Efficient Weight Initialization Method in Multi-Layer Perceptrons

Jaemin Han, Shijoong Sung, and Changho Hyun
College of Business Administration
Korea University

Abstract

Back-propagation is the most widely used algorithm for supervised learning in multi-layer feed-forward networks. However, back-propagation is very slow in convergence. In this paper, a new weight initialization method, called rough map initialization, in multi-layer perceptrons is proposed. To overcome the long convergence time, possibly due to the random initialization of the weights of the existing multi-layer perceptrons, the rough map initialization method initialize weights by utilizing relationship of input-output features with singular value decomposition technique. The results of this initialization procedure are compared to random initialization procedure in encoder problems and xor problems.

1. Introduction

Artificial neural net models, as a brain metaphor of information processing, have been studied for many years in the hope of achieving human-like performance in the fields of speech and image recognition (Vemuri, 1988). The basic idea of neural net came from the perceptron by Rosenblatt (Rosenblatt, 1962). The perceptron is a feedforward network with one output neuron that learns a separating hyperplane in a pattern space. n linear F_X neurons feed forward to one threshold output F_Y neuron. However, the perceptron only separates only linearly separable sets of patterns and perceptrons cannot separate linearly inseparable pattern sets (Minsky and Papert, 1969). Most interesting collections of pattern sets are linearly inseparable.

Ackley, Hinton, and Sejnowski proposed back-propagation algorithm in multi-layer perceptron and solved linear inseparability problem. Currently, back-propagation is the most widely used algorithm for supervised learning in multi-layer feed-forward networks. The basic idea of the back-propagation learning algorithm is the repeated application of the chain rule to compute the influence of each weight in the network with respect to an arbitrary error function (Rumelhart 1986).

The total input, net_{pj}, to units j is a linear function of the outputs, o_{pi}, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{1}$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, o_{pj}, which is a non-linear function of its total input

$$o_{pj} = f_j(net_{pj}) = \frac{1}{1+e^{-net_{pi}}} \tag{2}$$

The aim is to find a set of weights that ensure that for each input vector the output vector produced by the network is the same as or sufficiently close to the desired output vector. If there is a fixed, finite set of input-output cases, the total error in the performance of the network with a particular set of weights can be computed by comparing the actual and desired output vectors for every case. The total error, E, is defined as

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \tag{3}$$

where t_{pj} is the desired output, and o_{pj} is the computed output.

To minimize E by gradient descent, back-propagation algorithm compute the partial derivative of E with respect to each weight in the network. This is simply the sum of the partial derivatives for each of the input-output cases. For a given cases, the partial derivatives of the error with respect to each weight are computed in two passes. In forward pass the units in each layer have their states determined by the input they receive from units in lower layers using equations (1) and (2). The backward pass propagates derivatives from the top layer back to the bottom.

One way of using $\partial E/\partial w$ is to change the weights after every input-output case. The simplest version of gradient descent is to change each weight by an amount proportional to the accumulated $\partial E/\partial w$. To increase the learning speed, Rumelhart included a momentum to modify the generalized delta rule.

$$\Delta w_{ji}(n+1) = -\eta(\delta_{pj} o_{pj}) + \alpha \Delta w_{ji}(n) \qquad (4)$$

Back-propagation start with small random weight to break symmetry. Original back-propagation procedure uses a pure gradient-descent technique, a lot of iterations are needed to get reasonable solutions. The choice of the learning rate η, which scales the derivative, and α have an important effect on the time needed until convergence is reached. If it is set too small, too many steps are needed to reach an acceptable solution; on the contrary large learning rate will possibly lead to oscillation, preventing the error to fall below a certain value. Many algorithms have been proposed so far to deal with the above problem, e.g. by introducing a momentum term or doing some sort of parameter adaptation during learning (Fahlman 1988, Riedmiller 1993). However, the basic problem remains. The random initialization in multi-layer perceptrons seems to prevent the learning algorithm producing output in a reasonable time.

2. The Weight Initialization Method

The brain is believed to have a strong ability of self-organization (AMARI, 1983). Its characteristics are modified according to the nature of the environment from which signals are obtained, so that it adapts to the information structures of the environment. And it is also believed that fine structures of the brain are formed by self-organization to be compatible with outer world. In this perspective, back-propagation algorithm resembles brain mechanisms in that it modifies its weight according to the environment, even though it uses supervised learning. However, a rough map is necessary before self-organization takes place. An initial rough map seems to be developed by another mechanism which takes place at an early stage of development.

The basic idea of the weight initialization method presented in this paper is very simple. Back-propagation uses small random number usually from -1 to 1 as an initial weight. If somehow we can encode representatives of the examplar patterns into the initial weight matrices, and use learning algorithms to fine tuning, the convergence time can drop rapidly. We call this initial weights as rough map initialization. This section describes one of the method that can be used to derive rough map of the weights.

We can view the activations of Multi-layer perceptrons simply as doing mappings between input and output patterns. That is, let X as input matrix, W_1 and W_2 as weight matrices of Input-Hidden and Hidden-Output respectively and Y as output matrix.

