A Parallel Incremental Learning Algorithm for Neural Networks with Fault Tolerance

Jacques Bahi[1], Sylvain Contassot-Vivier[2], Marc Sauget[1,3] and Aurélien Vasseur[3]

1. LIFC, University of Franche-Comté, Belfort, France
{jacques.bahi,marc.sauget}@iut-bm.univ-fcomte.fr
WWW home page: http://info.iut-bm.univ-fcomte.fr/and
2. LORIA, University Henri Poincaré, Nancy, France
{sylvain.contassotvivier}@loria.fr
Web page: http://www.loria.fr/∼contasss/homeE.html
3. Femto-St, University of Franche-Comté, Montbéliard, France
{aurelien.vasseur}@pu-pm.univ-fcomte.fr

Abstract. This paper presents a parallel and fault tolerant version of an incremental learning algorithm for feed-forward neural networks used as function approximators. It has been shown in previous works that our incremental algorithm builds networks of reduced size while providing high quality approximations for real data sets. However, for very large sets, the use of our learning process on a single machine may be quite long and even sometimes impossible, due to memory limitations. The parallel algorithm presented in this paper is usable in any parallel system, and in particular, with large dynamical systems such as clusters and grids in which faults may occur. Finally, the quality and performances (without and with faults) of that algorithm are experimentally evaluated.

Key words: Neural Networks, Learning algorithms, Parallelism.

Introduction

The work presented in this paper takes place in a multi-disciplinary project called *Neurad*, involving physicists[1] and computer scientists[2], whose goal is to enhance the treatment planning of cancerous tumors by external radiotherapy. In our previous works [1, 2], we proposed an original approach to solve scientific problems whose accurate modeling and/or analytical description is not directly possible. That method is based on the collaboration of computation codes and neural networks used as universal approximators. Thanks to that method, the *Neurad* software provides a fast and accurate evaluation of radiation doses in any given environment (possibly heterogeneous) for given irradiation parameters.

[1] IRMA/Crest team of the FEMTO-ST institute
[2] AND team of the LIFC and Algorille team of the LORIA

In that context, a new learning algorithm has been designed which provides a network of limited size while giving very accurate results.

However, the sequential version of our algorithm is restrained by the use of a single machine at the same time, which does not allow the learning of very large data sets due to memory limitations and the induced important computation times. This is why the design of a parallel version has been planned. Our approach uses domain decomposition to exploit parallelism, so that the initial neural network is decomposed in several sub-networks. In order to ensure a good quality of the global network while preserving good performances, a fine tuning of the overlapping of the sub-domains is performed. Moreover, in order to be usable in any kind of parallel system (parallel machine, cluster or grid), a fault tolerance mechanism is included in our parallel algorithm.

In the following section, a brief state of the art on neural networks is presented. Then, our sequential incremental learning algorithm is detailed in Section 2. In Section 3, our parallel version is fully described as well as our fault tolerance mechanism. Finally, the presented algorithm is qualitatively and quantitatively evaluated in Section 4.

1 State of the art

Since the first developments of neural networks [3], the major encountered problems lie in their building and learning. Indeed, there are some results proving that a multi-layer neural network can be used as a universal approximator [4, 5]. However, there is no result about how to build an optimal structure. Many algorithms give good results, as the classical back propagation algorithm [6, 7]. Moreover, there exist many optimizations for that kind of algorithms. They concern the structure, as the Square MLP [8] or the HPU [9] designs, and the learning process, as the QuickProp [10] or the Rprop algorithm [11]. Nonetheless, they work on static structures which have to be inferred manually according to the user's experience.

In order to solve that recurrent problem, new learning processes have been proposed which aim at dynamically building the structure of the neural network during the learning process. There are two main kinds of such algorithms.

The first kind corresponds to the incremental learning algorithms. Their principle is to begin the learning process with a neural network of minimal size and to progressively increase the number of neurons until satisfying the desired criterion. The addition of a new node is conditioned by the stabilization of the learning process while the requested accuracy is not reached yet. There exist many variants of that incremental process [12–15].

