A BP Variant with Improved Convergence and Generalization

Ruthi Aloni-Lavi, Yaakov Metzger and Yaakov Stein
Efrat Future Technology Ltd.
110 Yigal Alon St. Tel Aviv 67891, Israel

Abstract

We present a variant of back propagation, which trains faster, gets caught in local minima less frequently, generalizes better for a given amount of training, and is specifically adapted to pattern recognition problems. We demonstrate our claims by simulations of an archetypal classification task.

1 Introduction

To date, the semilinear feedforward network, trained by back propagation (BP) [22] or by one of its many variants, has been the most popular neural network paradigm in applications. There are several justifications for this :

- these networks can be shown to be *universal approximators* [9, 10],
- when used as a black box replacements for classical *pattern classifiers*, their performance is close to optimal [20, 30]
- the *operation* of these networks (as distinct from the *training*) is relatively fast (no stabilization time is required as in Boltzmann machines or Hopfield networks),
- the memory required to store the network is relatively small (as compared with nearest neighbor, hypersphere, or basis function methods),
- BP provides an easily implemented, fully automatic, supervised training algorithm.

There are, however, several disadvantages to the scheme :

- the *false alarm rate* (ie. the probability that the classifier outputs a valid response when presented an input not corresponding to any of the desired classes) is usually high,
- the *architecture* (number of layers, number of neurons per layer) must be prespecified, and, at present, there is no general prescription for doing so,
- there is no *convergence theorem* for networks with hidden layers and the BP algorithm can get caught in poor local minima of the cost function,
- the learning problem is computationally expensive [13] and in practice the training time does not scale up well with the number of input units and patterns.

The high false alarm rate is a fundamental weakness of hyperplane classifiers (as opposed to nearest neighbor or distance based ones) and may be lowered by properly combining the outputs of several different networks [2]. The unknown architecture problem may be eliminated by adopting alternative training algorithms which dynamically vary the architecture (eg. [3, 8, 18]). The disadvantages of the learning algorithm may be ameliorated by many proposed alternatives to the BP philosophy, which explicitly search for internal representations [6], utilize nonlinear optimization techniques [29], linear [25], quadratic [11] or dynamic [24] programming, or even methods for solving stiff ODEs [21]. However, none of these alternatives can viably replace BP in large pattern recognition applications, being limited to a single output unit, or requiring tremendous amounts of computer memory, or being otherwise restricted to small toy problems.

Once turning (perhaps reluctantly) to the BP algorithm as the only practical solution, we are immediately struck by its sluggishness. Small toy problems often converge only after ridiculously large numbers of cycles through the entire training set (see eg. [23]), and even then the generalization

on the test set may be unacceptably poor. Many variations on the basic BP theme have thus been introduced, such as including second order terms [4, 19], empirically changing the weight updating rules [1, 7], individually optimizing each weight's learning rate [12], reducing the number of independent variables [15, 16], using alternative cost functions [14, 26], and so forth [1, 28].

While working on several large pattern recognition applications in a small computer environment, we have developed a BP variant which seems to be widely applicable, much faster than standard BP, conservative in memory, less prone to being caught in unfavorable minima, and which appears to provide superior generalization. The algorithm incorporates three main features and will be described in section 3. Before the description we explain the use of a feedforward network as a pattern classifier, and following it we present the results of simulations justifying our claims.

2 Feedforward Networks for Pattern Recognition Problems

Although feedforward networks are general function approximators, one of the main reasons for their popularity is the facility with which they can be employed as pattern classifiers. A pattern classifier inputs N *features* which may be continuous or discrete in nature, and outputs an identification linking the input to one of M *classes*. We are interested in getting high correct classification rate on the training examples (approximation), but more importantly on valid examples that are not in the training set (generalization). When the input patterns are not from a closed set, ie. input patterns may not correspond to any of the desired classes, we do not wish the classifier to respond to an input not belonging to one of the classes (*false alarm*). Classifiers with adjustable thresholds may trade off misidentification probability against false alarm rate.