How to Get Rough Structure of Weight Matrix

Define F[Y] and F^{-1}[Y]

$$F[Y] = \begin{bmatrix} f(y_{11})\ f(y_{12})\ \ldots\ f(y_{1n}) \\ f(y_{21})\ f(y_{22})\ \ldots\ f(y_{2n}) \\ \cdots \\ f(y_{m1})\ f(y_{m2})\ \ldots\ f(y_{mn}) \end{bmatrix}$$
$$F^{-1}[Y] = \begin{bmatrix} f^{-1}(y_{11})\ f^{-1}(y_{12})\ \ldots\ f^{-1}(y_{1n}) \\ f^{-1}(y_{21})\ f^{-1}(y_{22})\ \ldots\ f^{-1}(y_{2n}) \\ \cdots \\ f^{-1}(y_{m1})\ f^{-1}(y_{m2})\ \ldots\ f^{-1}(y_{mn}) \end{bmatrix} \tag{6}$$

In 2 layer perceptions, the middle layer output H is computed

$$H = F[XW_1]\ ,\quad where\ 0 \le h_{ij} \le 1 \tag{7}$$

And the output layer output Y is computed as follows.

$$Y = F[HW_2] \tag{8}$$

If we know H, we can derive both W_1, W_2

$$\begin{aligned} F[XW_1] &= H \\ XW_1 &= F^{-1}[H] \\ W_1 &= X^{+}F^{-1}[H] \\ F[HW_2] &= Y \\ HW_2 &= F^{-1}[Y] \\ W_2 &= H^{+}F^{-1}[Y] \end{aligned} \tag{9}$$

Where, superscript + means pseudoinverse - a way of deciding on a particular inverse when there is no normal way to decide. Note that psuedoinverse computes least squares solution to the problem.

Psuedoinverse can be computed by singular value decomposition.

$H = Q_1 \Sigma Q_2^T$

The columns of $Q_{1(m*m)}$ are eigenvectors of HH^T,

and the columns of $Q_{2(n*n)}$ are eigenvectors of H^TH.

The r singular values on the diagonal of $\Sigma_{(m*n)}$ are the square roots of the nonzero eigenvalues of both HH^T and H^TH.

The pseudoinverse H is $H^+ = Q_2\Sigma^+Q_1^T$.

If the correct patterns of H are known a priori, then the problem can be solved easily without weight update algorithm. However, the structure of H is not known, so starting with randomized H matrix can be an alternative. With H given as a starting point, we can compute the weights W_1 and W_2, and apply learning algorithm.

Even in this case, there is a significance between random initialization and rough map initialization. In conventional random initialization method, the initial weights, W_1, W_2 contributes nothing to producing correct outputs. So the weight update algorithm takes care of the rest. In rough map initialization method, the weights contributes some portion to produce correct outputs. And the burden of weight update algorithm decreases.

We simulated two kinds of problems. One is encoder problem. In this case when the number of hidden layer is sufficiently large and we apply rough map initialization, there might be almost no need to adopt learning algorithm. When the number of hidden units is very small, the rough map initialization might have no competitive advantage over random initialization. In this case, in constructing pseudoinvese we loose too much information. And the rough map initialization might not will generally better than random initialization. The other problem we tested is XOR. It can view as an instance of more general form of parity problem. Parity is a very difficult problem to learn in multi-layer perceptrons because the most similar patterns require different answers. In this case, there might be no difference between random and rough map initialization. But it is one of the extreme case. We think that most other problems lies in between.

3. Simulation Results

3.1 Methodology

In the following experiments, learning time was measured as the average number of epochs required until the task was learned in 20 different runs. An epoch is defined as the period in which every pattern of the training set is presented once and the weights are updated according to errors. So when the initial weights satisfy the task completion criterion, the epoch is calculated as zero.

In each tests, we compared results by number of epochs from conventional random and rough map initialization. Although preprocessing is necessary in rough map initialization, it actually takes almost no time. So comparison by epoch has little bias. In simulating the problems, there were some problems that fail to converge in a fixed number of iterations. So reporting average number of epochs only is not enough to compare the results. Although Fahlman allowed the program to restart a trial, with new random weights, whenever the network has failed to converge after a certain number of epochs, deciding the threshold of max iteration is subjective (Fahlman 1988). So in this paper, we reported the number of failures and the average of successful trials.

The well working region of the learning parameters might be different between random and rough initialization because of the difference of the range in initial weights. So comparing the results with the same η and α may not show the real differences. So we tested various combinations of learning rate and moment ranging from 0.1 to 0.9 respectively, stepping 0.2. And the best result was presented.

Learning is complete, if a binary criterion is reached. That is, the activation of each unit in the output layer is smaller than 0.4 if its target value is 0.0, and bigger than 0.6 if its target value is 1.0.

We compared whether random and rough map initialization is different with conventional back-propagation algorithm. However, It should be remembered that it is learning algorithm independent, so we can also apply other learning algorithms, such as Quickprop and RPROP (Fahlman 1988, Riedmiller 1993).