The second kind of dynamic learning algorithms consists of the symmetric approach, i.e, a decimation process. In this case, the learning process begins with an over-sized complete structure containing a maximal or sufficiently large number of fully connected neurons. Then, during the learning process, the links and neurons which reveal to be useless are deleted. The most known algorithms

in this class are probably the "Optimal Brain Damage" algorithm [16] and the "Optimal Brain Surgeon" algorithm [17].

Nevertheless, all those algorithms are sequential, which limit their use to a single mono-processor machine. So, even if they give very good results, they cannot be used in practice to process very large data sets. This is why there has also been an important effort led towards the parallelization of existing algorithms or the design of specific parallel learning algorithms. J.Torresen and S.Tomita present a detailed report on parallel approaches in the context of classification neural networks [18]. The major approach in this field is to decompose the initial data set and to build several sub-networks. In some cases, an additional global network is used to retrieve the sub-network corresponding to a given input, such as in [19]. However, to the best of our knowledge, all those studies are focused on classification networks and not on approximator ones. Moreover, they do not take into account the robustness of the algorithm when it is used in dynamical parallel systems, in which network or processor faults may occur. This is an important feature of the parallel learning algorithm presented in this paper.

2 Sequential building/learning algorithm

2.1 Network structure

As mentioned above, it has already been shown that a multi-layer neural network can be used as a universal approximator. We use here the common architecture which consists in three layers of neurons (input, hidden and output). The number of neurons in the input layer is determined by the number of parameters of the function to approximate. In the same way, the number of neurons in the output layer is directly induced by the number of outputs of the target function. In the context of the *Neurad* project, which aims at evaluating radiation doses, the number of neurons in the output layer is reduced to a single neuron which delivers the dose. Finally, the last important parameter in the network structure setup is the number of neurons in the hidden layer. As that number does not directly depend on the number of inputs and outputs of the problem, there is no precise rule to compute it. The only external information which may help to fix that number is the variation degree of the input data. However, there is no accurate relation between those two values. It is thus necessary to dynamically set that number of hidden neurons during the learning process to obtain the most suited networks. That incremental building is described in Section 2.2.

In addition to the three-layer organization, we have used a HPU (Higher-order Processing Unit) structure [20] in order to enhance the capacity of the network to approximate high degree functions with sharp variations while preserving a limited number of neurons. That structure also permits to obtain faster trainings. It consists in artificially increasing the number of inputs of the network with polynomial combinations of the original inputs up to a maximal degree (referred to as the order of the network). For example, the inputs of an HPU network of order 3 corresponding to an original network with two inputs (x_1, x_2) are $(x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3)$.

In our applicative context, we have also noticed that another structural modification that can enhance the results of the neural network is to replace the linear output neurons by sigmoid ones.

2.2 Incremental building/learning

The classical back-propagation learning method is known to be rather slow. In order to speed up the training process, we have chosen the Resilient back Propagation (Rprop) algorithm [11], which is one of the most efficient optimizations of that process (See [21] for a complete survey). Its main difference with the classical back-propagation is that it only uses the sign of the error derivative to update the weights in the network. Moreover, the updatings are performed, for each weight, with a distinct value independent from the error. Those values are respectively increased or decreased, similarly to an acceleration or a deceleration, according to the direction of the error evolution.

Concerning the incremental building of the hidden layer of the network, the principle of our algorithm, depicted in Figure 1, is to perform a Rprop learning over the current HPU neural network until the error either reaches the required accuracy or does not sensibly evolve anymore according to a given threshold.

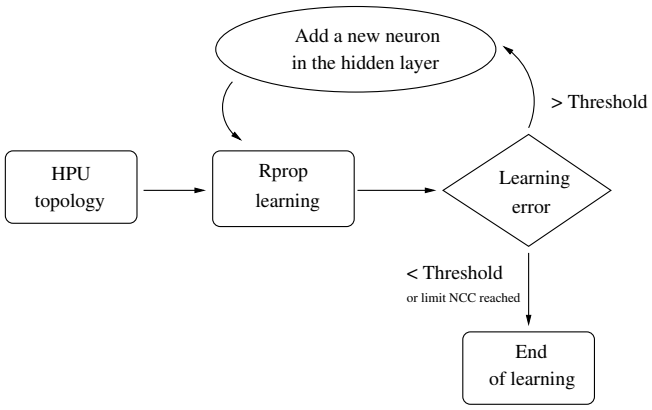

Fig. 1. Incremental building rule.

