IETE Journal of Research
Vol 44, Nos 4&5, July-October 1998, pp 219-225.

On Making Neural Network Based Learning Systems Robust

ASHISH GHOSH
Machine Intelligence Unit, Indian Statistical Institute, 203 BT Road, Calcutta 700 035, India.
ash@isical.ac.in

AND

HIDEO TANAKA
Department of Industrial Engineering, College of Engineering, Osaka Prefecture University,
1-1 Gakuen-cho,Sakai, Osaka 593, Japan.
tanaka@ie.osakafu-u.ac.jp

A method for making nerual network based learning systems robust with respect to component failure (damaging of nodes/links) is suggested in the present investigation. The method allows some of the components to fail at various instants of the entire learning process. The change in error value caused by this damage will be adjusted while the other components learn their parameters during the rest part of learning. The damaging/component failure process has been modeled as a Poisson process. The instants or moments of damaging are chosen by statistical sampling. The components to be damaged are determined randomly. As an illustration, the model is implemented on the back-propagation learning algorithm.

Indexing terms : Neural networks, Robustness, Learning algorithms

A neural network (NN) [1-5] based system consists of a large number of neurons with massive connectivity among them. Local connectivity among the neurons/nodes (computing elements) being very high and the storage of information being distributed, the approach is claimed to be highly robust and can be applied even when information is ill-defined and/or defective/partial or noisy. If some of the components fail to work completely or partially, the other components adjust themselves (during iterative learning) in such a manner that the output is not deteriorated much. This feature is exploited here to provide a model for designing robust NN based learning systems. As a NN based system consists of a large number of components, the possibility of some of its components (nodes and/or links) to fail to work is very high. Here lies the necessity of designing robust (under complete or partial failure of components) learning systems. In this context we mention that several works [6-9] have been done to design optimum NN architectures by damaging some of the components.

In the present work an attempt is made to provide a model for designing robust neural network based learning systems. This is done by damaging (completely) some of the components of a NN based system during the process of learning its parameters (weights and biases) and studying the change in its performance. Since (some of) the components are damaged at different time instants of the entire learning process, the error values at the output nodes will be different than if those components would not fail; and thus the other components will adjust their parameters so as to compensate for this damage. Thus the performance will not be deteriorated much due to this damage and the system will be robust. The component failure process of NN has already been shown to follow Poisson distribution under certain assumptions [10]. Under this model, the components (nodes and/or links) to be damaged are chosen randomly. The time instants (i.e., when a damage occurs) are determined by drawing random samples from the appropriate probability distribution [11-12].

Though the proposed model is valid, in general, for any NN based learning system, the problem of multi-layer perceptron based classification using back-propagation learning is considered here for a demonstration of the validity of the model. It is implemented on IRIS data [13]. To demonstrate the utility of the proposed model, a few links are damaged completely during the learning phase (this has the underlying assumption that the training/ learning phase takes a large amount of time compared to testing phase) of the classificatory system. In the testing phase, the performance of such a system is evaluated by measuring the percentage of correct classification. A network with the same configuration (as used in the previous experiment) is then allowed to learn with the same set of training samples (without any component damage). The same set of links which were damaged

during the learning phase of the previous experiment are then damaged (before testing) and the classification accuracy is measured. A comparative study of the perfect classification rates for these two experiments shows that the proposed model provides more robust (better accuracy even with damaging of components) performance.

MODELING AND SAMPLING OF FAILURES IN A NEURAL NETWORK

Modeling of failures

Let us consider a neural network system with N components, where a component could be a node (processor) or a link. (In practical case one can model the nodes and links separately also [10]). During the operation of the network some of its components may fail. We make the following assumptions about the failure process:

(*i*) The system has N identical components at the time instant $t = 0$ (when the operation of the network starts).

(*ii*) If a component fails, it fails for ever (no repair or replacement).

(*iii*) Failure of components occurs at an average rate of μ per unit time.