The most common implementation of a neural network classifier consists of N input neurons, and M outputs encoded via the so called *selector* representation, wherein there are M output neurons, each trained to take be active (+1) if the input corresponds to its class and inactive (0) otherwise. Thus the classes $1, 2, \ldots M$ are encoded $\{100\cdots0\}$, $\{010\cdots0\}$, $\{001\cdots0\}\ldots\{000\cdots1\}$. Neuron values significantly deviating from 1 and 0 are attributed to confusion, and more than one neuron being active to ambiguity. For 'raw classification' we simply output the index of the most active output neuron, however one may require that activity to exceed a threshold or be significantly higher than that of the next-runner-up. In certain applications the *thermometer* representation $\{100\cdots0\}$, $\{110\cdots0\}$, $\{111\cdots0\}\ldots\{111\cdots1\}$ is more natural. At times more complex output representations might make more sense from a system engineering point of view – for example we might reserve $\log_2 M$ output neurons and use binary coded class numbers, or eg. for character recognition we might conceivably want to directly obtain ASCII codes.

3 Algorithm Details

We preface our presentation by defining some necessary notation. All the neurons in the network are assumed continuous and to be in the range $[\mathcal{L}\ldots\mathcal{U}]$. A neuron i in a hidden or output layer calculates its value (*activity*) as

$$S_i \;=\; g\left(\sum_j J_{ij}\, S_j\right) \tag{1}$$

where the sum runs over all neurons j which feed neuron i. The activation function g is taken to be a sigmoid curve on the appropriate range and is most popularly taken to be the solution of the generalized *logistic* differential equation $g' = \frac{\beta}{\mathcal{U}-\mathcal{L}}(\mathcal{U}-g)(g-\mathcal{L})$, namely

$$g(h) \;=\; \mathcal{L} + \frac{\mathcal{U}-\mathcal{L}}{1+e^{-\beta h}}\,. \tag{2}$$

For example, when $\mathcal{L}=0$ and $\mathcal{U}=1$ it is the Fermi function $g(h)=\frac{1}{1+e^{-\beta h}}$, while when $\mathcal{L}=-1$ and $\mathcal{U}=1$ the activation function is the hyperbolic tangent $g(h)=\tanh(\beta h)$.

The *desired* output for the μ^{th} pattern will be designated D^μ, while the *actual* output is O^μ. The *pattern energy* is defined to be the Euclidean distance between the desired and actual outputs

$$E^\mu = \sum_i (D_i^\mu - O_i^\mu)^2 \,. \tag{3}$$

The normal cost function or *energy* is the sum of all the pattern energies

$$E = \sum_\mu E^\mu \tag{4}$$

and BP simply tries to reduce E by reducing one E^μ at a time using the method of steepest descent.

3.1 Neuron Bias Removal

We first address the problem of the range of the neurons. Assuming the features to be continuous variables, one may ask if *their* range is of importance. It is a simple matter to shift and contract or expand each feature, and the difficulty of the pattern recognition problem is unchanged by such transformations. However, it has been found that a neuron's bias (ie the deviation of its average value from zero) has significant influence on the convergence time of the standard BP algorithm. Empirical results of applying BP to the XOR problem and to random problems [27] reveal that it is most advantageous to keep the average of each neuron zero. In particular the popular choice $\mathcal{L} = 0$, $\mathcal{U} = 1$ is ruled out, but the choice $\mathcal{L} = -1$, $\mathcal{U} = 1$ is ideal assuming symmetric distribution of values about their expectations. Theoretical results for a single linear ($g(h) = h$) neuron reveal that for large N the training time increases dramatically with input bias [17], due to an eigenvalue of the pattern covariance matrix which scales in proportional to N. The theoretical result applies equally well to hidden layer neurons, being inputs to the next layer. Even the output neurons are best kept unbiased, since then small values of $|g(h)|$ only occur where the derivative g' is largest and the backward error propagation is unimpeded [14].

