Generalization and Network Design Strategies

Y. le Cun
Department of Computer Science
University of Toronto

Technical Report CRG-TR-89-4
June 1989

Send requests to:

The CRG technical report secretary
Department of Computer Science
University of Toronto
10 Kings College Road
Toronto M5S 1A4
CANADA

INTERNET: carol@ai.toronto.edu
UUCP: uunet!utai!carol
BITNET: carol@utorgpu

This work has been supported by a grant from the Fyssen foundation, and a grant from the Sloan foundation to Geoffrey Hinton. The author wishes to thank Geoff Hinton, Mike Mozer, Sue Becker and Steve Nowlan for helpful discussions, and John Denker and Larry Jackel for useful comments. The Neural Network simulator SN is the result of a collaboration between Leon-Yves Bottou and the author. Y. le Cun's present address is Room 4G-332, AT&T Bell Laboratories, Crawfords Corner Rd, Holmdel, NJ 07733.

Y. Le Cun. Generalization and network design strategies. Technical Report CRG-TR-89-4, University of Toronto Connectionist Research Group, June 1989. a shorter version was published in Pfeifer, Schreter, Fogelman and Steels (eds) 'Connectionism in perspective', Elsevier 1989.

Generalization and Network Design Strategies

Yann le Cun *
Department of Computer Science, University of Toronto
Toronto, Ontario, M5S 1A4. CANADA.

Abstract

An interesting property of connectionist systems is their ability to learn from examples. Although most recent work in the field concentrates on reducing learning times, the most important feature of a learning machine is its generalization performance. It is usually accepted that good generalization performance on real-world problems cannot be achieved unless some *a priori* knowledge about the task is built into the system. Back-propagation networks provide a way of specifying such knowledge by imposing constraints both on the architecture of the network and on its weights. In general, such constraints can be considered as particular transformations of the parameter space

Building a constrained network for image recognition appears to be a feasible task. We describe a small handwritten digit recognition problem and show that, even though the problem is linearly separable, single layer networks exhibit poor generalization performance. Multilayer constrained networks perform very well on this task when organized in a hierarchical structure with shift invariant feature detectors.

These results confirm the idea that minimizing the number of free parameters in the network enhances generalization.

1 Introduction

Connectionist architectures have drawn considerable attention in recent years because of their interesting learning abilities Among the numerous learning algorithms that have been proposed for complex connectionist networks,

*Present address: Room 4G-332, AT&T Bell Laboratories, Crawfords Corner Rd, Holmdel, NJ 07733.

Back-Propagation (BP) is probably the most widespread. BP was proposed in (Rumelhart et al , 1986), but had been developed before by several independent groups in different contexts and for different purposes (Bryson and Ho, 1969, Werbos, 1974, le Cun, 1985; Parker, 1985; le Cun, 1986) Reference (Bryson and Ho, 1969) was in the framework of optimal control and system identification, and one could argue that the basic idea behind BP had been used in optimal control long before its application to machine learning was considered (le Cun, 1988)

Two performance measures should be considered when testing a learning algorithm learning speed and generalization performance Generalization is the main property that should be sought, it determines the amount of data needed to train the system such that a correct response is produced when presented a patterns outside of the training set. We will see that learning speed and generalization are closely related.

Although various successful applications of BP have been described in the literature, the conditions in which good generalization performance can be obtained are not understood. Considering BP as a general learning rule that can be used as a black box for a wide variety of problems is, of course, wishful thinking Although some moderate sized problems can be solved using unstructured networks, we cannot expect an unstructured network to generalize correctly on every problem. The main point of this paper is to show that good generalization performance *can* be obtained if some *a priori* knowledge about the task is built into the network. Although in the general case specifying such knowledge may be difficult, it appears feasible on some highly regular tasks such as image and speech recognition.

Tailoring the network architecture to the task can be thought of as a way of reducing the size of the space of possible functions that the network can generate, without overly reducing its computational power Theoretical studies (Denker et al , 1987) (Patarnello and Carnevali, 1987) have shown that the likelihood of correct generalization depends on the size of the hypothesis space (total number of networks being considered), the size of the solution space (set of networks that give good generalization), and the number of training examples If the hypothesis space is too large and/or the number of traning examples is too small, then there will be a vast number of networks which are consistent with the training data, only a small proportion of which will lie in the true solution space, so poor generalization is to be expected Conversely, if good generalization is required, when the generality of the architecture is increased, the number of training examples must also be increased. Specifically, the required number of examples scales like the logarithm of the number of functions that the network architecture can implement

An illuminating analogy can be drawn between BP learning and curve fitting. When using a curve model (say a polynomial) with lots of parameters compared to the number of points, the fitted curve will closely model the training data but will not be likely to accurately represent new data. On the other hand, if the number of parameters in the model is small, the model will not necessarily represent the training data but will be more likely to capture the regularity of the data and extrapolate (or interpolate) correctly. When the data is not too noisy, the optimal choice is the minimum size model that represents the data.

