Improving Generalization Performance in Character Recognition

Harris Drucker
AT&T Bell Laboratories and Monmouth College
Monmouth College, West Long Branch, NJ 07764

Yann Le Cun
AT&T Bell Laboratories
Holmdel, NJ 07733

Abstract: One test of a new training algorithm is how well the algorithm generalizes from the training data to the test data. A new training algorithm termed double backpropagation improves generalization by minimizing the change in the output due to small changes in the input. This is accomplished by minimizing the normal energy term found in backpropagation and an additional energy term that is a function of the Jacobian.

Introduction

Backpropagation [1] has been a popular supervised training algorithm for a number of years. The general procedure is to learn on a training set and see how well the generalization is on a test set. In an attempt to improve generalization performance, one can create different architectures, or for a specific architecture one can impose additional constraints. Two of the latter techniques are weights decay [2-4] and training with noise [3,5,6].

The normal energy term in backpropagation, here denoted as E_f (f for forward) is of the form $E_f = (\mathbf{D} - \mathbf{X})(\mathbf{D} - \mathbf{X})^t/2$ where t indicates transpose, and **D** and **X** are the desired and actual output row matrices (of size m corresponding to the m output components), respectively. In weight decay we add to E_f a term of the form $\alpha||\mathbf{W}||^2$, i.e., a constant times the norm squared of the weight vector. The idea here is to force the weights to be small therefore keeping the output of the nonlinear elements out of the saturated regions. Although weight decay does work, it is not always obvious in advance what the value of α should be. The rationale of adding noise to the input patterns is to move the search out of a local minimum of the weight space in addition to providing variations of the input. The problem with this approach is determining how much noise should be added thereby requiring multiple runs to determine the noise level. Generalization may be increased by picking appropriate architectures or starting with an architecture and then pruning or adding units [7-9].

1991 IEEE Workshop on Neural Networks for Signal Processing

Double Backpropagation

In double backpropagation, in addition to the normal energy term E_f we try to minimize a term of the form:

$$E_b = \frac{1}{2}\left[\frac{\partial Ef}{\partial i_1}\right]^2 + \frac{1}{2}\left[\frac{\partial Ef}{\partial i_2}\right]^2 + ... + \frac{1}{2}\left[\frac{\partial Ef}{\partial i_n}\right]^2$$

where i_j is one of n input components. The rationale for this approach is that if the input changes slightly the energy function E_f should not change. One measure of this change is just the derivative of E_f with respect to all the inputs. Therefore, by forcing E_b to be small we force the appropriate derivatives to be small.

We can show that the above equation is equivalent to $E_b = (\mathbf{D}-\mathbf{X})\mathbf{J}\mathbf{J}^t(\mathbf{D}-\mathbf{X})^t$ where J is the m by n Jacobian matrix (corresponding to the m output components and n input components. The elements of J are composed of the derivatives of each output component to every input component.

If we now add the normal energy term to this last term we obtain for the total energy term E_t:

$$E_t = (\mathbf{D}-\mathbf{X})\left[\frac{1}{2}\mathbf{I}_m + \alpha\mathbf{J}\mathbf{J}^t\right](\mathbf{D}-\mathbf{X})^t$$

where $\mathbf{I}_m$ is the identity matrix of square size m and α is a multiplicative constant which will be related to the learning rate of the neural network. The constant α is greater than one, the rationale being that near the minimum, $(\mathbf{D}-\mathbf{X})(\mathbf{D}-\mathbf{X})^t$ is close to zero and hence $\mathbf{J}\mathbf{J}^t$ will have small effect near this minimum unless α is large. In practice the optimum solution is fairly insensitive to the choice of α and no time is spent searching for the optimum α. α is related to the choice of learning constant in the neural network.

One of the key ideas in double backpropagation is that the additional energy term E_b can be calculated in one backward pass through a neural network or one forward pass through what we will call an appended network. Let us examine a simple architecture (Fig 1 - below the dashed horizontal line). The layers are fully connected (to the layer immediately above) but not all weights are shown (nor is the bias). The input layer has linear neurons while the other layers have the nonlinear transfer function. Therefore $a_j^{(r)}$, represents the summed input of neuron number j at layer r and $x_j^{(r)} = f\left[a_j^{(r)}\right]$ where f is the hyperbolic tangent.

In this case the forward energy function is :

$$E_f = \frac{1}{2}\left[d_1 - x_1^{(2)}\right]^2 + \frac{1}{2}\left[d_2 - x_2^{(2)}\right]^2 . \qquad (1)$$

Now, let us look at some of the terms obtained by backpropagating through the network. First, the derivative of the forward energy function with respect to the

input of the first neuron of the output layer:

$$-\frac{\partial E_f}{\partial a_1^{(2)}} = \left[d_1 - x_1^{(2)}\right] f'\left[a_1^{(2)}\right] . \qquad (2)$$

where f' is the derivative.

