ISSN:0975-9646

Analysis of Optimization Techniques for Feed Forward Neural Networks Based Image Compression

Neha Relhan, Manoj Jain

CSE Department, Lingaya's University
Lingaya's University, Faridabad, India

***Abstract*— This paper reviews various optimization techniques available for training multi-layer perception (MLP) artificial neural networks for compression of images. These optimization techniques can be classified into two categories: Derivative-based and Derivative free optimization. The former is based on the calculation of gradient and includes Gradient Descent, Conjugate gradient, Quasi-Newton, Levenberg-Marquardt Algorithm and the latter cover techniques based on evolutionary computation like Genetic Algorithms, Particle Swarm Optimization. The core of this study is to investigate the most efficient and effective training algorithm for use in image compression.**

***Keywords*— Multilayer Perceptron, Image Compression/ Decompression, Optimization, Backpropagation**

I. INTRODUCTION

Images require large amounts of memory for storage and the transmission of image from one computer to another over the web can be very time consuming. Approximately 224 GB is needed to store an uncompressed two-hour SD movie. Therefore, to fit such movie on a standard DVD-9, data must be compressed by a factor of approximately 26.3. Image Compression addresses the problem of reducing the amount of data required to represent an image as well as to maintain the visual quality of an image.. By using compression techniques, it is possible to remove some of the redundant information contained in images, thus requiring less storage space and less time to transmit. The existing traditional techniques mainly are based on reducing redundancies in coding, interpixel and psycho visual representation [1]. New soft computing technologies such as neural networks are being developed for image compression. Adaptive learning, self-organisation, noise suppression, fault tolerance, and optimized approximations are some main reasons that encourage researchers to use artificial neural networks as an image compression approach. The Multilayer Perceptron (MLP) network which uses Back propagation training algorithm provides simple and effective structures [9]. The compression of images by three-layer Back-Propagation Neural Networks (BPNN) is investigated by many researchers. Learning algorithms has significant impact on the performance of neural networks. This paper presents a study that shows the effect of using different optimization algorithms on the performance of Multilayer Feed Forward Artificial Neural Network (MFFANN) in image compression. The paper starts with an Introduction followed by an overview of Multilayer Perceptron Artificial Neural Network approach in image compression. It then explains the functionalities of different learning algorithms and finally the conclusion is presented.

II. IMAGE COMPRESSION USING THREE-LAYERED MLP

This section defines the architecture of a Multilayer Feed Forward Neural network [9] that is used to compress images .The simplest structure is illustrated in Fig 1. It consists of one Input Layer and one Output Layer with N neurons each and one Hidden Layer with K neurons. Compression can be achieved by allowing the value of the number of neurons at the hidden layer, K, to be less than that of neurons at both input and output layers ($K \leq N$).

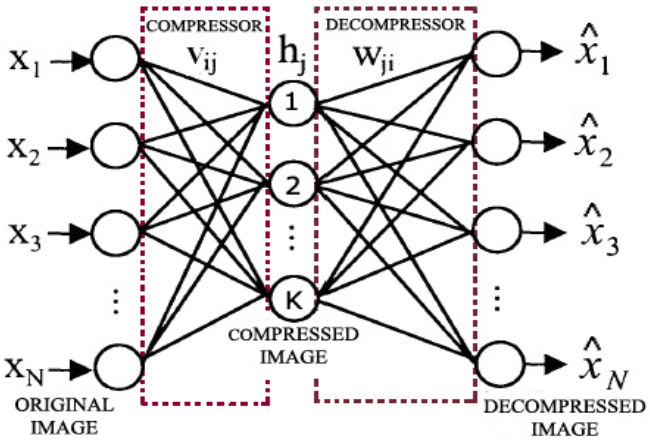

Fig. 1 Basic Neural network structure for image compression/decompression

Following steps are required for this process:

A. Image Preprocessing

Input values for network training are created in this step. The input image is split up into blocks or vectors of 8×8 pixels. Normalization of each block from integer values from the interval [0,255] to real numbers from the interval [0,255] is done. The 2-dimensional blocks are transformed into 1-dimensional vectors using linearization methods like scanning rows, columns. The Normalized and Linearized input vector and the desired output vector is presented to the input neurons of the network as it is necessary that desired output vectors should be equal to input vectors to achieve compression.