A common-sense rule inspired by this analogy tells us to minimize the number of free parameters in the network to increase the likelihood of correct generalization But this must be done without reducing the size of the network to the point where it can no longer compute the desired function. A good compromise becomes possible when some knowledge about the task is available, but the price to pay is an increased effort in the design of the architecture.

2 Weight Space Transformation

Reducing the number of free parameters in a network does not necessarily imply reducing the size of the network Such techniques as weight sharing, described in (Rumelhart et al , 1986) for the so-called T-C problem, can be used to reduce the number of free parameters while preserving the size of the network and specifying some symmetries that the problem may have.

In fact, three main techniques can be used to build a reduced size network.

The first technique is problem-independent and consists in dynamically deleting "useless" connections during training This can be done by adding a term in the cost function that penalizes big networks with many parameters Several authors have described such schemes, usually implemented as a non-proportional weight decay (Rumelhart, personnal communication 1988), (Chauvin, 1989, Hanson and Pratt, 1989), or using "gating coefficients" (Mozer and Smolensky, 1989) Generalization performance has been reported to increase significantly on small problems Two drawbacks of this technique are that it requires a fine tuning of the "pruning" coefficient to avoid catastrophic effects, and also that the convergence is significantly slowed down

2.1 Weight Sharing

The second technique is weight sharing Weight sharing consists in having several connections (links) be controlled by a single parameter (weight) Weight sharing can be interpreted as imposing equality constraints among the connection strengths. An interesting feature of weight sharing is that it can be

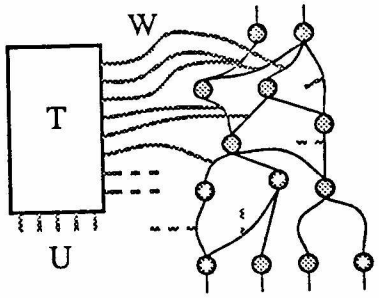

Figure 1. Weight Space Transformation.

implemented with very little computational overhead. Weight sharing is a very general paradigm that can be used to describe so-called Time Delay Neural Networks used for speech recognition (Waibel et al., 1988, Bottou, 1988), time-unfolded recurrent networks, or shift-invariant feature extractors. The experimental results presented in this paper make extensive use of weight sharing.

2.2 General Weight Space Transformations

The third technique, which really is a generalization of weight sharing, is called weight-space transformation (WST) (le Cun, 1988) WST is based on the fact that the search performed by the learning procedure need not be done in the space of connection strengths, but can be done in any parameter space that is suitable for the task This can be achieved provided that the connections strengths can be computed from the parameters through a given transformation, and provided that the Jacobian matrix of this transformation is known, so that we are able to compute the partials of the cost function with respect to the parameters The gradient of the cost function with respect to the parameters is then just the product of the Jacobian matrix of the transformation by the gradient with respect to the connection strengths. The situation is depicted on figure 1

2.2.1 WST to improve learning speed

Several types of WST can be defined, not only for reducing the size of the parameter space, but also for speeding up the learning

Although the following example is quite difficult to implement in practice, it gives an idea about how WST can accelerate learning. Let us assume that the cost function C minimized by the learning procedure is purely quadratic w r.t the connection strengths W In other words, C is of the form

$$C(W) = \frac{1}{2}W^T HW + J^T W + K$$

where W is the vector of connection strengths, H the Hessian matrix (the matrix of second derivatives) which will be assumed positive definite Then the surfaces of equal cost are hyperparaboloids centered around the optimal solution Performing steepest descent in this space will be inefficient if the eigenvalues of H have wide variations In this case the paraboloids of equal cost are very elongated forming a steep ravine The learning time is known to depend heavily on the ratio of the largest to the smallest eigenvalue. The larger this ratio, the more elongated the paraboloids, and the slower the convergence. Let us denote Λ the diagonalized version of H, and Q the unitary matrix formed by the (orthonormal) eigenvectors of H, we have $H = Q^T \Lambda Q$. Now, let Σ be the diagonal matrix whose elements are the square root of the elements of Λ, then H can be rewritten as $H = Q^T \Sigma \Sigma Q$ We can now rewrite the expression for $C(W)$ in the following way

