

Distributing SOM Ensemble Training using Grid Middleware

Bogdan L. Vrusias, Leonidas Vomvouridis, and Lee Gillam

Abstract—In this paper we explore the distribution of training of self-organised maps (SOM) on Grid middleware. We propose a two-level architecture and discuss an experimental methodology comprising ensembles of SOMs distributed over a Grid with periodic averaging of weights. The purpose of the experiments is to begin to systematically assess the potential for reducing the overall time taken for training by a distributed training regime against the impact on precision. Several issues are considered: (i) the optimum number of ensembles; (ii) the impact of different types of training data; and (iii) the appropriate period of averaging. The proposed architecture has been evaluated in a Grid environment, with clock-time performance recorded.

I. INTRODUCTION

Haykin has described a neural network as a “massively parallel distributed processor” [1]. Haykin’s description suggests that an ANN can be composed from separately trained partitions. The partitioning and distributed training presents challenges for topologies and algorithms that characterise ANNs, and influences the training regimes and the operation of the networks. Consideration needs to be made, for example, of whether a training regime should be *batch* (where weights are only updated after all of the inputs are presented) or *incremental* (where weights are updated after the presence of each input). This choice determines how the data are presented to the network and at what stage the network undertakes its training phase. At a desirable degree of precision against the test datasets, the ability to generalise, a fundamental success criterion for training most neural networks, can be assessed. Though the purpose of training is to achieve the best possible precision, output is always an approximation of the desired behaviour. Training techniques such as *bootstrap aggregating*, [2], or *boosting*, [3], are employed in training *ensembles* [4] (i.e. sets of identical neural networks). Such approaches demonstrate the range of issues that need to be addressed when training ensembles concurrently using different subsets of the input datasets. One issue, in particular, is in encompassing the training of all members of the ensemble: after each network has trained on its subset, averaging produces a new set of

identical networks that will have incorporated, in some ways, training from all other networks. This process tends towards less precision than having a single network trained on all available data, but more training can be performed in less overall time.

The distributed approach potentially requires the exchange of large amounts of data (input data and network states), but a variety of parameters can be selected that are more appropriate to a given configuration. Use of relatively large training increments, for example, could be more suited for distribution of ensembles over Grid infrastructures [11], with the amount of interaction limited to exchanging states at specified intervals. This would reduce the potential impacts of communication latency, which the use of low-latency computer networks may be able to limit further. Over low-latency networks, however, smaller training increments may be better for achieving desired precision more rapidly, and partitioning may be more suitable. The assumption is that quality of service, for the infrastructure, is known and can be guaranteed. A Grid that provides access to a variety of high-throughput and high-performance systems appears to provide a good environment for ANN experiments.

An ensemble approach has the potential for greater precision than a single neural network, as more training can be undertaken within the same time. Such an approach could be applicable to a large number of ANNs, however making the sum equal to the total of the parts is not necessarily possible, and here we may have to consider a trade-off of speed against precision. In our architecture we make a differentiation between *iteration* and *step*: iteration is a single repetition of all available training data inputs (i.e. an iteration trains an SOM with one or more data inputs); while a training step is the point where we collect all the outputs of dispatched SOMs and perform the weight averaging (i.e. a step trains the SOM with one or more iterations). Drawbacks in an ensemble approach relate to network size and slowest process. Large networks place demands on memory requirements often exceeding available physical RAM, and since ensemble members replicate the network size, this becomes an issue for every machine. Furthermore, the averaging step must wait for the slowest ensemble member to have completed its cycle before continuing, unless the ensemble is constructed such that members can be interrupted, interrogated and averaged at regular intervals.

Efficient training of a neural network has been considered using compute clusters for distributing the training of multi-layer perceptrons, self-organising maps and radial-basis function networks [5]. This approach serves as an inspiration

Manuscript received January 31, 2007. This work was supported in part by the EPSRC REVEAL project under Grant No.GR/S98450/01 and the EU LIRICS project under Grant No. 22236.