More precisely, our algorithm starts with a given number of hidden neurons (one or a few). Then, when the neural network reaches the desired accuracy, the learning process stops. Otherwise, when the learning limit of the current neural network is considered to be reached (stabilization or over-learning), a neuron initialized with null weights and threshold is added to the hidden layer without modifying the other neurons and links. The null initialization of the additional neuron and the non modification of the other elements in the network are important to avoid any deviation of the current network from its optimization path. After that, the learning process is resumed with that new network configuration.

That incremental process is repeated until the desired accuracy is reached or the difference between the results of two consecutive configurations of network becomes too small. That last case corresponds to situations where the overall limit of the network has been reached and the addition of hidden neurons does not improve the results anymore.

As in other learning algorithms, the specification of a validation data set is possible in order to control the learning process and avoid over-learning. Moreover, an upper bound to the number of hidden neurons can be specified in order to limit the final size of the network.

Finally, we obtain a sequential and incremental learning process that allows us to build and train efficient and accurate neural networks of limited size for function approximation. Moreover, our approach, used in the particular context of the *Neurad* project, is general and can be used for any kind of data, real or synthetic, and with any kind of activation function of the neurons.

3 Parallel algorithm

Our parallel algorithm is based on the classical client-server model applied to the distribution of the work. The role of the server is to distribute the different tasks constituting the overall process to the other nodes, which are the clients.

In order to obtain such a simple operating scheme, it is necessary to divide the learning in separate tasks. This is possible according to the principle that the approximation of a function on a given domain can be obtained by performing multiple approximations of that function on sub-domains forming a partition of the initial domain. The following paragraph describes the domain decomposition technique we have used in our algorithm.

3.1 Domain decomposition

In the case of neural networks, the domain decomposition leads to a composition of several sub-networks in order to perform the overall approximation of the target function. Indeed, the initial domain of the data set is divided into subspaces along one or several of its input dimensions. Obviously, the output dimensions cannot be divided as it would not be possible to know in advance what output sub-domain corresponds to a given input vector.

So, the overall approximation of the initial data set is obtained by the separate learnings of the data subsets induced by the domain decomposition. The decomposition along each dimension can be performed in any way. The simplest one is certainly to perform a decomposition which produces data subsets of approximately the same sizes. The decomposition principle is illustrated in Figure 2.

In addition to the possibility to design a simple and efficient parallel algorithm, the domain decomposition presents another important advantage in the case of neural network approximation. It significantly reduces the complexity of the target functions to approximate. Indeed, it is far more easier to approximate

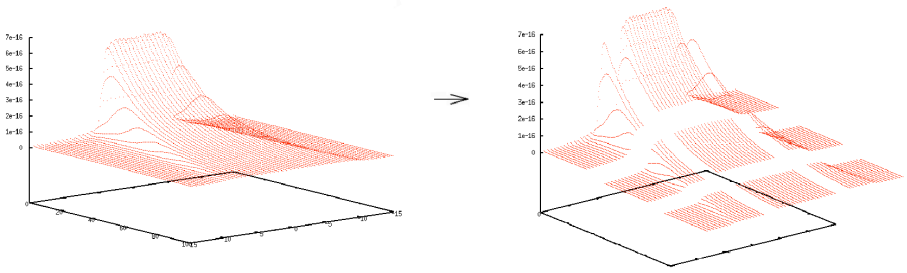

Fig. 2. Domain decomposition in 9 sub-domains of a two-dimensional data set.

a function on a small interval than on a large one, especially if there are sharp variations. Thus, a better accuracy can even be expected on each sub-domain.

However, performing the learnings on sub-domains constituting a partition of the initial domain is not satisfying according to the quality of the results. This comes from the fact that the accuracy of the approximation performed by a neural network is not constant over the learned domain. Thus, it is necessary to use an overlapping of the sub-domains as explained below.