$$C(W) = \frac{1}{2}W^T Q^T \Sigma \Sigma Q W + (J')^T \Sigma Q W + K$$

Using the notation $U = \Sigma Q W$ we obtain

$$C(U) = \frac{1}{2}U^T U + (J')^T U + K$$

In the space of U, the steepest descent search will be trivial since the Hessian matrix is equal to the identity and the surfaces of equal cost are hyper-spheres The steepest descent direction points in the direction of the solution and *is* the shortest path to the solution Perfect learning can be achieved in one single iteration if Q and Σ are known accurately. The transformation for obtaining the connection strengths W from the parameters U is simply

$$W = Q^T \Sigma^{-1} U$$

During learning, the path followed by U in U space is a straight line, as well as the path followed by W in W space This algorithm is known as Newton's algorithm, but is usually expressed directly in W space. Performing steepest descent in U space is equivalent to using Newton's algorithm in W space.

Of course in practice this kind of WST is unrealistic since the size of the Hessian matrix is huge (number of connections squared), and since it is quite

expensive to estimate and diagonalize Moreover, the cost function is usually *not* quadratic in connection space, which may cause the Hessian matrix to be non positive, non definite, and may cause it to vary with W Nevertheless, some approximations can be made which make these ideas implementable (le Cun, 1987, Becker and le Cun, 1988)

2.2.2 WST and generalization

The WST just described is an example of *problem-independent* WST, Other kinds of WST which are *problem-dependent* can be devised. Building such transformation requires a fair amount of knowledge about the problem as well as a reasonable guess about what an optimal network solution for this problem could be Finding WST that improve generalization usually amounts to reducing the size of the parameter space. In the following sections we describe an example where simple WST such as weight sharing have been used to improve generalization

3 An example: A Small Digit Recognition Problem

The following experimental results are presented to illustrate the strategies that can be used to design a network for a particular problem The problem described here is in no way a real world application but is sufficient for our purpose The intermediate size of the database makes the problem non-trivial, but also allows for extensive tests of learning speed and generalization performance

3.1 Description of the Problem

The database is composed of 480 examples of numerals represented as 16 pixels by 16 pixels binary images 12 example of each of the 10 digits were handdrawn by a single person on a 16 by 13 bitmap using a mouse Each image was then used to generate 4 examples by putting the original image in 4 consecutive horizontal positions on a 16 by 16 bitmap The training set was then formed by choosing 32 examples of each class at random among the complete set of 480 images the remaining 16 examples of each class were used as the test set Thus, the training set contained 320 images, and the test set contained 160 images On figure 2 are represented some of the training examples

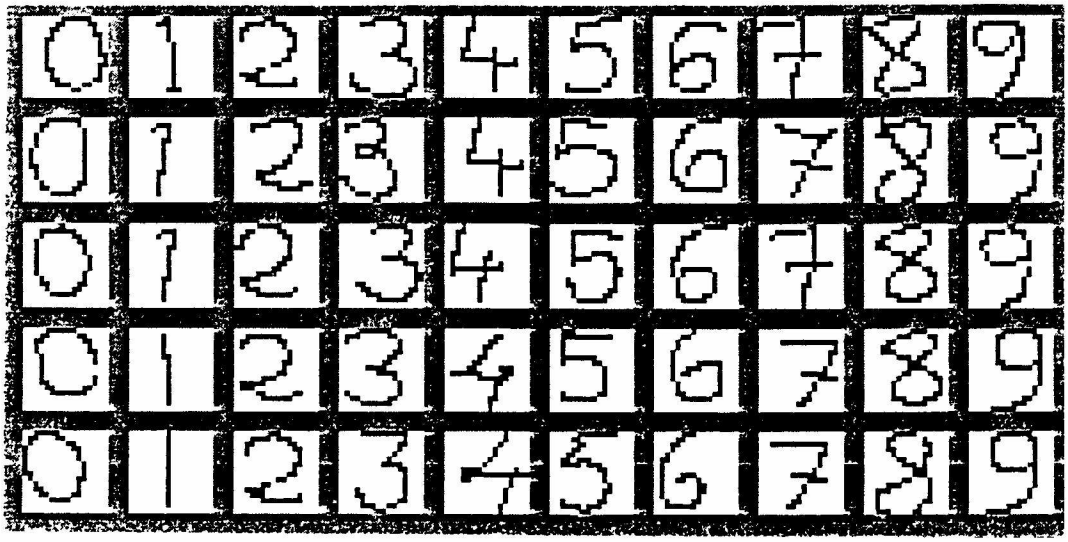

Figure 2. Some examples of input patterns.