B. L. Vrusias is with the Department of Computing, University of Surrey, UK (phone: +44 1483 682261; fax: +44 1683 686051; e-mail: b.vrusias@surrey.ac.uk).

L. Vomvouridis, was with the Department of Computing, University of Surrey, UK (e-mail: csm1lv@surrey.ac.uk).

L. Gillam is with Department of Computing, University of Surrey, UK (e-mail: l.gillam@surrey.ac.uk).

for the work presented in this paper. Other related work has been taken into account; the main approaches to tackle this challenge include: (i) construction of a distributed back-propagation algorithm [6], (ii) topologies for the message-passing parallel system, namely the ring, two-dimensional torus, binary tree, hypercube and extended hypercube topologies [7]; (iii) distribution of training of self-organising maps in a parallel virtual machine (PVM) [8]; (iv) experimental results of a 3-node-type architecture in distributing the training of a standard back-propagation neural network on a Myrinet cluster [9] (v) transformation of the problem of distributing the training of a multi-layered perceptron, to a linear algebra problem (QR factorisation), solved by numerical methods, using a distributed linear algebra library implementation in a Grid environment [10]. Such research attempts to overcome the network size bottleneck, however this comes with a cost: latency and throughput of the underlying computer network remain of crucial importance, hence an emphasis on clusters or local area Grids. In distributing or parallelising most software systems, the goal is usually to produce identical results to the sequential algorithm. Requiring an approximation of the desired behaviour, perhaps by defining an acceptable precision, alleviates this limitation for ANNs.

II. GRID NETWORKS AND ARTIFICIAL NEURAL NETWORKS

The notion of having many powerful processors concurrently working on the same problem, to achieve higher performance, has evolved into a major discipline variously referred to by terms such as high performance computing (HPC), high end computing (HEC), technical computing and parallel computing. Grids have recently been defined as “distributed computing performed transparently across multiple administrative domains” [11] and have been applied to large-scale high-complexity problem solving, such as protein folding, financial modelling, earthquake simulation, and climate/weather modelling [12]. So-called “Grid middleware” such as Globus has been developed to address issues of resource management, security and data exchange among disparate heterogeneous systems and networks [13]. Grid computing is distinguished from conventional distributed computing, “by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation” [14]. However, Globus does not provide a scheduler for easy deployment of a distributed application. A popular job scheduling and management systems that can be used in the Globus framework is Condor [15], and the experiments described in this paper made use of a Condor pool in the University of Surrey’s Department of Computing that provides upwards of 150 processors.

The experiments described relate to ongoing research also in the Department of Computing. In particular, the REVEAL project (EPSRC Grant GR/S98450/01) utilises self-organising maps for video annotation [16]. The training of these maps has proved to be cumbersome, due to their relatively

large size, and large amount of input data, which necessitates an efficient and cost-effective performance. The objective of this research is to examine performance, in terms of training time and accuracy, in the distributed training of the self-organising maps [17]. The Condor pool and related Grid infrastructure were originally composed for the concurrent analysis of high-frequency data in finance, including both instrument data and financial news [18].

The relative novelty of the Grid approach to distributed computing, in contrast with a relative maturity in artificial neural network research, has resulted in a limited volume of research in efficiently distributing the training of artificial neural networks on Grid systems. This paper proposes a two-level architecture for the experimental system and discusses the experimental methodology to determine an optimal set of parameters for this architecture. The architecture takes into consideration the fact that a Grid may be composed of both local cluster environments, sometimes over high-throughput low-latency networks, and across systems distributed over a medium-throughput but higher-latency Internet. Differences in latency suggest that on a local-area level it could be advantageous to adapt a partitioning scheme, while on a wide-area level ensembles could be more promising. In the local area level, each self-organising map would be partitioned and distributed among training nodes of that particular local area network by the local dispatcher, while training data is stored locally. The wide area dispatcher submits requests to wide area training nodes, which won’t actually carry out any training themselves, but will be submitting requests to local dispatchers, and collecting the resulting neural network states from local dispatchers. Wide area training nodes submit the results to the wide area dispatcher, which will conduct the averaging, store the resulting current network state, and continue to the next step.