Although our features are not always symmetrically distributed, we did not bother to determine the exact averages. Dramatic improvement in convergence time is obtained merely by using the symmetric range $[-1 \cdots +1]$ for all the neurons in the network. We can not overstress the value of this simply implemented tactic. We believe that many of the cycle number quotations found in the literature (hundreds to thousands of cycles for small toy problems) could be radically reduced by this trick alone.

3.2 Output Compensation

We next address the output neuron representation. We will concentrate on the *selector* representation defined in section 2, although the method can be easily extended to other output representations. Even with output neurons in the symmetrized range $[-1 \cdots +1]$ we have a bias problem, this time simply because most of the output units are desired to be inactive (-1) for every pattern. During training we repeatedly reinforce this tendency, since $+1$ outputs are scarce for large M. The output neurons quickly learn that they should be inactive, leading to the all-inactive output as being a preferred state (although completely illogical for a valid pattern).

To be more precise let us compare the energy of the totally inactive output state with a randomly chosen one. The desired value of output neuron i for pattern μ is D_i^μ, and for each μ D_i^μ is $+1$ for exactly one i. The totally inactive state $O_i^\mu = -1$ has pattern energy is $E^\mu = \sum_i (-1 - D_i^\mu)^2 = 4$, and thus the total energy is proportional to the number of patterns $E = \sum_\mu E^\mu = 4P$. At the beginning of training, the J_{ij} are random, and we expect the output neurons to be evenly distributed over the range $[-1 \cdots +1]$. In this case, E^μ is proportional to the number of output units M, and $E \sim MP$. When M is sufficiently large, the inactive state's energy is significantly lower, and the network will tend to it.

One way to solve this problem would be to replace the selector with an alternative output representation. We might use orthogonal vectors of ± 1, or arbitrarily chosen points (perhaps in a lower

dimensional space). This would entail saving a 'dictionary' translating these points to the desired classes - perhaps implemented as another network layer. Another way would be to somehow require at least one output neuron to be nonzero – but then invalid patterns would *always* evoke false alarms. The most direct way of dealing with the problem is to properly weight the cost of the outputs, causing the infrequent, but critical, active neurons to be more significant. We propose changing the definition of the pattern energy to

$$E^\mu = \sum_i F_i^\mu (O_i^\mu - D_i^\mu)^2 \tag{5}$$

and we take the 'compensation factor' to be

$$F_i^\mu = \frac{f-1}{2} D_i^\mu + \frac{f+1}{2} = \begin{cases} f & D_i^\mu = 1 \\ 1 & D_i^\mu = -1 \end{cases} .$$

If f is taken proportional to M, the pattern energy of the totally inactive state is proportional to M and no longer dominates random outputs.

3.3 Energy Proportional Order

In this subsection we turn to the method of presenting the patterns. We assume that the weights are updated after every pattern (pattern mode), rather than at the end of the entire cycle of all patterns (epoch mode). In this case, the order of presenting the patterns is significant. Three tactics are traditional – sequential order, random order (each new pattern is chosen at random), and random set order (a random permutation of the patterns is chosen each cycle). When all examples of a class are grouped together, sequential order is most significantly different from the other two.

All three of the above orderings blindly take patterns, without regard to how well classified the pattern may already be, or to the reduction in energy the pattern may bring. Previous authors (eg. [1, 5, 31]) have proposed various strategies :

- only updating the weights when the pattern energy exceeds a threshold,
- training on easy cases first and advancing to borderline cases (outliers) later on,
- stressing poorly classified patterns by replicating them in the pattern set or by increasing the learning rate for them,
- adding additional patterns for poorly learnt classes.

We suggest a more systematic method of taking the possible improvement into account. The maximum reduction in total energy that can be obtained while dealing with pattern μ, is its entire pattern energy E^μ, and thus selecting input patterns with probabilty proportional to E^μ updates more frequently the lesser well learnt patterns, which can contribute more. In practice, since $E = \sum_\mu E^\mu$, this selection involves randomly chosing a real number between zero and E, and determining within which pattern's 'energy interval' this number falls. As an additional ploy, one may chose to recalculate the entire energy only every C cycles, thus smoothing its fluctuations in a manner similar to the momentum term, and saving on computation time.