3.2 Experimental Setup

All simulations were performed using the BP simulator SN (Bottou and le Cun, 1988)

Each unit in the network computes a dot product between its input vector and its weight vector This weighted sum, denoted a_i for unit i, is then passed through a sigmoid squashing function to produce the state of unit i, denoted by x_i

$$x_i = f(a_i)$$

The squashing function is a scaled hyperbolic tangent:

$$f(a) = A \tanh Sa$$

where A is the amplitude of the function and S determines its slope at the origin, and f is an odd function, with horizontal asymptotes $+A$ and $-A$

Symmetric functions are believed to yield faster convergence, although the learning can become extremely slow if the weights are too small The cause of this problem is that the origin of weight space is a stable point for the learning dynamics, and, although it is a saddle point, it is attractive in almost all directions For our simulations, we use $A = 1.7159$ and $S = \frac{2}{3}$. with this choice of parameters, the equalities $f(1) = 1$ and $f(-1) = -1$ are satisfied The rationale behind this is that the overall gain of the squashing transformation is around 1 in normal operating conditions, and the interpretation of the state of the network is simplified Moreover, the absolute value of the second derivative

of f is a maximum at $+1$ and -1, which improves the convergence at the end of the learning session.

Before training, the weights are initialized with random values using a uniform distribution between $-2.4/F_i$ and $2.4/F_i$ where F_i is the number of inputs (fan-in) of the unit which the connection belongs to [1]. The reason for dividing by the fan-in is that we would like the initial standard deviation of the weighted sums to be in the same range for each unit, and to fall within the normal operating region of the sigmoid If the initial weights are too small, the gradients are very small and the learning is slow, if they are too large, the sigmoids are saturated and the gradient is also very small The standard deviation of the weighted sum scales like the square root of the number of inputs when the inputs are independent, and it scales linearly with the number of inputs if the inputs are highly correlated. We chose to assume the second hypothesis since some units receive highly correlated signals

The output cost function is the usual mean squared error·

$$C = \frac{1}{P}\sum_p \sum_o \frac{1}{2}(D_{op} - X_{op})^2$$

where P is the number of patterns, D_{op} is the desired state for output unit o when pattern p is presented on the input. X_{op} is the state of output unit o when pattern p is presented. It is worth pointing out that the target values for the output units are well within the range of the sigmoid This prevents the weights from growing indefinitely and prevents the output units from operating in the flat spot of the sigmoid. Additionally, since the second derivative of the sigmoid is maximum near the target values, the curvature of the error function around the solution is maximized and the convergence speed during the final phase of the learning process is improved

During each learning experiment, the patterns were presented in a constant order, and the training set was repeated 30 times. The weights were updated after each presentation of a single pattern according to the so-called stochastic gradient or "on-line" procedure Each learning experiment was performed 10 times with different initial conditions All experiments were done both using standard gradient descent and a special version of Newton's algorithm that uses a positive, diagonal approximation of the Hessian matrix (le Cun, 1987, Becker and le Cun, 1988)

All experiments were done using a special version of Newton's algorithm that uses a positive, diagonal approximation of the Hessian matrix (le Cun, 1987, Becker and le Cun, 1988) This algorithm is not believed to bring a tremendous

[1] since several connections share a weight this rule could be difficult to apply, but in our case, all connections sharing a same weight belong to units with identical fan-ins

increase in learning speed but it converges reliably without requiring extensive adjustments of the learning parameters

At each learning iteration a particular weight u_k (that can control several connection strengths) is updated according to the following rule

$$u_k \leftarrow u_k + \epsilon_k \sum_{(i,j)\in V_k} \frac{\partial C}{\partial w_{ij}}$$

where C is the cost function, w_{ij} is the connection strength from unit j to unit i, V_k is the set of unit index pairs (i, j) such that the connection strength w_{ij} is controlled by the weight u_k. The step size ϵ_k is not constant but is function of the curvature of the cost function along the axis u_k. The expression for ϵ_k is:

$$\epsilon_k = \frac{\lambda}{\mu + h_{kk}}$$

where λ and μ are constant and h_{kk} is a running estimate of the second derivative of the cost function C with respect to u_k. The terms h_{kk} are the diagonal terms of the Hessian matrix of C with respect to the parameters u_k. The larger h_{kk}, the smaller the weight update The parameter μ prevents the step size from becoming too large when the second derivative is small, very much like the "model-trust" methods used in non-linear optimization. Special actions must taken when the second derivative is negative to prevent the weight vector from going uphill Each h_{kk} is updated according to the following rule:

$$h_{kk} \leftarrow (1-\gamma)h_{kk} + \gamma \sum_{(i,j)\in V_k} \frac{\partial^2 C}{\partial w_{ij}^2}$$

where γ is a small constant which controls the length of the window on which the average is taken The term $\partial^2 C/\partial w_{ij}^2$ is given by:

$$\frac{\partial^2 C}{\partial w_{ij}^2} = \frac{\partial^2 C}{\partial a_i^2} x_j^2$$

where x_j is the state of unit j and $\partial^2 C/\partial a_i^2$ is the second derivative of the cost function with respect to the total input to unit i (denoted a_i) These second derivatives are computed by a back-propagation procedure similar to the one used for the first derivatives (le Cun, 1987):

$$\frac{\partial^2 C}{\partial a_i^2} = f'(a_i)^2 \sum_k w_{ki}^2 \frac{\partial^2 C}{\partial a_k^2} - f''(a_i)\frac{\partial C}{\partial x_i}$$