An efficient strategy for distributing the training of self-organising maps has been described elsewhere [5]; here a (large scale) self-organising map is partitioned into smaller sub-maps, which the dispatcher (master) submits to training nodes (slaves). Although this proved relatively successful on dedicated clusters, the time elapsing from a node transmitting its local winner, until it receives the global winner, may prove comparable to the duration of its local training cycle. This will be dependent on the degree of partitioning, and the variance in resource availability on each node, but will impact the training time for the SOM.

We propose that in the wide area level, instead of partitioning a (large) SOM into smaller pieces for each node to train, identical copies of the (large) SOM are distributed to each node, but each node is presented with a different “batch” of input datasets. “Multicasting” (broadcasting to interested participants) could be beneficial, not for the input data, but for the current global neural network state, at the beginning of each step. Local dispatchers can carry out the actual training and return the results to the wide area level nodes, which in turn submit resulting neural network states to

the wide area dispatcher. At the end of this high-level training step, weight averaging will be performed at the wide area dispatcher. Large amounts of data need to be transported during the steps of the training procedure, and data compression techniques may be employed to reduce the volume of transported data, with a relatively small computational overhead.

Although the experiments performed here are on a local level, there is potential for wide area distribution of training. The main purpose of the wide-area level is to distribute training among geographically distant networks, so that potentially partitioning the network, which requires low network latency, may be performed within the local area network of each member of the ensemble.

III. TRAINING SELF-ORGANISING MAP ENSEMBLES

An appropriate set of training data is one of the primary considerations with any neural network experimentation. Although averaging has proven advantageous in the case of MLPs, there has been no relevant research as regards SOMs, therefore a simple data suite should be utilized for this preliminary phase of experimentation, to determine the extent of applicability (if any) of averaging SOM ensembles.

A. Training Dataset

For the purpose of the experimentation we used two artificial datasets: an existing dataset and a dataset that we produced. The first dataset is a collection of topologically different series of datasets, the ‘‘Fundamental Clustering Problem Suite’’ (FCPS) [19], generated to verify that a newly invented clustering algorithm functions properly by correctly clustering a series of datasets with known classification. Each FCPS datasets is designed to address a specific category of problems commonly encountered in the development of clustering mechanisms. However, the datasets are generally small (300 points, 3 dimensions, 2 categories is a typical example). For this size of problem, a single computer performs adequately, training the network in a few seconds, or even less. Therefore there is little specific benefit from distributing the training of these problems, however, if the averaging of ensembles allows such problems to be solved with a satisfactory precision, it would be a good indication that the proposed technique might prove applicable to larger problems as well.

For the second artificial dataset, a program was implemented to generate random points, in the proximity of the vertices of an n -dimensional hypercube. The n -dimensional hypercube is a regular convex n -polytope, whose boundary consists of regular convex $(n-1)$ polytopes where n is the number of dimensions. For $n=3$, the shape is a cube (3-d) whose boundary consists of squares (2-d). For this type of layout the number of categories (or classes) of input patterns, is 2^n . Input points are generated with an n -dimensional Gaussian distribution where the centre of the distribution for all input points of the same class is the vertex

of the hypercube that corresponds to that class. For small dimensions, a reasonable SOM may be used (e.g. 10x10 for 3-d inputs), however for larger dimensions, the SOM sizes increase substantially, assuming that each class will need a minimum of a 3x3 area on the map in order to classify successfully (TABLE 1). The hardware available imposed a 10-d ceiling in the experiments, since for 10-d, presenting each input vector 100 times, resulted in training times in the order of magnitude of 5×10^4 sec (13h).