4 Simulations

In order to compare our algorithm with BP in a pattern classification setting, we chose an archetypal classification problem and measured the effect of our proposed improvements with respect to training time and generalization.

The input (feature) space is N-dimensional real space $\mathcal{R}^N$. We randomly chose L of the $2N$ corners of the unit side hypercube (ie. L N-dimensional vectors with elements $\pm\frac{1}{2}$) to be henceforth called 'centers'. We now define M classes, each of which is composed of the vicinities of one or more centers. The individual center vicinities are linearly separable, and thus a single hidden layer network with L hidden units should certainly suffice for classification. In the simplest case, $C = L$ and each class is

unimodal comprising those vectors closest to a hypercube corner; in more complex cases the classes are multimodal. For the simulations we chose either one or two centers per class.

The training set of P vectors was generated by adding random gaussian noise of width $\sigma = 1$ to the appropriate hypercube corners. The test set contained Q vectors similarly generated but with width σ normally taken to be very large. Since centers may be only one distance unit apart (although two randomly chosen centers are $\sqrt{N/2}$ from each other) we reject vectors which are closer to another center.

The simulations were performed with a single hidden layer network of $[-1 \cdots +1]$ neurons. The input dimension N, number of classes M, number of hidden units, and size of training set P were all varied over wide ranges. After each training cycle (defined to be exactly P updates) the 'raw classification' performance was checked on both the training and test sets. Training normally continued at least until 100% correct classification was achieved on the training set. The training was performed using random order backpropagation (BP), output compensation (OC), energy proportional order (EP), or a combination of the latter two (OCEP). The same learning rate was used for all methods and the momentum was fixed at $\alpha = 0.5$. The same initial network was used for all algorithms compared.

First let us consider the unimodal case. In all runs we noticed significant differences in the training times. When training a network with many hidden units (for example 20 to 40), in approximately half the cases, BP became caught in local minima, although OC and EP were never observed to do so. When BP *did* converge, it did so after many more cycles than OC, EP or OCEP. In a typical case, 600 100-dimensional training patterns were taken from 10 classes and the network contained 40 hidden units. BP had not converged after 200 cycles, Ep took 50 cycles, OC with $f = 10$ took 10 cycles, and OCEP took only 7 cycles. The combined algorithm OCEP is usually an order of magnitude faster than BP. Varying f we found a very broad minimum between $f = 10$ and $f = 50$.

For the bimodal case we found that the EP algorithm was consistently faster than BP. OC alone was usually even faster when M was very large, but became caught in local minima for smaller M. For example, six hundred 100-dimensional vectors were generated from 20 classes and a network with only *two* hidden units was trained. After 800 cycles BP still classified only 85 % of the training set properly, OC was similarly unable to converge, EP required 475 cycles to classify 98 %. For 1000 patterns from 200 classes, OC attained 100 % training set classification after only 12 cycles, while after 30 cycles EP still classified only 60 % properly and BP 45 %. The combined algorithm OCEP performed 100 % after only 8 cycles.

We compared the generalization abilities of the algorithms. For the unimodal case, when all of the algorithms converged, the generalization depended mainly on the number of training cycles. Thus, for a given number of cycles, although EP could be performing significantly better on the training set, the generalization would be similar. When BP or OC became caught, this was invariably caused by one or more classes being very poorly learnt while the others were classified well. Thus the generalization for these unlearnt classes would be close to zero. For the bimodal case, although the EP and OCEP algorithms required further training after 100 % training set classification in order to improve the generalization, the final results were consistently better.

Our experience with real world applications bears out the results of the simulations, although due to longer training times the conclusions could not be as systematically derived. In a typical case, 5000 patterns in a 400-dimensional space were used to train a network with about 10,000 weights. We found that for identical training times, OC would generalize better than BP by 10 %, RP by 5 % and OCEP by up to 20 %.