(*iv*) The probability of an event occurring between time t and $t + h$ depends only on the length of h, i.e., the probability does not depend on either the number of events that has occurred up to time t or the specific value of t, i.e., the probability density function has stationary increments.

(*v*) The probability of a failure during a very small interval of time h is positive but less than one (1), i.e., not certain.

(*vi*) At most one failure can occur during a very small interval of time h.

Let $p_n(t)$ be the probability that the system has n components active at time instant t, i.e., $N - n$ failures during time interval $[0, t]$. It can be shown [11] that under the assumption *s* (*i*)-(*vi*), $p_n(t)$ is given by the formula:

$$p_n(t) = \frac{e^{-\mu t}(\mu t)^{N-n}}{(N-n)!}, \quad n = 1, 2, \ldots, N \tag{1}$$

and

$$p_0(t) = 1 - \sum_{n=1}^{N} p_n(t). \tag{2}$$

Thus we see that $p_n(t)$ is a truncated Poisson distribution with mean μt.

If $f(t)$ is the probability density function (pdf) of the inter-failure time (i.e., time interval between two successive failures), then it can be shown that for the earlier. Poisson failure process, $f(t)$ is given by

$$\begin{aligned} f(t) &= \mu e^{-\mu t} \quad t > 0 \\ &= 0 \qquad t \le 0. \end{aligned} \tag{3}$$

Thus, when the failure process is governed by a Poisson distribution, the inter-failure time is described by an exponential distribution (3) with expected value (mean)

$$\begin{aligned} E(t) &= \int_0^\infty t f(t)\, dt \\ &= \frac{1}{\mu}. \end{aligned} \tag{4}$$

Sampling

In order to simulate the failure process, one needs to draw random samples from the exponential distribution (3). Before describing the exact algorithm, let us first, consider the general strategy for sampling from any distribution.

Let $f(x)$ be the *pdf* of the random deviate x, and $F(x)$ be the cumulative density function (*cdf*) of x, i.e.,

$$F(X) = \int_0^\infty f(t)\, dt. \tag{5}$$

It can be easily shown that the random variable $y = F(x)$ is uniformly distributed over [0, 1], regardless of the distribution of x. Hence, if R is a random number drawn from uniform [0, 1], then $x = F^{-1}(R)$ is a random sample from the *pdf* $f(x)$. Therefore, sampling from any distribution can be done using the following simple method having two steps.

Step 1: Generate a random number R in [0,1] and assign it to $F(x)$.

Step 2: Solve for x from $R = F(x)$.

The above sampling method is known as method of inversion.

Sampling from exponential distribution

For an exponential distribution the *pdf* is

$$\begin{aligned} f(t) &= \mu e^{-\mu t} \quad \mu > 0 \quad t > 0 \\ &= 0 \qquad\qquad\quad t \le 0. \end{aligned} \tag{6}$$

Then

$$F(t) = \int_0^t \mu e^{-\mu x}\, dx = 1 - e^{-\mu t}.$$

If the random number drawn is R then

$$R = F(t)$$

or, $R = 1 - e^{-\mu t}$

or, $t = -\frac{1}{\mu} \ln (1 - R) = -\frac{1}{\mu} \ln R.$ (7)

The last step is possible because if R is a random number on [0, 1] then so is $(1 - R)$ and we can replace $(1 - R)$ by R for convenience.

It has been established before that if the failure process is described by a Poisson distribution, then the time between the occurrence of failures (inter-failure time) must follow the corresponding exponential distribution. Thus in order to simulate the component failure process described by the Poisson distribution with mean μt, over a time period $[0, T]$, all one has to do is to sample the corresponding exponential distribution with mean $1/\mu$ as many times as necessary until the sum of the corresponding exponential random samples generated exceeds T for the first time. It can further be explained as follows.

Suppose R_i is the ith random sample drawn from uniform [0, 1], then

$$t_i = -\frac{1}{\mu} \ln R_i \tag{8}$$

is the ith sample from the exponential distribution (3). Therefore

$$T_i = \sum_{j=1}^{i} t_j \tag{9}$$

gives the time instant when the ith (component) failure occurs. The process is repeated for the maximum number of times (K, say) such that $T_k \leq T$.