TABLE I
HYPERCUBE DATASET

Number of dimensions	Total input patterns	SOM size	SOM memory requirements
3	800	10x10	21K
4	1600	13x13	56K
5	3200	18x18	140K
6	6400	25x25	335K
7	12800	35x35	780K
8	25600	50x50	1.8M
9	51200	70x70	4M
10	102400	100x100	8.8M

Hypercube vertex Gaussian input datasets (with 100 patterns per class)

B. Experimental setup

All experiments were based on two existing SOM implementations: the first (MMUC system) from the University of Surrey [20]; the second (SOM_PAK) from the Helsinki University of Technology [21].

Three dispatchers were coded: one for a traditional single-process training, to create the benchmark, one to spawn local processes on a single machine, to test the averaging algorithm’s behaviour, and also for running on a shared-memory multiprocessor machine, and finally the Grid version of the dispatcher. The approach for implementing the SOM ensemble can be summarised by the following stages:

- STAGE 1: Weight initialisation; calculation of initial error
- STAGE 2: Dispatching of SOMs with input data, initial weights and training parameters
- STAGE 3: Training for a number of iterations
- STAGE 4: Collecting the resulting weights, averaging, and calculating the MQE
- STAGE 5: Until training is completed, goto STAGE 2

The purpose of the experiments was to compare the efficiency of an ensemble of SOMs to that of a single network, with identical parameters. Correct choice of learning rate decrement plays an important role in efficiently training a neural network, and some treatment is provided, however an exhaustive treatment of values and decrement profiles of the learning rate is beyond the scope of this paper. For each training cycle, the learning rate value follows a linear transformation of the step number, in the range [0, 1]. Provided the number of steps is sufficiently large (at least 10), the approximation of the natural exponential curve by a number of chords is satisfactory.

Averaging is performed by calculation of the arithmetic

mean of each dimension for each node of the SOMs in the ensemble. Averaging is performed after a fixed number of training iterations, with the number specified on start-up as the total number of training iterations, divided by the number of averaging steps, divided by the number of networks in the ensemble. If communication time and scheduling latency were zero, the system would exhibit a speedup equal to the number of networks in the ensemble.

IV. EXPERIMENTATION

Each of the following experiments was conducted 20 times, and an average taken to discount inappropriate results due to initial random weights and random selection of training data subsets. The *mean quantisation error (MQE)* was considered, for the same dataset where the network was trained on, to calculate the training efficiency. MQE is calculated by taking the weighted sum of the activation function for all input signals. We outline the five main experiments here:

A. Number of ensembles for 4-d hypercube problem

This experiment attempts to determine the extent to which averaging affects training. A 15x15 SOM was trained, with a linear decrement of the learning rate from 0.5 to 0.01, and a linear decrement of the Gaussian neighbourhood radius from 8 to 1. All training parameters, as well as the input dataset and initial state of the network, were retained among all experiments, to ensure comparability of results.

Fig. 1 suggests that, if the latency of distributing members of the ensemble, periodically averaging weights, and synchronising training is very small compared to the computational duration of each step, then distributing an averaging SOM ensemble would be beneficial for training times and/or accuracy. The traditional method for decreasing a SOM's MQE has been to increase the duration of the training process. With averaging, the SOM is exposed to a larger number of training examples, but in a manner that can be easily distributed across a wide-area Grid. But, this is not as efficient as exposing the SOM sequentially.

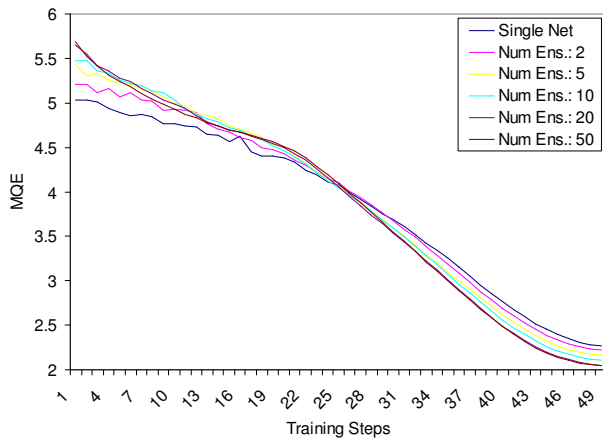


Fig. 1. MQE for distribution on ensembles of various sizes. Average for 20 runs of 200 iterations per step per ensemble member, for a 15x15 SOM trained on 100 points per vertex of a 4-d hypercube