The change in weight A is now due to the term $-\frac{\partial E_f}{\partial a_1^{(2)}}$ multiplied by both $x_1^{(1)}$ and the learning constant. Similarly we can update the other weights. In normal backpropagation, these are all the gradients needed. However, we can proceed further and calculate the derivative of the forward energy function with respect to the input (recalling that the input neurons are linear):

$$-\frac{\partial E_f}{\partial i_1} = \left[E\frac{\partial E_f}{\partial a_1^{(1)}} + F\frac{\partial E_f}{\partial a_2^{(1)}}\right]$$

Terms of the form $\frac{\partial E_f}{\partial i_j}$, where j ranges over the n input components, are called the *input gradients*. It is important to note that the calculation of the input gradients is a linear operation. We now show that the input gradients can be calculated by appending a network (the top part of Figure 1) to the original network.

The appended network, which is a "mirror image" about the dashed line, uses linear neurons:

$$y_j^{(r)} = k_j^{(r)} b_j^{(r)}$$

$$k_j^{(r)} = \begin{cases} f'(a_j^{(r)}) & r > 0 \\ 1 & r = 0 \end{cases}$$

where k is the multiplicative constant whose value is related to the derivative of the input state of a neuron in the lower network. Note that the superscripts decrease as one approaches the top of the appended layer.

Although not all weights are shown consider the input state of the first neuron of the appended layer:

$$b_1^{(2)} = \left[x_1^{(2)} - d_1\right]$$

$$y_1^{(2)} = k_1^{(2)}\, b_1^{(2)}$$

$$= f'\left[a_1^{(2)}\right] \left[x_1^{(2)} - d_1\right]$$

which is equation (2). Proceeding in this fashion, we see that the output of the appended network is just the input gradient of the forward energy function.

Therefore, three steps are involved in calculating the input gradient: (1) forward propagate through the lower network (2) copy the derivatives of the input states from the lower network to the appended network as the multiplicative constants of the linear neurons and (3) forward propagate through the appended network. Once training is complete, the appended network is no longer needed and is removed.

Now we can form the backward energy function, so named because it could be obtained by backpropagating through the lower network:

$$E_b = \frac{1}{2}\left[y_1^{(0)}\right]^2 + \frac{1}{2}\left[y_2^{(0)}\right]^2 + \frac{1}{2}\left[y_3^{(0)}\right]^2$$

$$= \frac{1}{2}\left[\frac{\partial Ef}{\partial i_1}\right]^2 + \frac{1}{2}\left[\frac{\partial Ef}{\partial i_2}\right]^2 + \frac{1}{2}\left[\frac{\partial Ef}{\partial i_3}\right]^2 \qquad (3)$$

We will now minimize the sum of a constant times the backward energy function (3) and the forward energy function (1). The general idea is to backpropagate through the lower network to minimize the forward energy function and do another backpropagation starting at the top of the appended network to minimize the backward energy function (hence the description of the training algorithm as double backpropagation).

Backpropagation through the upper network has some subtleties because the weights are shared between the upper network and the lower network. First let us find the derivative of the backward energy function with respect to the state of the first neuron in the top layer recalling both that the neuron is linear and that for the top layer, the multiplicative constant is 1.

$$\frac{\partial Eb}{\partial b_1^{(0)}} = y_1^{(0)}$$

Now, the gradient with respect to the the weight F:

$$\frac{\partial E_b}{\partial F} = \frac{\partial E_b}{\partial b_1^{(0)}}\frac{\partial b_1^{(0)}}{\partial F} + \frac{\partial E_b}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial F} = \frac{\partial E_b}{\partial b_1^{(0)}} y_2^{(1)} + \frac{\partial E_b}{\partial a_2^{(1)}} i_1 .$$

The first term of the sum is found in normal backpropagation, the second is not. However, we will get the equivalent result by backpropagating through the whole network. Therefore, the algorithm proceeds as follows:

1. Present the input pattern and propagate it to the output (the top of the lower network).
2. Backpropagate the gradient of the forward energy function through the lower network. Compute the change in weights but do not change the weights yet.
3. Copy the appropriate derivatives from the lower network to the appended network.

4. Propagate forward through the upper network
5. Now backpropagate the backward energy function from the top of the appended network down through the original network, calculating the weights changes but do not change the weights yet.
6. Finally, change the weights using the weight changes calculated in steps 2 and 5.