The first term on the right hand side of the equation is always positive, while the second term, involving the second derivative of the squashing function f,

can be negative. For the simulations, we used an approximation to the above expression that gives positive estimates by simply neglecting the second term:

$$\frac{\partial^2 C}{\partial a_i^2} = f'(a_i)^2 \sum_k w_{ki}^2 \frac{\partial^2 C}{\partial a_k^2}$$

This corresponds to the well-known Levenberg-Marquardt approximation used for non-linear regression (see for example (Press et al., 1988)).

This procedure has several interesting advantages over standard non-linear optimization techniques such as BFGS or conjugate gradient. First, it can be used in conjunction with the stochastic update (after each pattern presentation) since a line search is not required. Second, it makes use of the *analytical* expression of the diagonal Hessian, standard quasi-Newton methods *estimate* the second order properties of the error surface Third, the scaling laws are much better than with the BFGS method that requires to store an estimate of the full Hessian matrix [2]

In this paper, we only report the results obtained through this pseudo-Newton algorithm since they were consistently better than the one obtained through standard gradient descent The input layer of all networks were 16 by 16 binary images, and their output layer was composed of 10 units, one per class. An output configuration was considered correct if the most-activated unit corresponded to the correct class.

In the following, when talking about layered networks, we will refer to the number of layers of modifiable weights. Thus, a network with one hidden layer is referred to as a two-layer network

3.3 Net-1: A Single Layer Network

The simplest network that can be tested on this problem is a single layer, fully connected network with 10 sigmoid output units (2570 weights including the biases) Such a network has successfully learned the training set, which means that the problem is linearly separable But, even though the training set can be learned perfectly, the generalization performance is disappointing. between 80% and 72% depending on when the learning is stopped (see curve 1 on figure 3). Interestingly, the performance on the test set reaches a maximum quite early during training and goes down afterwards This over-training phenomenon has been reported by many authors The analysis of this phenomenon is outside the scope of this paper. When observing the weight vectors of the output units, it becomes obvious that the network can do nothing but develop a set of matched filters tuned to match an "average pattern" formed by superimposing

[2] Recent developments such as Nocedal's "limited storage BFGS" may alleviate this problem

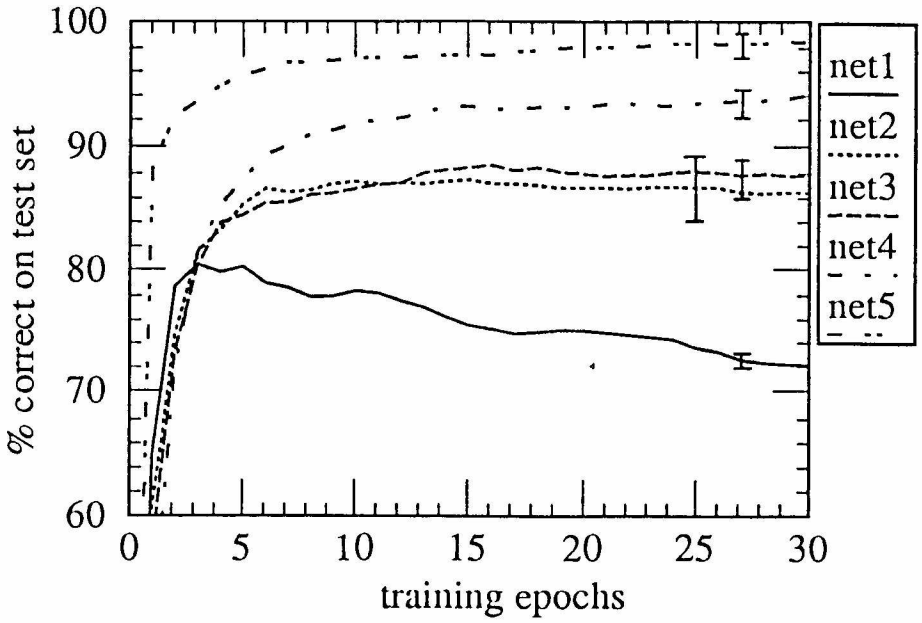

Figure 3 Generalization performance vs training time for 5 network architectures Net-1 single layer, Net-2: 12 hidden units fully connected, Net-3. 2 hidden layers locally connected, Net-4· 2 hidden layers, locally connected with constraints, Net-5· 2 hidden layers, local connections, two levels of constraints

all the training examples Despite its relatively large number of parameters, such a system cannot possibly generalize correctly except in trivial situations, and certainly not when the input patterns are slightly translated. The classification is essentially based on the computation of a weighted overlap between the input pattern and the "average prototype"