References

[1] Allred LG and Kelly GE, Proc. IJCNN-90, I-721 (1990) and references therein

[2] Aloni-Lavi R, Even A, Metzger Y and Stein Y, *in preparation*

[3] Ash T, *Connection Science*, **1**, 365 (1989)

[4] Becker S and Le Cun Y, U. of Toronto Connectionist Research Group, Tech. Rept. **CRG-TR-88-5** (1988); Proc. 1988 Connectionist Models Summer School (Morgan Kaufmann : 1988)

[5] Cheung RKM, Lustig I and Kornhauser AL, Proc. IJCNN-90, I-673 (1990)

[6] Grossman T, Meir R and Domany E, Complex Systems **2**, 555 (1988); Grossman T, Complex Systems **3**, 407 (1988); Grossman T, *NIPS* **2**, 516 (1989)

[7] Fahlman SE, Proc. 1988 Connectionist Models Summer School (Morgan Kaufmann : 1988)

[8] Fahlman SE and Lebiere C, Carnegie Mellon Univ. Tech. Rept. **CMU-CS-90-100** (1990)

[9] Hecht-Nielson R, IEEE 1^{st} Intl. Conf. On Neural Networks, III-11 (1987)

[10] Hornik K, *Neural Networks* **4**, 251 (1991) and references therein

[11] Hsiung JT, Suewatanakul W and Himmelblau DM, Proc. IJCNN-91, I-353 (1991)

[12] Jacobs RA, *Neural Networks* **1**, 295 (1988)

[13] Judd S, IEEE 1^{st} Intl. Conf. On Neural Networks, II-685 (1987); *Neural Network Design and the Complexity of Learning*, (MIT Press : 1990)

[14] Krzyzak A, Dai W and Suen CY, Proc. IJCNN-90, III-225 (1990)

[15] Le Cun Y, U. of Toronto Connectionist Research Group, Tech. Rept. **CRG-TR-88-6** (1988)

[16] Le Cun Y, Denker JS, Solla S, Howard RE and Jackel LD, *NIPS* **2**, 598 (1989)

[17] Le Cun Y, Kanter I and Solla SA, *NIPS* **3**, 918 (1990)

[18] Mézard M and Nadal J-P, J. Phys. **A22**, 2191 (1989); Nadal J-P, Intl. J. of Neural Systems, **1**, 55 (1989); Sirat JA and Nadal J-P, Network **1**, 423 (1990)

[19] Parker DB, IEEE 1^{st} Intl. Conf. on Neural Networks, II-593 (1987)

[20] Ruck DW, Rogers SK, Kabrisky M, Oxley ME and Suter BW, IEEE Trans. on Neural Networks, **1**, 296 (1990)

[21] Owens AJ and Filkin DL, Proc. IJCNN-89, II-381 (1989)

[22] Rumelhart DE, Hinton GE and Williams RJ, Nature, **323**, 533 (1986)

[23] Rumelhart DE, McClelland JL and the PDP Research Group, *Parallel Distributed Processing* (MIT Press : 1988)

[24] Saratchandran P, IEEE Trans. on Neural Networks **2**, 465 (1991); Proc. IJCNN-91, I-397 (1991)

[25] Shawe-Taylor JS and Cohen DA, *Neural Networks* **3**, 575 (1990)

[26] Solla SA, Levin E and Fleisher M, Complex Systems **2**, 625 (1988)

[27] Stornetta WS and Huberman BA, IEEE 1^{st} Intl. Conf. On Neural Networks, II-637 (1987)

[28] Tveter D, AI EXPERT **July 1991**, 36 (1991)

[29] Watrous RL, IEEE 1^{st} Intl. Conf. On Neural Networks, II-619 (1987)

[30] Webb AR and Lowe D, *Neural Networks* **3**, 367 (1990)

[31] Whitley D and Karnunanithi N, Proc. IJCNN-91, II-77 (1991)