Experimental Results

Each results presented below is the average error rate on the test set for ten runs for a particular architecture and particular training algorithm. Each training cycle was followed by one test cycle and the best test results were retained. Five learning algorithms were used:

1. Backpropagation
2. Full double backpropagation which was described above.
3. Partial double backpropagation is a modified form of the computationally expensive full double backpropagation. In this case we form a backward energy term that is a function of the gradients at the input to the hidden layer: $\frac{\partial E_f}{\partial a_j^{(1)}}$. The rationale is that small changes in the input to the hidden layer should not affect the output. In this case, the appended network required to calculate these derivatives is smaller (minus the uppermost layer in Figure 1).
4. Normal backpropagation followed by full double backpropagation. The reasoning is that backpropagation is faster and full double backpropagation following normal backpropagation should require less training cycles. In this case, full double backpropagation starts with the results of the network with the best test score found in (1) above.
5. Normal backpropagation followed by partial double backpropagation.

A database consisting of 320 training examples and 180 test samples were used on four locally constrained architectures that had been previously shown to give good results using backpropagation [10]. Twelve examples of each of the ten digits were hand drawn by a single person on a 16 by 13 bitmap using a mouse. Each image was then used to generate four examples by putting the original image in four consecutive horizontal positions on a 16 by 16 bitmap. Thus the architectures and training algorithms will be specifically tested against a database consisting strictly of translations.

The four architectures :

1. local-net: A locally connected architecture with two hidden layers (figure 2). The output of the second hidden layer is fully connected to the output.
2. local2-net: A network with two hidden layers and weight sharing (figure 2).. All units in the first hidden layer share the same weights.
3. local21-net: Same as local2-net except that the weights in going from the first to second hidden layer are also shared.
4. local4-net: Two hidden layers, the first of which consists of four 8x8 feature maps and the second four 4x4 feature maps. All the units in a feature map share the same weights, the receptive fields being of size 3x3 in going from the input to first hidden layer and of size 5x5 going from the first to second hidden layer.

In these cases of multiple hidden layers, partial double backpropagation means that the appended network consists of two layers: the mirror image (around the dashed line of Figure 1) of the output layer and the hidden layer closest to the output. The resultant error rates for the four architectures and five learning algorithms are shown in Table 1. Except for two cases of the local21 architecture double backpropagation improves performance. Our local21 architecture was never able to learn the training set in those cases where double backpropagation gives worse results. Of special interest is that fact that the local4 architecture which gives the best results using normal backpropagation has the most significant increase in performance (to 2.2%) using double backpropagation.

Our next trial was on a very large database consisting of 9709 training samples and 2007 test samples taken from handsegmented zip code data obtained from the U.S. Postal Service. The architecture is fully explained in [11,12] and consists of 4645 neurons, 2578 weights (some shared), and 98442 connections. We therefore did one run starting from the best network configuration to date. That best result using a Newton version of regular backpropagation has previously been reported as 5.03%. Using partial double backpropagation starting from these best results, the error rate was reduced to 4.68% after twenty-three iterations (approximately twelve days).

Analysis of the distribution of weights shows the reason why double backpropagation improves performance. In double backpropagation, the distribution of weights to the first hidden layer have a unimodal distribution with significantly smaller variance than that of a network trained using backpropagation. By inputting all the test patterns we can also obtain a distribution of the summed input (essentially corresponding to the $a_j^{(1)}$ of figure 1.) This distribution is bimodal with smaller variance when trained using double backpropagation. The transfer function of the sigmoid is linear over a small region centered around zero. For networks trained using double backpropagation, many more of the signals at the input to the first hidden layer are in the linear region (typically 20%) than when trained used backpropagation (where the typical

number is 5%). This type of behavior is not exhibited at the higher layers, therefore the improved performance is due to the different distribution of weights at the input to the first hidden layer.

Conclusions

Double backpropagation has been shown to be a technique that improves performance by forcing the output to be insensitive to incremental changes in the input. The improvements are especially significant for those architectures which show very good performance when trained using backpropagation. The penalty paid is an increased running time which is not too large a penalty if partial double propagation is used. Double backpropagation can be used following normal backpropagation but generally does not give as good results as double backpropagation alone. It was furthermore shown that double backpropagation creates a weight distribution at the input to the first hidden layer that has a smaller variance than that generated using backpropagation.