B. Training

Figure1 Network is trained using one of the chosen learning algorithms. When the network achieves its goal (calculated and desired weight minimization) in a certain number of

iterations, its weight matrices have to be stored in parameters. In accordance with the structure of neural network shown in Figure1, the operation for adjusting weights for compressing and de-compressing can be described as the following equations.

$$H_j^{in} = \sum_{i=1}^{N} V_{ij} X_i \ , \quad h_j = f(H_j^{in}); \ 1 \le j \le K \qquad (1)$$

$$\hat{X}_j^{in} = \sum_{j=1}^{K} W_{ji} h_j \ , \quad g(\hat{X}_j^{in}); \ 1 \le i \le N \qquad (2)$$

In the above equations, *f* and *g* are the activation functions which can be linear or nonlinear. V_{ij} and Wji represent the weights of compressor and de-compressor, respectively. The extracted $N \times K$ transform matrix in compressor and $K \times N$ *in* de-compressor of linear neural network are in direction of PCA transform. This transform provides optimum solution for linear narrow channel type of image compression and minimizes the mean square error between original and reconstructed image. The training process of the neural network structure in Fig.1 is iterative and is stopped when the weights converge to their true values. In real applications the training is stopped when the error of equation (3) reaches to a threshold or maximum number of iterations limits the iterative process.

$$E = \frac{1}{2} \sum_{k=1}^{N} (X_k - \hat{X}_k)^2 \qquad (3)$$

C. Compression

New inputs and desired outputs are fed into input neurons. Using previously stored weights, hidden layer output is calculated and then observed values are quantified with 8-bits and remembered. Stored data is compressed image and image dimension in pixels is also remembered to achieve successful decompression. Example, to compress an image block of 8×8, 64 inputs and output neurons are required. In this case, if the number of hidden neurons is 16 (i.e. block image of size 4× 4), the compression ratio would be 64:16=4:1. In general, the compression ratio of the basic network illustrated in the Fig. 1 for an image with *n* blocks is computed as

$$CR = nNBI / n\,KB_H = NB_I / KB_H \qquad (4)$$

here B_I and B_H are the number of bits needed to code the output of input and hidden layers and *N* and *K* are the number of neurons in the input and hidden layers, respectively.

D. Decompression

To reconstruct compressed image, stored data (compressed image) is read and set as hidden layer outputs. Again using the stored weight matrices, network output is calculated for the hidden layer and the output layer using weights between them and thus leads to creation of reconstructed image.

III. NETWORK LEARNING ALGORITHM

In this study two categories of optimization algorithms are considered i.e. derivative-based and derivative-free.

A. Derivative-based

The basic algorithm for training of network is backpropagation. On applying some modifications to the primary algorithm, other more sophisticated techniques are produced. They all generally use gradient of the performance function to identify the way how to modify weights and to minimize the function. ΔE (w) is gradient vector (g) of error function is defined as follows:

$$\Delta E(w) = \partial E / \partial w \qquad (5)$$

Following presents the optimization techniques for weight updation in neural networks:

- Gradient descent(GD)
- Conjugate Gradient(CG)
- Quasi Newton(QN)
- Levenberg Marquardt(LM)

1) Gradient descent: In this technique [2, 7] the adjustments applied to the weight vector are in the direction opposite to the gradient vector Δ E (w).

$$w_{n+1} = w_n + \Delta w_n \qquad (6)$$

$$\Delta w_n = -\eta_n g_n + \alpha \Delta w_n \qquad (7)$$

where *g* is gradient vector, η is the learning rate or step size and *α refers* to the momentum. Gradient descent only indicates the direction to move, however the step size or learning rate needs to be decided as well. Too low a learning rate makes the network learn very slowly, while too high a learning rate will lead to oscillation. One way to avoid oscillation for large *η* is to make the weight change dependent on the past weight change which is shown by α.

2) Conjugate Gradient: The conjugate-gradient method [2, 7] reduces oscillatory behavior and adjusts weight according to the previously successful path directions as it uses a direction vector which is a linear combination of past direction vectors and the current negative gradient vector. Let p(*n*) denotes the direction vector at the *n*th iteration and the weight vector of the network is shown as

$$w_{n+1} = w_n + \eta_n p_n \qquad (8)$$

The initial direction vector, p(0), is set equal to the negative gradient vector ,g(0) at the initial weight w(0) and the successive direction vectors are computed as a linear combination of the current negative gradient vector and the previous direction vector.

