel self-organizing feature maps for unsupervised pattern recognition. International Journal of General Systems **16**(4) (1990) 357–372

[5] Tsao, E., Lin, W.C., Chen, C.T.: Constraint satisfaction neural networks for image recognition. Pattern Recognition **26**(4) (1993) 553–567

[6] Brachman, R.J., Anand, T.: The process of knowledge discovery in databases: A human-centered approach. In: Advances in knowledge discovery and data mining. (1996) 37–58

[7] Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery: an overview. In: Advances in knowledge discovery and data mining. (1996) 1–34

[8] Herbert, J., Yao, J.T.: Time-series data analysis with rough sets. In: Proceedings of the Fourth International Conference on Computational Intelligence in Economics and Finance. (2005) 908–911

[9] von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, Princeton (1944)

[10] Haykin, S.: Neural Networks: A Comprehensive Foundation - Second Edition (1994) 30

[11] Hagan, M.T., Demuth, H.B., Beale, M.H. In: Neural Network Design. PWS Publishing Company (1996)

[12] Fudenberg, D., Tirole, J., Game Theory. The MIT Press (1991)

[13] Chandrasekaran, V., Liu, Z.: Topology constraint free fuzzy gated neural networks for pattern recognition. IEEE Transactions on Neural Networks **9**(3) (1998) 483–502

[14] Pal, S.K., Dasgupta, B., Mitra, P.: Rough self organizing map. Applied Intelligence **21**(3) (2004) 289–299

[15] Nash, J.: The bargaining problem. Econometrica **18**(2) (1950) 155–162

[16] Roth, A.: The evolution of the labor market for medical interns and residents: a case study in game theory. Political Economy **92** (1984) 991–1016

[17] Bell, M.G.F.: The use of game theory to measure the vulnerability of stochastic networks. IEEE Transactions on Reliability **52**(1) (2003) 63–68

[18] Fischer, J.,Wright, R.N.: An application of game-theoretic techniques to cryptography. Discrete Mathematics and Theoretical Computer Science **13** (1993) 99–118

[19] Gossner, O.: Repeated games played by cryptographically sophisticated players. Technical report, Catholique de Louvain - Center for Operations Research and Economics (1998)

[20] Fritzke, B.: Some competitive learning methods. Technical report, Institute for Neural Computation. Ruhr-Universit at Bochum (1997)

[21] Blackmore, J., Miikkulainen, R.: Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In: Proceedings of the International Conference on Neural Networks. Volume 1. (1993) 450–455

[22] Santini, S., Jain, R.: Similarity measures. IEEE Transactions: Pattern Analysis and Machine Intelligence **21**(9) (1999) 871–883

[23] Kolen, J.F., Pollack, J.B.: Back propagation is sensitive to initial conditions. In: Advances in Neural Information Processing Systems **3** (1991) 860–867