3.2 Overlapping between sub-domains

As mentioned above, the disadvantage brought by the use of several sub-networks of neurons in place of a single one comes from accuracy problems at the frontiers between each sub-network. Although the neural networks have the capacity of generalization on any given training domain, they do not provide representative results outside this domain and the approximation error increases toward the limits of the domain. This mainly comes from the fact that on the borders of the domain there is less available information about the target function than in the middle of the domain. So, the error is smaller in the middle of the domain than on its borders.

If this is not relevant when using a single neural network, it becomes an issue when several sub-networks are used to represent the domain. Indeed, an increase of the number of sub-networks directly increases the number of frontiers between the sub-domains and then, the number of higher error areas in the domain. Consequently, the average accuracy of the approximation may be importantly reduced. Moreover, the error distribution becomes decomposition-dependent, which is a very restrictive feature as it implies that no decomposition should be made in the areas of higher interest.

Fortunately, there is a solution to that problem which consists in masking the borders of the domains by performing an overlapping of the sub-domains during the learning phase. Thus, we obtain a set of sub-networks whose approximation errors at the frontiers between them is of the same order as anywhere else in the domain. This mechanism implies the distinction, for each sub-network, between

its learning domain and its exploitation domain. The former is the domain used to perform the learning of the sub-network; it overlaps with the learning domains of the neighboring sub-networks. The latter is the domain of validity of the sub-network during the exploitation phase; as it is used to find the most suited sub-network to process a given input vector, it does not overlap with the exploitation domain of any other sub-network. The overall principle is depicted in Figure 3.

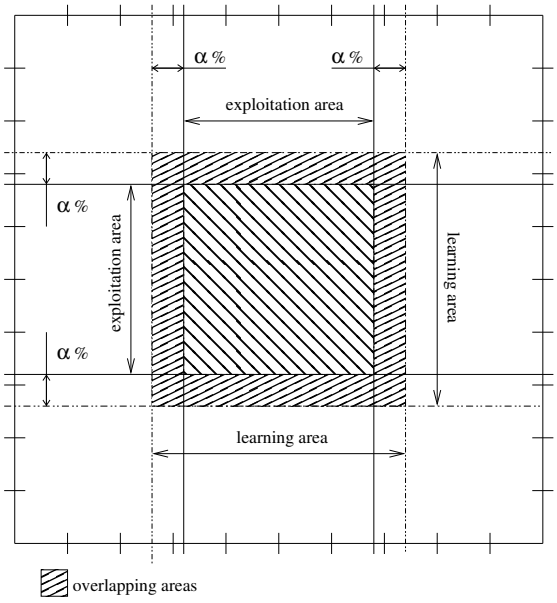

Fig. 3. Overlapping for a sub-network in a two-dimensional domain with ratio α.

In this way, each sub-network has an exploitation domain smaller than its training domain and the accuracy problems at the borders are no longer relevant. Nonetheless, in order to preserve the performances of the parallel algorithm, it is important to carefully set the overlapping ratio α. It must be large enough to avoid the borders errors, and as small as possible to limit the size increase of the data subsets. The trade-off value depends on the nature of the initial data set and thus, on the application field. In Section 4, a case study of that ratio is performed in the context of the *Neurad* project, for radiation dose distributions.

3.3 Fault tolerance mechanism

In addition to our parallel scheme, a fault tolerance mechanism is included that enables its use in any dynamical parallel system such as open multiuser clusters or grids. Typically in such contexts, the communication links may be temporarily or definitely interrupted, the processors speed may sharply vary due to the multiuser context and the processors may even stop working. However, it is reasonable to assume that the conjunction of all those possible faults is quite a rare

event. Moreover, extreme cases, in which all the processors or links are faulty, are obviously out of consideration. Consequently, we assume in the following that during the learning process, there is always at least one operational node (the server is also the client). With those hypothesis, the goal of our fault tolerance mechanism is to ensure that the learning process continuously progresses as efficiently as possible whatever the events occurring on the system are.

The principle of our system is based on regular message exchanges between the server and the nodes in order to monitor their current state.

The server initializes, for each client, a structure containing:

- its current state: either waiting, learning or in fault
- the identifier of the sub-network it is in charge of
- the date of the last received message from that client

Then, it enters the main learning loop in which it distributes the sub-networks to train to the available clients and monitor their states in order to react to potential faults. Hence, when a client is idle, the server sends it one of the remaining sub-networks to process and its associated data subset.