$$p_{n+1} = -g_{n+1} + b_n p_n \quad where \qquad (9)$$

$$b_n = [g^T_{n+1} g_{n+1}] / [g^T_n g_n] \qquad (10)$$

3) Quasi Newton: Quasi newton method [2,7] use an approximation of an inverse Hessian matrix. This method is shown as

$$w_{n+1} = w_n - \eta_n B_n g_n \qquad (11)$$

where B_n is approximate inverse Hessian matrix which is adjusted from iteration to iteration. The step size η_n is chosen by a line search. The important idea behind the method is that two successive iterates x_n and x_{n+1} together with the gradients $\Delta f(x_n)$ and $\Delta f(x_{n+1})$ contain Hessian information.

$$\Delta f(x_{n+1}) - \Delta f(x_n) \approx H(x_n)(x_{n+1} - x_n) \qquad (12)$$

Therefore, at every iteration it would be necessary to choose B_{n+1} to satisfy

$$B_{n+1}\, q_n = z_n \quad (13)$$

Where

$$Z=x_{n+1}-x_n;\ q=\Delta f(x_{n+1}) - \Delta f(x_n) \quad (14)$$

Boyden, Fletcher, Goldfarb and Shanno (BFGS) version:

$$B_{n+1}=B_n+zz^T/q^Tq-B_nqq^TB_n/q^TB_nq + q\,B_n\,q^T+z/z^Tz-Bq/q^TB_nq \quad (15)$$

4) Levenberg Marquardt: Levenberg-Marquardt [2] is a trust region based method that can be applied to the Gauss-Newton method [2] . This method can handle well ill-conditioned matrices J^TJ in Gauss-Newton method altering the equation

$$w_{n+1}=w_n-(J^TJ)^{-1}J^T e \quad (16)$$

to

$$w_{n+1}=w_n-(J^TJ+\lambda I)^{-1}J^T_n e_n \quad (17)$$

where λ is some nonnegative value called learning parameter. It is always possible to choose λ sufficiently large enough to ensure a descent step. The learning parameter is decreased as the iterative process approaches to a minimum.

B. Derivative-free

Also referred to as Evolutionary Algorithms are based on the principles of biological evolution such as genetic inheritance and natural selection. These are stochastic population-based global search methods that start with an initial population of candidate individuals for an optimal solution. Two of the popular developed approaches are:

- Genetic Algorithms(GA)
- Particle Swarm Optimization(PSO)

Assuming an optimization problem with N_{var} input variables and N_{pop} individuals (chromosomes or particles), the population at the kth iteration is a matrix $\boldsymbol{P}(k)_{Npop\times Nvar}$ of floating-point elements, denoted by $p^k_{m,n}$, with each row corresponding to an individual.

1) Genetic Algorithms: GA[5,8,14] begins with a population of random chromosomes. Each corresponds to a vector with N_{var} floating-point optimization variables and evaluated by means of its associated cost, which is computed through the cost function E given in (3).

$$chromosome(k,m)=[p^k_{m,1}, p^k_{m,2}, \ldots .. p^k_{m,Nvar}] \quad m=1,2,\ldots .N_{pop} \quad (18)$$

$$cost(k,m)=E(chromosome(k,m)) \quad (19)$$

Based on the cost associated to each chromosome, the population evolves through generations with the application of genetic operators, such as: selection, crossover and mutation. Population selection is performed after the N_{pop} chromosomes are ranked from lowest to highest costs. Then, the most-fit chromosomes are selected to form the mating pool and the rest are discarded to make room for the new offspring. Mothers and fathers pair in a random fashion. Each pair produces two offspring that contain traits from each parent. In addition, the parents survive to be part of the next generation. After mating, a fraction of chromosomes in the population will suffer mutation. Then, the chromosome variable selected for real-value mutation is added to a normally distributed random number. A GA is an iterative process. Each iteration is called a generation. A typical number of generations for a simple GA can range from 50 to over 500. A common practice is to terminate a GA after a specified number of generations and then examine the best chromosomes in the population. If no satisfactory solution is found, then the GA process is restarted.