3.4 Net-2: A Two-Layer, Fully Connected Network

The second step is to insert a hidden layer between the input and the output. The network has 12 hidden units, fully connected both to the input and the output There is a total of 3240 weights including the biases. Predictably, this network can also learn perfectly the training set in a few epochs [3] (between 7 and 15) The generalization performance is better than with the previous

[3]The word epoch is used to designate an entire pass through the training set, which in our case is equivalent to 320 pattern presentations

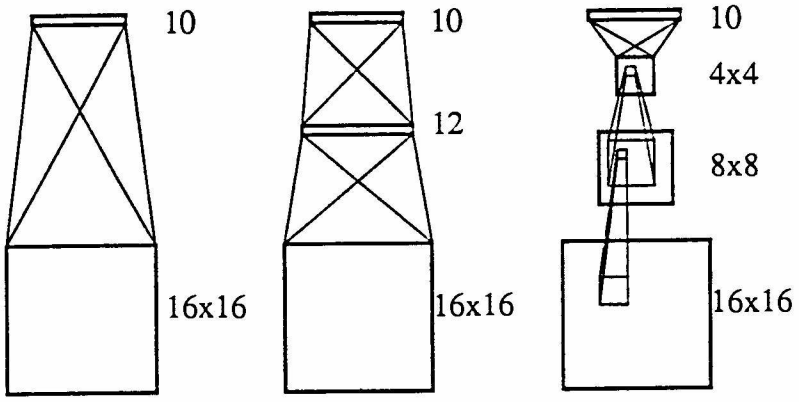

Figure 4: three network architectures Net-1, Net-2 and Net-3

network and reaches 87% after only 6 epochs (see figure 3). A very slight over-learning effect is also observed, but its amplitude is much smaller than with the previous network. It is interesting to note that the standard deviation on the generalization performance is significantly larger than with the first network. This an indication that the network is largely underdetermined, and the number of solutions that are consistent with the training set is large. Unfortunately, these various solutions do not give equivalent results on the test set, thereby explaining the large variations in generalization performance.

From this result, it is quite clear that this network is too big (or has too many degrees of freedom)

3.5 Net-3: A Locally Connected, 3-Layer Network

Since reducing the size of the network will also reduce its generality, some knowledge about the task will be necessary in order to preserve the network's ability to solve the problem A simple solution to our over-parameterization problem can be found if we remember that the network should recognize images. Classical work in visual pattern recognition have demonstrated the advantage of extracting local features and combining them to form higher order features We can easily build this knowledge into the network by forcing the hidden units to only combine local sources of information. The architecture comprises two hidden layers named H1 and H2. The first hidden layer, H1, is a 2-dimensional array of size 8 by 8. Each unit in H1 takes its inputs from 9 units on the input plane situated in a 3 by 3 square neighborhood. For units in layer H1 that are

one unit apart, their receptive fields (in the input layer) are two pixels apart Thus, the receptive fields of two neighbouring hidden units overlap by one row or one column. Because of this two-to-one undersampling in each direction, the information is compacted by a factor of 4 going from the input to H1.

Layer H2 is a 4 by 4 plane, thus, a similar two-to-one undersampling occurs going from layer H1 to H2, but the receptive fields are now 5 by 5. H2 is fully connected to the 10 output units. The network has 1226 connections (see figure 4)

The performance is slightly better than with Net-2: 88.5%, but is obtained at a considerably lower computational cost since Net-3 is almost 3 times smaller than Net-2. Also note that the standard deviation on the performance of Net-3 is smaller than for Net-2. This is thought to mean that the hypothesis space for Net-3 (the space of possible functions it can implement) is much smaller than for Net-2

3.6 Net-4: A Constrained Network

One of the major problems of image recognition, even as simple as the one we consider in this work, is that distinctive features of an object can appear at various locations on the input image. Therefore it seems useful to have a set feature detectors that can detect a particular instance of a feature anywhere on the input plane. Since the *precise* location of a feature is not relevant to the classification, we can afford to loose some position information in the process. Nevertheless, an *approximate* position information must be preserved in order to allow for the next levels to detect higher order features.