Algorithm 1 Parallel algorithm (Client)

```
Inputs:
  Network net      // Network to be trained
  TrSet set        // Training data set
Variables:
  Msg msg          // Message received from the server
Results:
  Network net      // Network trained

  // Main loop of the learning process
  while inProgress = True  do
    if  receivedMsg(server,msg) then
      if msg = TRAINING_REQUEST then
        // Starting of a sub-network training
        receiveSet(server,set)
        receiveSubNet(server,net)
        initiateLearning(net, set)
      else if msg = INTERRUPTION_REQUEST then
        // Stopping of the current training
        notifyLearningInterruption()
      else if msg = TERMINATION_REQUEST then
        // Stopping of the overall process
        inProgress ← False
      end if
    end if
    updateLearningState()
    if learningState=FINISHED then
      sendFinishedMsg(server)
      sendSubNet(server,net)
    else if learningState=BACKUP then
      sendBackupMsg(server)
      sendSubNet(server,net)
    else if learningState=IN_PROGRESS then
      sendInProgressMsg(server)
    else
      wait(someTime)
      sendWaitingMsg(server)
    end if
  end while
```

It is important to note that the sub-network sent may be already partially trained. This is the case when the sub-network has been recovered from a previous learning interruption due to a fault in the system. When the client is processing a sub-network, the server regularly verifies that it is still alive. Finally, in order to avoid the restarting from scratch of a previously faulted learning, the client regularly sends to the server the current version of its sub-network. So, when a fault occurs on the client, the sub-network can be redistributed to another idle client. However, it may occur that a fault of a client comes from a temporary interruption of its link with the server. In such cases, there is no way to identify the cause of the fault and the server will also redistribute the work of that client to another one. Then, when the faulty link comes back working, the server will detect a dual processing of the corresponding sub-network on two clients.

Algorithm 2 Parallel algorithm (Server)

```
Inputs:
  Network [ ] nets   // Meta-network to be trained, list of the sub-networks
  int nbSN           // Number of sub-networks
  int nbP            // Number of nodes
Variables:
  NodeState states[nbP]  // List of the nodes states
  TrSet [ ] sets         // Table describing the different training data subsets
  int [ ] remTrSets      // List of the remaining sub-networks to train
  int sizeRTS            // Size of remTrSets
  int id                 // Index of the following sub-network to train
  Msg msg          // Message received from a client
Results:
  Network [ ] nets // Trained meta-network

  // Initialization of the states structure
  init(states)
  // General loop of the training process
  while inProgress = True do
    for i= 0 to nbP−1 do
      if receivedMsg(i,msg) then
        if msg = WAITING then
          sizeRTS ← updateRTS(remTrSets)
          if sizeRTS > 0 then
            id ← nextSet(remTrSets)
            sendTrainingRequest(i)
            sendSet(i,sets[id])
            sendSubNet(i,nets[id])
            notifyTrainingInProgress(states, i, id)
          end if
        else if msg = FINISHED then
          nets[i] ← receiveSubNet(i)
          sendWaitingRequest(i)
        else if msg = IN_PROGRESS then
          notifyInProgress(i, states)
        else if msg = BACKUP then
          nets[i] ← receiveSubNet(i)
        end if
      end if
    end for
    lifeControl(states, remTrSets)
    inProgress ← verifyRemTrain(states, remTrSets)
  end while
  save(nets)
```

The chosen policy in this case is to let both the clients processing the same sub-network and, as soon as one of them returns the result, the server sends an interruption message to the other client in order to make it accept another work. When there is no idle client although there remains sub-networks to process, the server waits until one of the clients sends back its resulting sub-network after the learning completion. Finally, the main process stops when there is no more sub-network to train and all the clients have returned their results. Once this is made, the server sends a message to all the clients, indicating the end of the overall process, and saves the complete structure of the global network.

On the client side are the complementary operations to the server. When a client receives a training message, it takes delivery of the sub-network and the associated data subset. Then, the node performs the learning of the sub-network using the learning algorithm previously described. During this stage, the node regularly notifies its state to the server by sending the evaluation of its training error. It also sends, less frequently, a safeguard copy of its current sub-network.