MAKING THE LEARNING SYSTEMS ROBUST

Though the methodology that is going to be developed for making neural network based learning systems robust is true for any NN based systems, the present discussion is made by considering a specific type of NN (the multi-layer perceptron). So, let us briefly describe the architecture and working principles of the multi-layer perceptron first.

In general, a multilayer perceptron (MLP) [1] is made up of sets of nodes arranged in layers. Nodes of two different consecutive layers are connected by links or weights, but there is no connection among the elements of the same layer. The layer where the inputs are presented is known as the input layer. On the other hand the output producing layer is called the output layer. The layers in between the input and the output layers are known as hidden layers. The output of nodes in one layer is transmitted to nodes in another layer via links that amplify or attenuate or inhibit such outputs through weighting factors. Except for the input layer nodes, the total input to each node is the sum of weighted outputs of the nodes in the previous layer. Each node is activated in accordance with the input to the node and the activation function of the node. The total input (I_i) to the ith unit of any layer is,

$$I_i = \sum_j w_{ij} o_j \tag{10}$$

with o_j as the output of the jth neuron in the previous layer and w_{ij} is the connection weight between the ith node of one layer and the jth node of the previous layer. The output of a node i is obtained as

$$o_i = f(I_i), \tag{11}$$

where f is the activation function [1]. Mostly the activation function is sigmoidal with the form

$$f(x) = \frac{1}{1 + e^{-(x-\theta)/\theta_0}}. \tag{12}$$

The function is symmetrical around θ, and θ_0 controls the steepness of the function. θ is known as the threshold/bias value.

The back-propagation learning algorithm

For the operation of the multi-layer perceptron, initially very small random values are assigned to the links/weights. In the learning phase (training) of such a network we present the pattern $X = \{x_i\}$, where x_i is the ith component of the vector X, as input and ask the net to adjust its set of weights in the connecting links and also the thresholds in the nodes such that the desired output $\{t_i\}$ is obtained at the output nodes. After this, we present another pair of X and $\{t_i\}$, and ask the net to learn that association also. In fact, we desire the net to find a simple set of weights and biases that will be able to discriminate among all the input/output pairs presented to it. This process can pose a very strenuous learning task and is not always readily accomplished. Here the desired output $\{t_i\}$ basically acts as a teacher which tries to minimize the error.

In general, the output $\{o_i\}$ will not be the same as the target or desired value $\{t_i\}$. For a pattern the error is,

$$E = \frac{1}{2} \sum_i (t_i - o_i)^2 \tag{13}$$

where the factor of one half is inserted for mathematical convenience. The incremental change in weights for a particular pattern p is given by [1]

$$\Delta w_{ji} = \eta \delta_j o_i \tag{14}$$

with

$$\partial_j = -\frac{\partial E}{\partial o_j} f'(I_j). \tag{15}$$

As E can be directly calculated in the output layer, for the links connected to the output layer the change in weight is given by

$$\Delta w_{ji} = \eta \left(-\frac{\partial E}{\partial o_j}\right) f'(I_j)\, o_i. \tag{16}$$

For the weights which do not affect the output nodes directly

$$\frac{\partial E}{\partial o_j} = \sum_k (-\delta_k)\, w_{kj}. \tag{17}$$

Hence

$$\Delta w_{ji} = \begin{cases} \eta\,(-\dfrac{\partial E}{\partial o_j}) f'(I_j)\, o_i \\ \eta\,(\sum\limits_k \delta_k w_{kj}) f'(I_j)\, o_i \end{cases} \tag{18}$$

for the output layer and other layers, respectively. In particular equations (11) and (12), if

$$O_j = f(I_j) = \frac{1}{1 + e^{-(\sum_i w_{ji} o_i - \theta_j)}} \tag{19}$$

then

$$f'(I_j) = \frac{\partial o_j}{\partial I_j} = o_j (1 - o_j) \tag{20}$$

and thus we get

$$\Delta w_{ji} = \begin{cases} \eta\,(-\dfrac{\partial E}{\partial o_j})\, o_j\,(1 - o_j)\, o_i \\ \eta\,(\sum\limits_k \delta_k w_{kj})\, o_j (1 - o_j)\, o_i \end{cases} \tag{21}$$

for the output layer and other layers, respectively.