2) Particle Swarm optimization: PSO [8, 14] is a very simple natural optimization algorithm, based on the behaviour of swarms in the nature, such as birds, fish, etc. In this Algorithm, a candidate solution is presented as a particle. It uses a collection of flying particles (changing solutions) in a search area as well as the movement towards a promising area in order to get to a global optimum. The PSO algorithm updates the velocities and positions of the particles based on the best local and global solutions.

$$v^{k+1}_{m,n} = C[r_0 v^k_{m,n} + \Gamma_1 r_1 (p_{m,n}^{localbest(k)} - p_{m,n}^k) + \Gamma_2 r_2 (p_{m,n}^{globalbest(k)} - p_{m,n}^k) \quad (20)$$

$$p_{m,n}^{k+1} = p_{m,n}^k + v^{k+1}_{m,n} \quad (21)$$

Here, $v_{m,n}$, is the particle velocity, $p_{m,n}$, is the particle variables, $r0$, $r1$ and $r2$ are independent uniform random numbers. Γ1 and Γ2 are is the cognitive and social parameters, respectively, $p_{m,n}^{localbest(k)}$ and $p_{m,n}^{globalbest(k)}$ are the best local and global solutions, respectively. C is the constriction parameter. Like GA, PSO is also an iterative process. A typical number of iterations can range from 25 to over 200.

If the best local solution has a cost less than the best cost of the current global solution, then the best local solution replaces the best global solution.

IV. CONCLUSION

In this paper three-layered MLP with different optimization algorithms for image compression has been discussed. Based on this study it is concluded that Backpropagation is the most commonly used technique for updating neural network weight parameters. I t has a slow convergence speed and might at times diverge. It may be difficult to implement when no gradient information is available for all activation functions. The BP based gradient descent takes less time during training as compared to Conjugate Gradient methods and Quasi Newton methods. However Quasi Newton performs better in term of minimizing the error. LM and QN algorithm-based BPNN networks are equally efficient. LM algorithm has fastest network convergence rate, followed by BFGS version of QN. GA and PSO are similar in the sense that they are both population-based search approaches and that they both depend on information sharing among their population members to enhance their search processes using a combination of deterministic and probabilistic rules. Although PSO and the GA on average yield the same effectiveness, PSO is more computationally efficient than the GA.

REFERENCES

[1] R. C. Gonzales, R. E. Woods, *Digital Image Processing*, Second Edition, Prentice-Hall.

[2] Jyh-Shing, Roger Jang, Chuen-Tsai Sun, Eiji Mizutani, *Neuro Fuzzy and Soft Computing*, Prentice-Hall.

[3] Robert D. Dony, *Neural Network Approaches to Image Compression,* Proceedings of the IEEE, Vol.83, No.2, Feb1995.

[4] J.Jiang, "Image Compression with Neural Networks-A survey",Sigmal Processing: *Image Communication 14(1999) 737-760.*

[5] Merlo G.,Caram,F, Fernandez V. ,Britos, P. Rossi, B. & Garcia-Martinez R. , *Genetic Algorithm Based Image Compression,* SBAI,1999.

[6] M.Egmont-Petersen, D. de Ridder, H. Handels, *Image Processing with neural networks- a review,* The Journal of Pattern Recognition 35(2002) 2279-2301.

[7] Omer Mahmoud, Farhat Anwar,Momoh Jimoh E. Salami, *Learning algorithm effect on multilayer feedforward neural network performance in image coding*, Journal of Engineering Science and Technology, Vol. 2, No. 2(2007) 188-199.

[8] Jing-Ru Zhang, Jun Zhang, Tat-Ming Lok, Michael R. Lyu, *A hybrid Particle Swarm Optimization-back- propagationalgorithm for feed forward neural network training,* Applied Mathematics and Computation 185(2007) 1026-1037.

[9] Hadi Veisi, Mansour Jamzad, *A complexity-based approach in image compression using neural networks*, World Academy of Science, Engineering and Technology, 59 2009.

[10] Davoud Sedighizadeh and Ellips Masehian, *PSO Methods,Taxonomy & Applications,* International Journal of Computer Theory and Engineering, Vol. 1, No. 5, December 2009.

[11] Venkata Rama Prasad Vaddella, Kurupati Rama, *Artificial Neural Networks for Compression of Digital Images: A Review,* International Journal of Revies in Computing, 2009-2010.

[12] David J. Montana and Lawrence Davis, *Training Feed Forward Neural Networks Using Genetic Algorithms.*

[13] Dubravka Ilic, Ivana Berkovic, *Grayscle Image Compression Using Back propagation Neural Network.*

[14] Rossana M.S. Cruz, Helton M. Peixoto and Rafael M. Magalhaes, *Artificial Neural Networks and Efficient Optimization Techniques for Applications in Engineering.*

[15] Brian Clow, Tony White , *An Evolutinary Race: A Conparison of Genetic Algorithms and Particle Swarm optimization Used for Training Neural Networks.*