Algorithm 1 and Algorithm 2 respectively depict the general algorithmic schemes used on the client side and on the server side.

4 Experimental results

In this section, the quality, performance and robustness of our algorithm are experimentally evaluated. Our algorithm has been implemented in standard C++ with the LAM/MPI communication library [22]. Although that library is not the most suited to fault tolerance, it offers the minimal features of robustness allowing us to validate our algorithm. It must be pointed out that our algorithm does not depend on any implementation environment and another communication library may be used.

The presented experiments have been performed on a multiuser cluster of 20 nodes (Intel PIV, 3Ghz, 1 Go RAM, Debian Linux).

4.1 Quality of the parallel learnings

The first test shows the impact of the overlapping between sub-domains on the learning error. We have used a training set containing only one distribution of radiation dose deposit in a homogeneous material to perform this test. The results of that first experiment are presented in Table 1.

Overlapping (%)	0	5	7	10	50
Mean Error (%)	2.86	2.38	2.37	2.37	2.45
Mean Bias (%)	2.28	1.86	1.82	1.83	1.85

Table 1. Impact of the overlapping between sub-domains onto the learning error

This experiment shows that the overlapping is necessary to obtain the best accuracy with our parallel learning algorithm. Effectively, it can be seen that the overlapping ratio directly influences the accuracy of the final network. However,

it also shows that this overlapping ratio cannot be fixed a priori without a preliminary study. Indeed, If it is too small, the errors at the frontiers reduce the quality of the final results, and if it is too large, the quality will also be degraded due to a larger size of the sub-domains, implying a potentially higher complexity of the sub-functions to approximate. Those two aspects clearly show that the optimal value of the overlapping ratio is strictly positive but is also far smaller than the maximal possible range. This comes from the fact that the accuracy of the approximation at any given position in the domain, and in particular at the frontiers between sub-domains, directly depends on the number of neighbors used to perform the approximation and, in some sense, on the relative flexibility of the elementary functions used in the neural network. Although there is no analytical way to compute the optimal value of that overlapping ratio for the moment - this will be the subject of future works - it seems quite obvious that this ratio is directly linked to the complexity (maximal and/or mean frequency, value range,...) of the function to approximate.

4.2 Performances of the parallel learning scheme

That second experiment aims at evaluating the impact of the domain decomposition on the performances and quality of our parallel learning algorithm. It has been achieved with a training set generated by the BeamNrc code. That code is a simulator based on the Monte Carlo technique for nuclear applications. The data set is the results of three irradiations of a homogeneous environment of water at three different distances (98, 100, and 102 cm). The set is composed of 1,500,000 points. For each point, we store: the spatial position, the material density and the length between the water environment and the particle accelerator.

The results of our parallel learning algorithm on that data set are presented in Table 2 for several configurations of domain decomposition performed on the three spatial input parameters. The convergence ratio indicates the percentage of sub-networks which have actually reached the requested accuracy. In fact, for each sub-network, the learning process may either stabilize before reaching the desired accuracy or not reach it in reasonable time.

Decomposition	$1 \times 1 \times 1$	$2 \times 1 \times 2$	$2 \times 2 \times 2$	$3 \times 1 \times 3$	$3 \times 2 \times 3$	$3 \times 3 \times 3$
Mean Error	6.20e-4	1.57e-4	1.0e-4	1.63e-4	1.0e-4	1.01e-4
Min Error	6.20e-4	9.99e-5	9.99e-5	9.99e-5	9.99e-5	9.99e-5
Max Error	6.20e-4	2.3e-4	1.01e-4	4.97 e-4	1.01e-4	1.23e-4
Convergence ratio (%)	0	25	33	62	66	92
Min Time	4H34	4H06	0H54	1H11	0H04	0H03
Max Time	4H34	8H10	3H25	5H59	3H47	1H42

Table 2. Results of our parallel algorithm for several domain decompositions performed on the three spatial dimensions of the training set.

The results show that our parallel learning algorithm increases the global accuracy of the neural network while decreasing its learning time. That double gain is due, as mentioned in Section 3.1, to the fact the learnings are performed on smaller domains than the initial one.