Fig. 2 shows a single network, trained on a total number of steps that is equal to the corresponding distributed cases illustrated in Fig. 1. The difference is that while a monolithic and a 50-ensemble network theoretically take the same time to compute, the latter case performs 50 times more training than the monolithic scenario, and therefore the result is better in terms of the final value of MQE to which the training process converges.

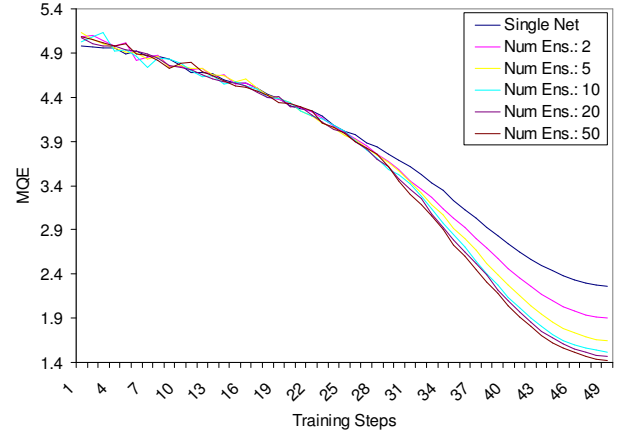


Fig. 2. MQE for distribution on a single 15x15 SOM. Average for 20 runs of 1000 iterations per step per ensemble member.

B. Number of ensembles for FCPS problem

To further examine the potential for SOM ensemble averaging, experiment A was repeated for the 10 datasets in the FCPS dataset. Learning rate and neighbourhood radius were decreased linearly, while the ensemble size ranged from 1 to an ensemble with 50 members. For all datasets of the FCPS, a rectangular topology with 15x15 nodes is used.

Fig. 3 shows only one of the many datasets tested, and suggests that the application of averaging is indeed beneficial, since as the number of members in the ensemble increases, MQE decreases, in relation to a single network that would take the same time to compute. The remaining datasets show more or less the same. It seems that the most efficient size for an ensemble is 5 to 10 networks; beyond that the relative benefit is limited.

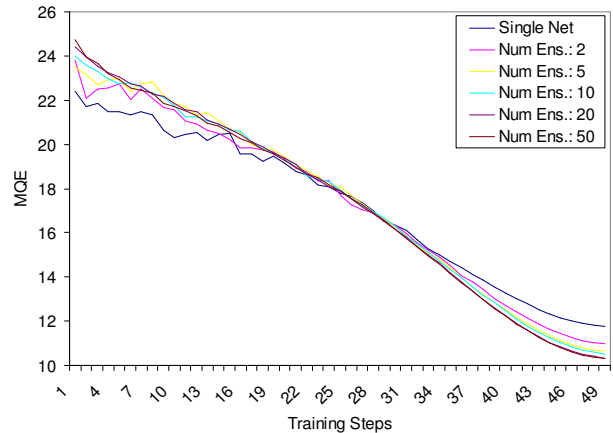


Fig. 3. MQE for various SOM ensemble sizes (1 to 50) over same time, for the "atom" dataset of the FCPS.

C. Learning rate decrement for 4-d hypercube problem

The choice of learning rate and decrement are of great importance. To examine learning rate decrements, an exponentially decreasing learning rate of $0.9e^{-i}$ is used, where i is the step number. A 15x15 network is trained for the single (sequential) and averaging (10-member ensemble) cases. Ideally the single network would take 10 times more to compute, however this just represents the ideal performance that the ensemble should attempt to match.