Detection of feature at any location on the input can be easily done using weight sharing. The first hidden layer can be composed of several planes that we will call *feature maps* All units in a plane share the same set of weights, thereby detecting the same feature at different locations. Since the exact position of the feature is not important, the feature maps need not be as large as the input An interesting side effect of this technique is that it reduces the number of free weights in the network by a large amount.

The architecture of Net-4 is very similar to Net-3 and also has two hidden layers The first hidden layer is composed of two 8 by 8 feature maps. Each unit in a feature map takes input on a 3 by 3 neighborhood on the input plane For units in a feature map that are one unit apart, their receptive fields in the input layer are two pixels apart. Thus, as with Net-3 the input image is *undersampled* The main difference with Net-3 is that all units in a feature map share the same set of 9 weights (but each of them has an independent bias) The undersampling technique serves two purposes The first is to keep

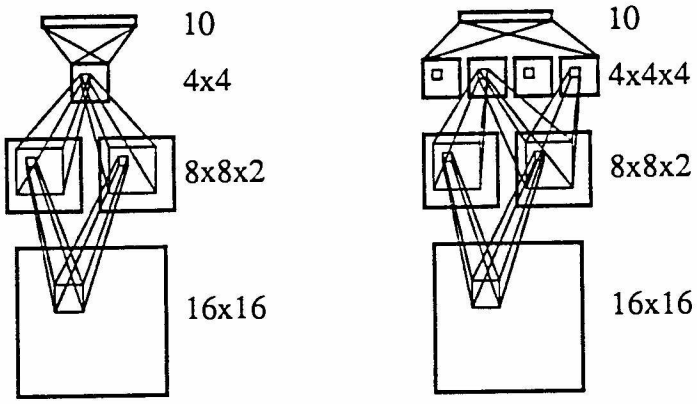

Figure 5 two network architectures with shared weights: Net-4 and Net-5

the size of the network within reasonable limits The second is to ensure that some location information is discarded during the feature detection

Even though the feature detectors are shift invariant, the operation they collectively perform is not. When the input image is shifted, the output of the feature maps is also shifted, but is otherwise left almost unchanged. Because of the two-to-one undersampling, when the shift of the input is small, the output of the feature maps is *not* shifted, but merely slightly distorted.

As in the previous network, the second hidden layer is a 4 by 4 plane with 5 by 5 local receptive fields and no weight sharing. The output is fully connected to the second hidden layer and has, of course, 10 units. The network has 2266 connections but only 1132 (free) weights (see figure 5)

The generalization performance of this network jumps to 94%, indicating that built-in shift invariant features are quite useful for this task. This result also indicates that, despite the very small number of independent weights, the computational power of the network is increased.

3.7 Net-5: A Network with Hierarchical Feature Extractors

The same idea can be pushed further, leading to a hierarchical structure with several levels of constrained feature maps

The architecture of Net-5 is very similar to the one of Net-4, except that the second hidden layer H2 has been replaced by four feature maps each of which is a 4 by 4 plane. Units in these feature maps have 5 by 5 receptive fields in the

first hidden layer. Again, all units in a feature map share the same set of 25 weights and have independent biases. And again, the two-to-one undersampling occurs between the first and the second hidden layer.

The network has 5194 connection but only 1060 free parameters, the smallest number of all networks described in this paper (see figure 5).

The generalization performance is 98.4% (100% generalization was obtained during two of the ten runs) and increases extremely quickly at the beginning of learning. This suggests that using several levels of constrained feature maps is a big help for shift invariance.

4 Discussion

The results are summarized on table 1.

As expected, the generalization performance goes up as the number of free parameters in the network goes down and as the amount of built-in knowledge goes up. A noticeable exception to this rule is the result given by the single-layer network and the two-layer, fully connected network. Even though the two layer net has more parameters, the generalization performance is significantly better One explanation could be that the one-layer network cannot classify the whole set (training plus testing) correctly, but experiments show that it can. We see two other possible explanations. The first one is that some knowledge is implicitly put by inserting a hidden layer: we tell the system that the problem is not first order. the second one is that the efficiency of the learning procedure (as defined in (Denker et al., 1987)) is better with a two layer net than with a one layer net, meaning that more information is extracted from each example with the former. This is highly speculative and should be investigated further.

4.1 Tradeoff Between Speed, Generality and Generalization

Computer scientists know that storage space, computation time and generality of the code can be exchanged when designing a program to solve a particular problem For example, a program that computes a trigonometric function can use a series expansion, or a lookup table. the latter uses more memory than the former but is faster. Using properties of trigonometric functions, the same code (or table) can be used to compute several functions, but usually results in some loss in efficiency.