ARCHITECTURE	local	local21	local2	local4
BACKPROPAGATION	8.6	8.2	4.5	3.8
FULL DOUBLE BACKPROPAGATION	5.0	6.2	3.3	3.1
PARTIAL DOUBLE PROPAGATION	6.9	8.5	2.8	2.2
FULL DOUBLE BACKPROPAGATION FOLLOWS BACKPROP	5.8	6.1	3.2	2.9
PARTIAL DOUBLE BACKPROPAGATION FOLLOWS BACKPROP	6.5	9.5	3.7	2.6

Table 1. Error rate in percent for four architectures and five training algorithms. 320 items in training set and 180 in test set."

References

[1] D.E.Rumelhart, et. al., "Learning internal representations by error propagation," **Parallel Distributed Processing: Explorations in the**

Microstructure of Cognition, 1 Rumelhart and McClelland (eds.), MIT Press, Cambridge MA, 1986, pp. 318-362.

[2] G.E. Hinton, "Learning Distributed Representations of Concepts", **Proceeding of the Eight Annual Conference of the Cognitive Science Society**, pp. 1-12 (Amherst 1986), Hillsdale: Erlbaum, pp. 1- 12.

[3] R. Scaletter and A. Zee, "Emergence of Grandmother Memory in Feed Forward Networks: Learning with Noise and Forgetfulness. **Connectionist Models and Their Implications: Readings from Cognitive Science** , D. Waltz and J.A. Feldman (eds.), pp. 309-332. Norwood: Ableex, 1988.

[4] A. H. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks, **Advances in Neural Information Processing Systems I** , (Denver 1988), D.S. Touretzky (ed.), San Mateo: Morgan Kaufmann, 1989.

[5] A. von Lehman, et. al., "Factors Influencing Learning by Back Propagation, **IEEE International Conference on Neural Networks**, (San Diego 1988), vol I, pp. 335-341, New York, IEEE,

[6] J. Sietsma and R.J.F. Dow, "Neural Net Pruning--Why and How", **IEEE International Conference on Neural Networks** (San Diego 1988), vol I, pp. 325 - 333, New York: IEEE.

[7] S.E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture, **Advances in Neural Information Processing Systems II**, D.S. Touretzky (ed.), 1990, pp. 524-532, San Mateo: Morgan Kaufmann.

[8] M. Mezard and J.-P. Nadal, "Learning in Feedforward Layered Networks: The Tiling Algorithm, **Journal of Physics A 22**,(1989) pp. 2191-2204.

[9] S.I. Gallant, "Optimal Linear Discriminants, **Eighth International Conference on Pattern Recognition**, (Paris 1986) pp. 849-852 New York: IEEE.

[10] Y. LeCun, "Generalization and Network Design Strategies, **Connectionism in Perspective**, Pfeifer, et. al. (eds.), 19 Zurich, Switzerland: Elsevier

[11] Y. Le Cun, et. al., "Backpropagation Applied to Handwritten Zip Code Recognition, **Neural Computation 1**, (1989), pp. 541-551.

[12] Y. Le Cun, et. al., "Handwritten Digit Recognition with a Back-Propagation Network", **Advances in Neural Information Processing Systems**, (Denver - 1989) D.S. Touretzky (ed.), pp. 396-404, San Mateo: Morgan Kaufmann.