Fig. 4 shows how the learning rate didn't follow the $0.9e^{-i}$ curve, but an approximation comprised of 10 linear segments, the ends of which were at $(i, 0.9e^{-i})$ and $(i+1, 0.9e^{-(i+1)})$. This is not viewed as a major difference, and in fact, the "two phases of training" mentioned by Kohonen [17], where the learning rate follows a steep linear decrement for the "ordering" phase and a less steep linear decrement for the "convergence" phase, may be viewed as 2 linear segments that are chords of an exponentially decreasing curve.

The ensemble seems to converge to a much higher value than the single network for the choice of exponential learning rate decrement. Since the curve is quite steep, the same network with the same initial state, and input data, was trained for an equal amount of training, with a learning rate decrement following chords of $0.9e^{-i/2}$. The final value for the MQE is below 1.5 (Fig. 4), while previously was over 2.0. With a less steep exponential learning rate decrement, the performance of the ensemble was remarkably comparable to the single network – only the single network took 10 times more to compute.

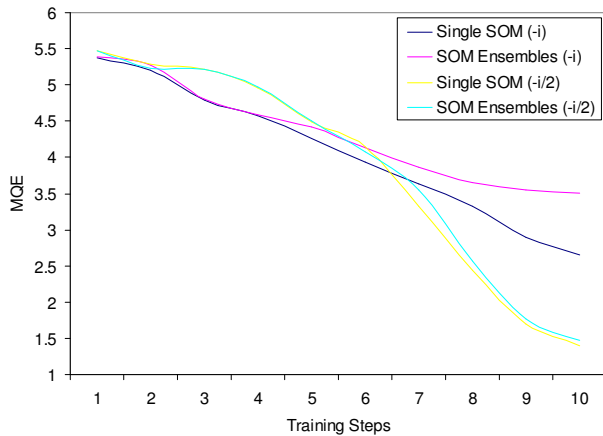


Fig. 4. Single network and averaging ensemble MQE for the 4-d hypercube, with a learning rate of $0.9e^{-i}$ and $0.9e^{-i/2}$, where i is the step number.

D. Frequency of averaging for 4-d hypercube problem

Another factor crucial in the averaging ensemble paradigm is the frequency of averaging. In all of the experiments carried out, this frequency was constant, since averaging was performed at the end of each (equal) step.

Fig. 5 shows the results of the MQE while training the 4-d hypercube on the same number of iterations, but performed in a variable number of steps (5 to 500 steps); where learning rate and neighbourhood radius are decreased linearly. From the results we observed that the precision of the ensemble

increases, as the frequency of averaging increases, but it never reaches the precision of the single net. However, the time taken for the ensemble to learn is, in theory, 10 times less (i.e. 10 ensembles) compared to the single net. But, in reality more averaging means more time wasted in packets (SOM weights) transferred over the network.

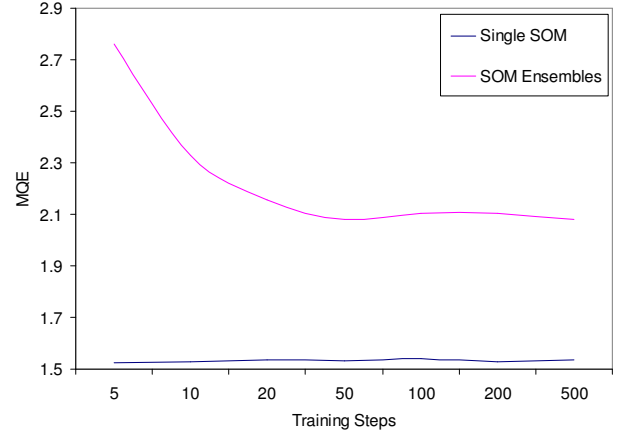


Fig. 5. Final MQE for single and 10-network ensemble trained from 5 to 500 training steps.