3.2 The 10-5-10 Encoder Problem

The first problem to be described is the 10-5-10 Encoder task. The task is to learn an auto-assocation between 10 binary input and output patterns. The network consists of 10 neurons in both the input and the output layer, and a hidden layer of 5 neurons. [Table 1] shows the resuls obtained from the simulations. We see the significance difference between random initialization and rough map initialization. Rough map initialization is more than 8 times faster than random initialiazations.

[Table 1] The 10-5-10 Encoder Problem

Problem	Init Method	η	α	No. Failed	MAX	MIN	AVG	S.D.
10-5-10	Random	.7	.5	0	220	94	144.35	33.01
	Rough Map	.7	.9	0	43	9	17.10	7.65

3.3 The 12-n-12 Encoder Problem

The task of the 12-n-12 Encoder is to learn an auto-assocation of 12 input and output patterns. The network consists of both 12 neurons in the input and the output layer, and a hidden layer of n neurons. We set 'n' from 3 to 12 neurons. With this experiment, we can see how the single changes in the number of hidden units affects learning time.

Rough map initialization is much better than random initialization in all cases. Even in the hard problem, that is when n is set to 3, rough map initialization takes only a third of random initialization. And the differences increases when n is increased. When n is greater than 9, we found lots of cases required no training at all.

[Table 2] The 12-n-12 Encoder Problems

Problem	Init Method	η	α	No. Failed	MAX	MIN	AVG	S.D.
12-3-12	Random	.9	.5	0	1019	472	729.25	138.21
	Rough Map	.9	.9	2	1099	83	211.00	231.88
12-4-12	Random	.9	.5	0	354	153	272.60	51.23
	Rough Map	.9	.7	1	292	83	97.16	72.06
12-5-12	Random	.9	.3	0	328	125	207.95	55.39
	Rough Map	.9	.9	0	170	9	28.50	33.60
12-6-12	Random	.9	.3	0	225	93	167.00	35.67
	Rough Map	.7	.9	0	108	8	19.00	21.22
12-7-12	Random	.9	.3	0	211	86	145.60	34.71
	Rough Map	.7	.9	1	19	4	8.79	3.24
12-8-12	Random	.9	.1	0	204	81	126.30	32.58
	Rough Map	.9	.9	2	8	2	4.72	2.02
12-9-12	Random	.7	.3	0	239	90	122.25	34.52
	Rough Map	.7	.7	3	11	0	2.18	1.58
12-10-12	Random	.5	.9	0	251	100	140.60	41.35
	Rough Map	.9	.9	2	2	0	0.33	0.67
12-11-12	Random	.7	.1	0	177	87	133.10	27.83
	Rough Map	.5	.7	0	0	0	0.00	0.00
12-12-12	Random	.7	.1	0	214	78	119.95	29.64
	Rough Map	.9	.9	2	2	0	0.33	0.67

3.4 XOR

The final problem is exclusive-or problem. When the number of input is n, and our aim is to decide whether even or odd number of 1's exists in the input vectors, this problem can be an parity problem. In this problem, the results are not so fine as encoder problem. But the results seem to be a promising one.

[Table 3] The XOR Problem

Problems	Init Method	η	α	No. Failed	MAX	MIN	AVG	S.D.
XOR	Random	.7	.9	2	120	62	84.50	19.93
	Rough Map	.9	.9	6	109	12	32.36	24.63

4. Conclusions and Future Work

In this paper, we compared conventional random and rough map initialization. The results seem to be a promising one. Further research is needed in this direction.

The concept described in this paper can be used in weight update algorithm. The objective of multi-layer perceptron is finding a set of weights that minimize sum squared errors. The basic idea of utilizing input-output relations can be encoperated in the learning algorithm. In this perspective, the objective of the multi-layer perceptron can be finding a set of hidden layer patterns that minimize sum squared errors. Currently we are working to this direction.

In this paper, we randomized patterns of hidden layer. But by utilizing input-output patterns, different weight initialization strategy can be pursued using some kinds of clustering techniques.

References

Amari, S., "Field Theory of Self-Organizing Neural Nets," IEEE Transaction on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, September/October 1983.

Fahlman, S.E., "An Empirical Study of Learning Speed in Back-Propagation Networks," CMU-CS-88-162, September 1988.

Hinton, G.E., "Connectionist Learning Procedures," Artificial Intelligence, pp.

185-234, 1988.

Jones, W.P. and J. Hoskins, "Back-Propagation - A generalized delta learning rule," BYTE, October 1987.

Lippmann, R.L., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, pp. 4-22, April 1987,

Minsky, M. and S. Papert, Perceptrons, Cambridge, MA:MIT Press, 1969.

Riedmiller, M. and H. Braun, "RPROP - A Fast Adaptive Learning Algorithm," to appear in Proceedings of ISCIS VII, 1993.

Rumelhart, D.E., "Learning representations by back-propagating errors," Nature, 323:533-536.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," PDP Vol. 1, pp 318-362, 1986.

Rosenblatt, F., Principles of neurodynamics, New york: Spartan, 1962.

Strang, G., Linear algebra and its application, 2nd ed., Academic press, 1986.

Vemuri, V., "Artificial Neural Networks: An Introduction," IEEE, 1988.