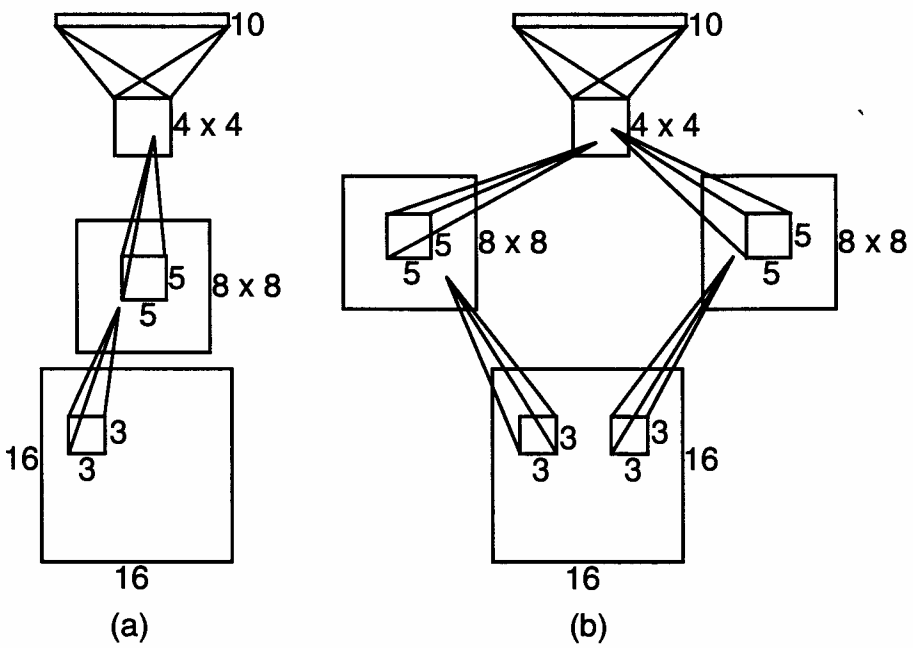

Fig. 2. (a) local architecture
(b) local21 and local2 architecture

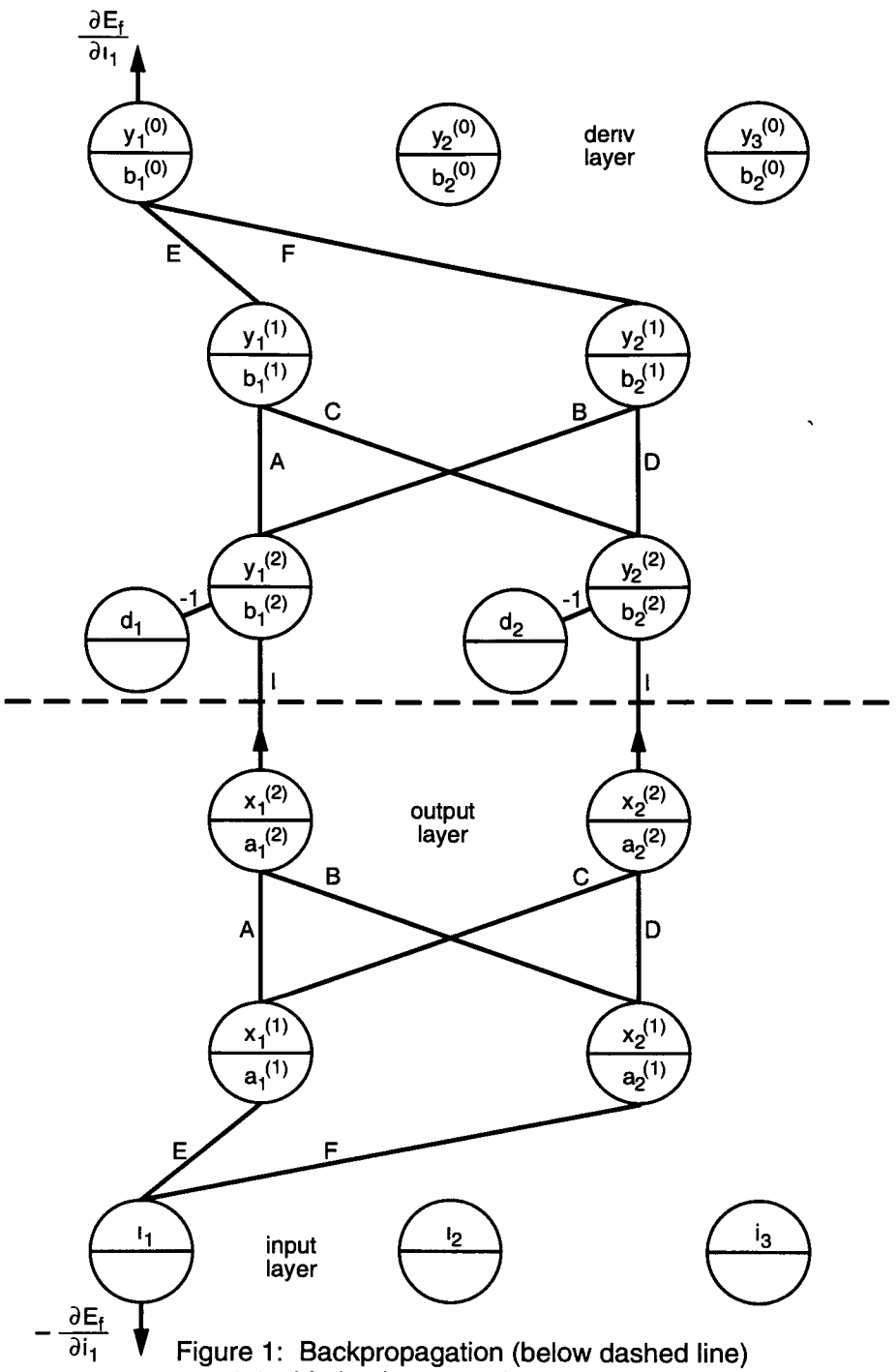

Figure 1: Backpropagation (below dashed line) and double backpropagation (entire network)