E. Wall-clock time for hypercube problem

The last experiment tests the proposed architecture on the real-world. A 10-d hypercube was trained sequentially on a single computer, and as an averaging ensemble on the departmental Condor pool at the University of Surrey. The size of the ensemble was 6 machines, while averaging was performed in 50 steps. The initial states of the 100x100 networks were identical. Both the single computer and ensemble were trained for a total of 10 million iterations, so it was expected that the Condor version would take less time to compute, but produce a higher final MQE.

Typical scheduling time for the Condor pool was 5min, although on few occasions, it was as low as 10sec, and on one occasion, it was 100min. For the first experiment, total training time was around 12h for the 6 SOM ensembles and 14h for the single SOM (Fig. 6).

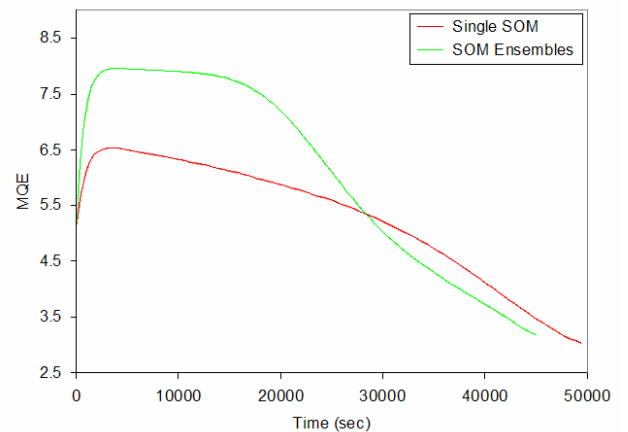


Fig. 6. MQE per wall-clock time for a 10-d linear alpha.

The results were as expected; with the figures showing

how MQE progressed for the single and averaging cases, with the horizontal axis indicating the duration in seconds of “wall-clock” time that elapsed since training started. The speed increase is less than 10%, although 6 machines are used instead of just one. However, those 6 machines were also engaged in other Grid computing tasks, during the execution of this experiment, while the single machine was dedicated. Furthermore, typical time to compute the MQE was 100sec, and averaging the weights took nearly as much.

Since for the 4-d hypercube problem it was observed that an exponential decrement of the learning rate is more efficient than a linear one, for a final “proof-of-concept” experiment, an 8-dimensional hypercube problem was also trained, on a 50x50 network, with a learning rate that followed 50 chords of an exponential curve. The size of the ensemble was increased to the maximum number of available Condor virtual machines (24), and the initial radius of the neighbourhood function was set at 25. The single network was trained for a total of 2×10^7 iterations, while the ensemble was trained with a total of 108 iterations. Fig. 7 shows how the MQE progressed per wall-clock time.

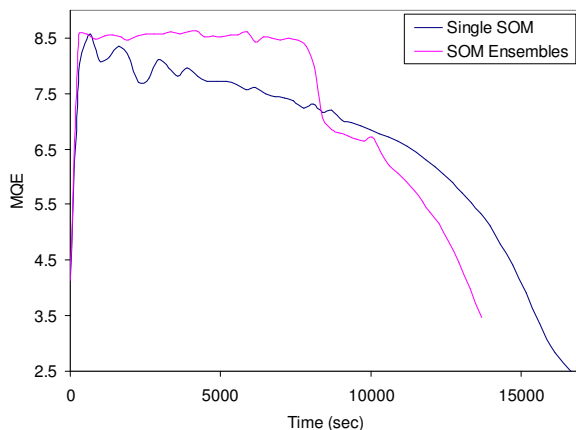


Fig. 7. MQE per wall-clock time for an 8-d hypercube, exponential alpha.

V. CONCLUSION

This paper examined the effectiveness of distributing ensembles of SOMs across Grids. Factors that affect the quality of a distributed SOM, following the averaging ensemble paradigm, include the frequency of averaging, the number of members of the ensemble, and the learning rate decrement that is applied. The main findings from these experiments is that faster training time using the SOM ensembles architecture proposed comes at a cost of higher MQE, which may affect precision. This does not necessarily mean that the SOM classifier does not perform well, but it places demands on defining adequate or acceptable performance, in exchange for faster training. Further experimentation to discover better selections of distribution parameters may help us to diminish the impact of averaging so that the performance of the resulting network is only slightly diminished in comparison to the equivalent single SOM, with a substantially reduced training time.