It may be mentioned here that a large value of η corresponds to rapid learning but might result in oscillations. A momentum term of $\alpha\, w_{ji}\,(t)$ can be added to increase the learning rate and thus expression (14) can be modified as

$$\Delta w_{ji}\,(t+1) = \eta \delta_j\, o_i + \alpha\, \Delta w_{ji}\,(t) \tag{22}$$

where the quantity $(t + 1)$ is used to indicate the $(t + 1)$th time instant, and α is a proportionality constant. The second term is used to specify that the change in w_{ji} at $(t + 1)$th instant should be somewhat similar to the change undertaken at instant t.

Making the learning systems robust

In any system, components may fail with passage of time. In case of NN based systems the components are the neurons/nodes and links. So, in such systems some of the neurons or links or both may get damaged in course of time. NN based information processing systems are normally claimed to be robust under components failure as the NN architectures involve massive processing elements and connectivity among them (mostly with redundant components). Thus even if we damage some of the components in the learning phase, the strength of other links and biases of other nodes will automatically get adjusted so as to compensate for this damage during the rest part of learning resulting in higher classification accuracy during testing phase. This basic property can be used to design robust learning systems.

Let T be the total time required for I iterations to learn the parameters of a NN on a monoprocess system (which can roughly be estimated from previous experiments). Then the time required per iteration is

$$T' = \frac{T}{I}. \tag{23}$$

Note that time spent for testing is negligible compared to T (learning time). Also let t be the time required for updating a single node (includes collecting the input to it, transforming the input to output, updating the links connected to it). In practical case, t may not be equal for all nodes. Suppose there is an n-layer network; where the operation between layers is strictly sequential and operation among the nodes in the same layer are parallel. Let there be n_1 nodes in the first layer, n_2 nodes in the second layer, n_3 nodes in the third layer and so on. As no time is spent in the input layer,

$$t\, n_2 + t\, n_3 + ... + t\, n_n = T'. \tag{24}$$

So,

$$t = \frac{T'}{n_2 + n_3 + ... + n_n}. \tag{25}$$

Since there are $(n - 1)$ layers which operate strictly sequentially, the time required per iteration for parallel implementation is

$$(n-1)\, t = \frac{(n-1)T'}{n_2 + n_3 + ... + n_n}. \tag{26}$$

Thus total time (for I iterations) required for parallel implementation is

$$T_p = \frac{(n-1)T'}{n_2 + n_3 + ... + n_n} I = \frac{(n-1)T}{n_2 + n_3 + ... + n_n}. \tag{27}$$

If D components are to be damaged during this process, then the parameter (μ) for the Poisson distribution (1) is estimated as,

$$\mu = \frac{D}{T_p}. \tag{28}$$

Let the inter-damage time periods (samples) be t_j, $j = 1,2, ..., L$ $(L \approx D)$. Now for each of the L time instants select a component to be damaged. In other words, select L components and damage the ith selected component after T_i seconds, where

$$T_i = \sum_{j=1}^{i} t_j. \tag{29}$$

Now if the ith component is to be damaged at a time

T_i, and T_i falls in kth iteration, then for the kth and subsequent iterations assume the ith component as damaged.

Thus we notice that the components are getting damaged at various moments of learning. Only one component may get damaged at the end of the learning phase. Since the learning process continues even after damaging of some of the components, the adjustment of the other components will be in a way so as to get the optimum performance (i.e., with this configuration only the error value is minimized) thereby compensating for this damage. Now if these components would have been failed after the completion of learning, the performance of the system would degrade a lot. Since in a NN based system failure of components is natural, it is better to incorporate this fact (by deliberately assuming some of the components as damaged) while learning the parameters thereby achieving robust performance. Please note that even though there is no change in the learning algorithm, incorporation of damaging of components during learning will make the system robust.