Concerning the quality of the network, it can be seen that the decomposition of all the input dimensions facilitate the overall convergence of the learning process. Moreover, the convergence rate is also greatly improved according to the number of decomposed input dimensions and the total number of sub-networks.

Concerning the performances, it must be noticed that as the initial training set is quite large, the first result with no domain decomposition has a null level of convergence and its learning time does not actually correspond to the time required to obtain the desired accuracy. So, that time should be far larger.

The larger maximal time of the $3 \times 1 \times 3$ decomposition according to the $2 \times 2 \times 2$ one is due to the difference in complexity of the function to approximate in the respective sub-domains obtained. Moreover, as for the case without any decomposition, the maximal times variations also come from the fact that the learning process may be stopped before the convergence, in order to obtain reasonable times. So, $3 \times 1 \times 3$ has a larger maximal time but also has a far better convergence rate. If the stopping criteria had been based only on the convergence, it is strongly probable that the maximal time of $2 \times 2 \times 2$ would have been larger. It is also the case for $3 \times 2 \times 3$ compared to $2 \times 2 \times 2$.

It has to be noticed that our current version of the algorithm realizes an implicit form of load-balancing by the use of a tasks queue managed in the client-server scheme. However, that load-balancing could be improved - this is also a future work - by performing a non-regular decomposition of the domain in order to obtain approximately the same complexities of the sub-domains. This should induce similar learning times of the sub-domains and thus sensibly enhances the overall learning time of the entire domain.

4.3 Performances of the robustness mechanism

As our fault tolerance mechanism allows our algorithm to always successfully terminate as long as there remains at least one operational client node in the system, we focus here on the performances of our algorithm in presence of faults.

However, as such an evaluation should require an entire study in itself, we do not try to be exhaustive here but just give some hints on the behavior of our algorithm. Hence, we use only one frequency of the intermediate backups of the sub-networks here (every 5 learning iterations) but its optimal implementation and value should be discussed in a further study.

We give on the left side of Table 3 the total times of a learning decomposed in 9 sub-domains started with 9 clients in function of the number of permanent client faults occurring during the process. Those results are means of several executions with a general (uniform) random distribution of the faults during the process. In order to make a comparison, the learning times obtained without any fault are given for different numbers of clients on the right side of Table 3.

It can be seen that the progression of the learning times according to the number of faults merely follows the times obtained without faults when the number of clients decreases. However, it is interesting to see that this progression tends to slow down when the number of faults increases.

In fact, those results can be explained by the fact that, additionally to the number of faults, the instants at which the faults occur also have an impact

# faults	1	2	3	4	5	6	7	8
Times (min)	13	14	18	19	21	23	26	42

# clients	9	8	7	6	5	4	3	2	1
Times (min)	10	10	11	12	16	16	29	32	58

Table 3. Learning times with 9 initial clients in function of the number of permanent client faults on the left. Learning times without faults in function of the number of clients on the right.

on the performances. The latter they occur during the process, the better are the performances. This comes from two factors. The most obvious and general one is that the latter the faults occur, the longer the algorithm works with a larger number of clients. The latter is more specific to the current version of our parallel learning algorithm as there may be sensible differences between the learning times of the sub-domains. Hence, for the latest faults, it is highly probable that some of the sub-domains learnings are already completed and that some clients are idle, offering the possibility to perform an immediate re-assignation of a stopped learning when a fault occurs. In such cases, the impact of the fault on the performances is minimal.

So, for uniform faults distributions and large numbers of faults during the process, it is highly probable that some of them occur in the particular context described above and thus have a very small impact on the overall performances.

Conclusion

A parallel learning algorithm has been presented which includes fault tolerance. Its principle is based on a domain decomposition of the input parameters of the training data set and on an overlapping of the sub-networks to ensure a good accuracy of the network on the entire domain of the data set. Moreover, the fault tolerance enables the use of that algorithm on a large class of parallel systems, including dynamical ones.

Qualitative and quantitative evaluations of the algorithm have been performed experimentally on real data sets. They confirm the good behavior of our algorithm in terms of performances, quality and robustness.

In the following of the *Neurad* project, it should be interesting to add another important feature to our learning process which is the possibility to make a network learn new data without loosing its previously accumulated knowledge.