REFERENCES

- [1] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.
- [2] L. Breimen, “Bagging Predictors”, *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
- [3] Y. Freund, and R. Schapire, “A Short Introduction to Boosting”, *Journal of Japanese Society for AI*, vol. 14, no. 5, pp. 771-780, 1999.
- [4] T. Vin, M. Seng, N. Kuan, and F. Haron, “A Framework for Grid-based Neural Networks”, *Proc. of the 1st Int. Conf. on Distributed Frameworks for Multimedia Applications*, pp. 246-253, 2005.
- [5] D. Calvert, and J. Guan, “Distributed Artificial Neural Network Architectures”, *Proc. of the 19th IEEE Int. Symposium on High Performance Computing Systems and Applications*, pp. 2-10, 2005.
- [6] H. Yoon, J. Nang, and S. Maeng, “A Distributed Backpropagation Algorithm of Neural Networks on Distributed-Memory Multiprocessors”, *Int Conf on Parallel Processing*, pp 358-363, 1991
- [7] M. Kumar, and L. Patnaik, “Mapping of Artificial Neural Networks onto Message Passing Systems”, *IEEE Trans. on Systems, Man, and Cybernetics*, Part B, vol. 26, no. 6, pp. 822-835, 1996.
- [8] N. Bandeira, V. Lobo, and F. Moura-Pires, “Training a Self-Organizing Map distributed on a PVM network”, *Proc. of IEEE Joint Conf. on Neural Networks*, vol. 1, pp. 457-461, 1998.
- [9] J. Lu, D. Goldman, M. Yang, and N. Bourbakis, “High-performance Neural Network Training on a Computational Cluster”, *Proc. of the 7th IEEE Int. Conf. on High Performance Computing and Grid in Asia Pacific Region*, pp. 46-472, 2004.
- [10] A. Gutierrez, R. de Llano, F. Cavero, and J. Gregorio, “Parallelization of a Neural Net Training Program in a Grid Environment”, *Proc. of the 12th Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, pp. 258- 265, 2004.
- [11] P.V. Coveney, “Scientific Grid Computing”. *Phil. Trans. of the Royal Society*, vol. 363, no. 1833, pp. 1701-1713, 2005.
- [12] I. Foster, and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure (2nd Ed)*, Morgan Kaufmann, San Francisco, 2004.
- [13] I. Foster, and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, *Int. Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.
- [14] I. Foster, C. Kesselman, and S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, *Int. Journal of Supercomputer Applications*, vol. 15, no. 3, pp. 200-222, 2001.
- [15] D. Thain, T. Tannenbaum, and M. Livny, “Distributed Computing in Practice: The Condor Experience”, *Concurrency and Computation*, vol. 17, no. 2-4, pp. 323-356, 2005.
- [16] K. Ahmad, B. Vrusias, and M. Zhu, “Visualising an Image Collection”, *Proc. of the 9th Int. Conf. On Information Visualization*, pp. 268-274, 2005.
- [17] T. Kohonen, *Self-Organization and Associative Memory (3rd Ed)*, Springer, Berlin, 1989.
- [18] L. Gillam, K. Ahmad, and G. Dear, “Grid-enabling Social Scientists: some infrastructure issues”, *Proc. of 1st e-Social Science Conf., Manchester, UK*, 2005.
- [19] A. Ultsch, “Clustering with SOM: U*C”, *Proc. Workshop on Self-Organizing Maps*, pp. 75-82, 2005.
- [20] B. Vrusias, *Combining unsupervised classifiers: a multimodal case study*, PhD thesis, University of Surrey, 2004.
- [21] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen, “SOM_PAK: The Self-Organizing Map Program Package.” *Technical Report A31*, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland, 1996.