A similar discussion can be made on treating the links as separate components and assuming that the total time is spent on updating the links.

We have conducted the present simulation study by damaging the links only. This is due to the fact that the number of nodes in the hidden layer is very small and the usefulness of the proposed algorithm cannot be demonstrated rigorously. As there remains no redundancy in the input and output nodes the present technique may not be useful to study the failure process of these nodes. A similar study can be done by damaging a combination of links and nodes.

RESULTS AND DISCUSSION

The proposed method is implemented and tested on the standard MLP based classification problem. Implementation is done on the IRIS data set (150 samples) which has 4 input features and 3 classes. The network architecture chosen is 4-3-3, i.e., it has 4 input nodes, one hidden layer with 3 nodes and 3 nodes in the output layer. For different simulations 10%, 20% and 50% samples were taken randomly for training; and the whole set (of 150) was taken for testing. The percentage of correct classification (with different training sizes and parameter values) are depicted in Tables 1-4. Similar experiments were performed by damaging 2 links and 4 links (out of $4 \times 3 + 3 \times 3 = 21$) while the network is in the training phase. The classification accuracies are also put in the same set of tables. The damaging of the links were performed according to the procedure described earlier. Another set of experiments were performed using the learned network by damaging the same set of links (i.e., those links which were damaged during training phase of the formerly described set of experiments) in the

TABLE 1 Percentage of correct classification with learning rate $\eta = 0.2$ and momentum value $\alpha = 0.2$

Training size (in %)	Usual MLP	Damaged during learning		Damaged during testing	
		2-components	4-components	2-components	4-components
10	96	96	66	96	66
20	96	96	96	96	92
50	98	66	64	66	33

TABLE 2 Percentage of correct classification with learning rate $\eta = 0.2$ and momentum value $\alpha = 0.5$

Training size (in %)	Usual MLP	Damaged during learning		Damaged during testing	
		2-components	4-components	2-components	4-components
10	96	98	94	33	33
20	95	96	62	93	33
50	98	98	66	98	66

TABLE 3 Percentage of correct classification with learning rate $\eta = 0.5$ and momentum value $\alpha = 0.2$

Training size (in %)	Usual MLP	Damaged during learning		Damaged during testing	
		2-components	4-components	2-components	4-components
10	98	96	96	97	95
20	98	94	94	33	33
50	97	97	97	33	33

TABLE 4 Percentage of correct classification with learning rate $\eta = 0.5$ and momentum value $\alpha = 0.5$

Training size (in %)	Usual MLP	Damaged during learning		Damaged during testing	
		2-components	4-components	2-components	4-components
10	93	93	93	95	95
20	98	98	96	62	33
50	96	96	96	66	33

testing phase and evaluating the classification accuracy on this damaged architecture.The percentage scores obtained by damaging 2 and 4 links (of this settled network) are also included in Tables (1 - 4).

From the tables we notice that in most of the cases (with the same network architecture and same set of parameter values) the classification accuracy is more when the links were damaged during the training/learning phase than they were damaged during testing. For example let us consider the situation with $\eta = 0.5$ and $\alpha = 0.2$ (Table 3). For 20% training sample, the classification accuracy is 98% without any damage. The accuracy reduces to 94% with damaging 2 and 4 links during learning. But, if the same links are damaged while testing (on the learned network) the classification accuracy is drastically reduced to 33%. Thus it is advisable to consider the possibility of component failure in NN while learning its parameters thereby making the system more robust.

CONCLUSIONS AND FURTHER SUGGESTIONS

A method to make neural network based learning systems robust with respect to component failure (damaging of nodes/links) is suggested in the present investigation. The components are allowed to be damaged at different instants of the learning phase; thereby allowing time to the new architecture (with fewer components) to adjust its parameters so as to provide optimum performance. Thus the overall performance will not be deteriorated much due to this damage. The damaging/component failure process has been modeled as a Poisson process. The instants or moments of damaging are chosen by statistical sampling. The components to be damaged are determined randomly. As an illustration, the proposed model is implemented and tested on the back-propagation learning based classification algorithm. A comparative study of the scores obtained by the proposed learning system and the standard back-propagation algorithm establishes the superiority of the proposed algorithm.