Acknowledgments

The authors thank the LCC (Ligue Contre le Cancer) for the financial support, the OPRAD project (Région Franche-Comté) and the CAPM (Communauté d'Agglomération du Pays de Montbéliard).

References

1. Sauget, M., Martin, E., Gschwind, R., Makovicka, L., Contassot-Vivier, S., Bahi, J.: Développement d'un code de calcul dosimétrique basé sur les réseaux artificiels de neurones. In: 44ièmes Journées Scientifiques de la Société Française de Physique Médicale, Avignon, France (2005)

2. Bahi, J., Contassot-Vivier, S., Makovicka, L., Martin, E., Sauget, M.: Neural network based algorithm for radiation dose evaluation in heterogeneous environments. In: Artificial Neural Networks - ICANN 2006. Volume 4132/2006 of Lecture Notes in Computer Science., Athens, Greece, Springer Berlin / Heidelberg (2006) 777–787
3. Pitts, W., McCulloch, W.S.: A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics **5** (1943) 115–133
4. Cybenko, G.: Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems **2** (1989) 303–314
5. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Network **2** (1989) 359–366
6. le Cun, Y.: Modèles connexionnistes de l'apprentissage. PhD thesis, Université Pierre et Marie Curie (1987)
7. Rumelhart, D., McClelland, J., research group, P.: Parallel Distributed Processing. Volume 1-2. The MIT Press (1986-87)
8. Flake, G.: Square unit augmented, radially extended, multilayer perceptrons (1998)
9. Giles, C., Taxwell, T.: Learning, invariance and generalization in high-order neural networks. Optical Neural Networks (1994) 344–350
10. Fahlman, S.E.: Faster-learning variations on back-propagation: An empirical study. In Morgan-Kaufmann, ed.: Connectionist Models Summer School. (1988)
11. Riedmiller, M., Braun, H.: A direct adaptative method for faster backpropagation learning: The rprop algorithm. In: Proceedings of the IEEE Internationnal Conference on Neural Networks (ICNN93), San Francisco (1993)
12. Polikar, R., Udpa, L., Udpa, S., Honavar, V.: Learn++: An incremental learning algorithm for supervised neural networks. IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews **31** (2001) 497–508
13. Dunkin, N., Shawe-Taylor, J., Koiran, P.: A new incremental learning technique. In Verlag, S., ed.: Neural Nets Wirn Vietri-96. Proceedings of the 8th Italian Workshop on Neural Nets. (1997) 112–118
14. Kurkova, V., Beliczynski, B.: An incremental learning algorithm for gaussian radial-basis-function approximation. In: Second International Symposium on Methods and Models in Automation and Robotics. (1995) 675–680
15. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In Touretzky, D.S., ed.: Advances in Neural Information Processing Systems. Volume 2., Denver 1989, Morgan Kaufmann, San Mateo (1990) 524–532
16. LeCun, Y., Denker, J., Solla, S., Howard, R.E., Jackel, L.D.: Optimal brain damage. In Touretzky, D.S., ed.: Advances in Neural Information Processing Systems II, San Mateo, CA, Morgan Kauffman (1990)
17. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S.J., Cowan, J.D., Giles, C.L., eds.: Advances in Neural Information Processing Systems. Volume 5., Morgan Kaufmann, San Mateo, CA (1993) 164–171
18. Torresen, J., Tomita, S.: A review of parallel implementations of backpropagation neural networks (1998)
19. Sheng-Uei Guan, Qi Yinan, S.K.T., Li, S.: Output partitioning of neural networks. In Science, E., ed.: Neurocomputing. Volume 68. (2005) 38–53
20. Ghosh, J., Shin, Y.: Efficient higher-order neural networks for classification and function approximation. Int. J. Neural Systems **3** (1992) 323–350
21. Tertois, S.: Réduction des effets des non-linéarités dans une modulation miltiporteuse à l'aide de réseaux de neurones. PhD thesis, Rennes 1 (2003)
22. Burns, G., Daoud, R., Vaigl, J.: LAM: An Open Cluster Environment for MPI. In: Proceedings of Supercomputing Symposium. (1994) 379–386