The work presented here shows a preliminary attempt on designing NN based robust learning systems. A number of problems related to this contribution remains to be investigated. The most important and natural extension of the present concept will be to develop algorithms which can handle systems with partially damaged components. Fuzzy set theoretic approach seems to be a viable alternative for this task. Further, generalization of this model so as to handle neuro-fuzzy learning systems which deal with linguistic or fuzzy input vectors and provide output with multiple class labels and certainty factors will constitute another important study.

ACKNOWLEDGEMENT

A part of this work was done when Dr Ashish Ghosh held a research fellowship of the Ministry of Education, Science, Sports and Culture, Govt. of Japan.

REFERENCES

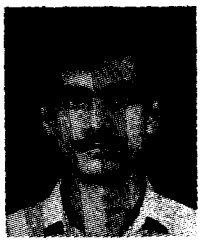
1. D E Rumelhart, J McClelland, *et al, Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol 1, MIT Press, Cambridge, MA, 1986.
2. S Grossberg (ed.), *Neural Networks and Natural Intelligence*, MIT Press, Reading, MA, 1988.
3. Y H Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison Wesley, Massachusetts, 1987.
4. T Kohonen, *Self-organization and Associative Memory*, Springer Verlag, Berlin, 1989.
5. P D Wassermann, *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York, 1990.
6. G E Hinton, Connectionist learning procedure, *Artificial Intelligence*, vol 40, pp 185-235, 1989.
7. S E Fahlman & C Lebiere, The cascade correlation learning architecture, in *Advances in Neural Information Processing Systems* 2, D S Touretzky, editor, Morgan Kaufmann, 1990.
8. T C Lee, A M Peterson & J C Tsai, A multi-layer feed-forward neural network with dynamically adjustable structures, *Proceedings IEEE International Conference on Systems, Man, and Cybernetics*, pp 367-369, 1990.
9. A S Weigend, D E Rumelhart & B A Huberman, Generalization by weight elimination with application to forecasting, in *Advances in Neural Informaton Processing Systems 3*, R P Lippmann, J E Moddy and D S Touretzky, (eds), Morgan Kauffmann, CA, pp 875-882, 1991.
10. A Ghosh, N R Pal & S K Pal, Modeling of component failure in neural networks for robustness evaluation: an application to object extraction, *IEEE Transactions on Neural Networks*, vol 6, pp 648-656, 1995.

11. H A Taha, *Operations Research: An Introduction*, Macmilan Publishing Co, New York, 1982.

12. A Gupta, *Groundwork of Mathematical Probability and Statistics*. Academic Publishers, New Delhi, India, 1983.

13. R G Johnson & D W Wichern, *Applied Multivariate Statistical Analysis* Prentice Hall, Inc, New Jersey, 1982.

AUTHOR



Ashish Ghosh is a Lecturer at the Machine Intelligence Unit, Indian Statistical Institute, Calcutta. He received the BE degree in Electronics and Telecommunication from the Jadavpur University, Calcutta in 1987, and the MTech and PhD degrees in Computer Science from the Indian Statistical Institute, Calcutta in 1989 and 1993, respectively. He received the prestigious and most coveted Young Scientists award in Engineering Sciences from the Indian National Science Academy in 1995; and in Computer Science from the Indian Science Congress Association in 1992. He has been selected as an Associate of the Indian Academy of Sciences, Bangalore in 1997. He visited the Osaka Prefecture University, Japan with a Post-doctoral fellowship during October 1995 to March 1997; and Hannan University, Japan as a visiting scholar during September-October 1997. His research interests include Evolutionary Computation, Neural Networks, Image Processing, Fuzzy Sets and Systems, and Pattern Recognition.