The same kind of exchange exists for learning machines. It is trivial to design a machine that learns very quickly, does not generalize, and requires an enormous amount of hardware In fact this learning machine has already

network architecture	links	weights	performance
single layer network	2570	2570	80 %
two layer network	3240	3240	87 %
locally connected	1226	1226	88.5 %
constrained network	2266	1132	94 %
constrained network 2	5194	1060	98.4 %

Table 1 Generalization performance for 5 network architectures. Net-1. single layer; Net-2: 12 hidden units fully connected; Net-3 2 hidden layers locally connected; Net-4: 2 hidden layers, locally connected with constraints; Net-5· 2 hidden layers, local connections, two levels of constraints. Performance on training set is 100% for all networks

been built and is called a Random Access Memory On the other hand, a back-propagation network [4] takes longer to train but is expected to generalize. Unfortunately, as shown in (Denker et al., 1987), generalization can be obtained only at the price of generality

4.2 On-Line Update vs Batch Update

All simulations described in this paper were performed using the so-called "on-line" or "stochastic" version of back-propagation where the weights are updated after each pattern, as opposed to the "batch" version where the weights are updated after the gradients have been accumulated over the whole training set. Experiment show that stochastic update is far superior to batch update when there is some redundancy in the data. In fact stochastic update *must* be better when a certain level of generalization is expected. Let us take an example where the training database is composed of two copies of the same subset. Then accumulating the gradient over the whole set would cause redundant computations to be performed Stochastic gradient does not have this problem. This idea can be generalized to training sets where there exist no precise repetition of the same pattern but where some redundancy is present.

4.3 Conclusion

We showed an example where constraining the network architecture improves both learning speed and generalization performance dramatically This is re-

[4] unless it is designed to emulate a RAM

ally not surprising but it is more easily said than done. However, we have demonstrated that it *can* be done in at least one case, image recognition, using a hierarchy of shift invariant local feature detectors. These techniques can be easily extended (and have been) to other domains such as speech recognition.

Complex software tools with advanced user interfaces for network description and simulation control are required in order to solve a real application. Several network structures must be tried before an acceptable one is found and a quick feedback on the peformance is critical.

We are just beginning to collect the tools and understand the principles which can help us to *design* a network for a particular task. Designing a network for a real problem will require a significant amount of engineering, which the availability of powerful learning algorithms will hopefully keep to a bare minimum

Acknowledgments

This work has been supported by a grant from the Fyssen foundation, and a grant from the Sloan foundation to Geoffrey Hinton. The author wishes to thank Geoff Hinton, Mike Mozer, Sue Becker and Steve Nowlan for helpful discussions, and John Denker and Larry Jackel for useful comments. The Neural Network simulator SN is the result of a collaboration between Léon-Yves Bottou and the author.

References

Becker, S and le Cun, Y (1988). Improving the convergence of back-propagation learning with second-order methods. Technical Report CRG-TR-88-5, University of Toronto Connectionist Research Group.

Bottou, L -Y (1988) Master's thesis, EHEI, Universite de Paris 5.

Bottou, L.-Y and le Cun, Y. (1988). Sn: A simulator for connectionist models. In *Proceedings of NeuroNimes 88*, Nimes, France.

Bryson, A and Ho, Y (1969). *Applied Optimal Control* Blaisdell Publishing Co

Chauvin, Y (1989) A back-propagation algorithm with optimal use of hidden units In Touretzky, D., editor, *Advances in Neural Information Processing Systems* Morgan Kaufmann.

Denker, J., Schwartz, D., Wittner, B , Solla, S. A., Howard, R., Jackel, L., and Hopfield, J. (1987) Large automatic learning, rule extraction and generalization. *Complex Systems*, 1:877–922.

Hanson, S. J. and Pratt, L. Y (1989). Some comparisons of constraints for minimal network construction with back-propagation. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*. Morgan Kaufmann.

le Cun, Y (1985). A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France

le Cun, Y (1986). Learning processes in an asymmetric threshold network. In Bienenstock, E., Fogelman-Soulié, F., and Weisbuch, G., editors, *Disordered systems and biological organization*, pages 233–240, Les Houches, France Springer-Verlag

le Cun, Y. (1987). *Modèles Connexionnistes de l'Apprentissage*. PhD thesis, Université Pierre et Marie Curie, Paris, France

le Cun, Y. (1988). A theoretical framework for back-propagation. In Touretzky, D , Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, CMU, Pittsburgh, Pa. Morgan Kaufmann.

Mozer, M. C and Smolensky, P (1989) Skeletonization: A technique for trimming the fat from a network via relevance assessment. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*. Morgan Kaufmann.

Parker, D B. (1985). Learning-logic. Technical report, TR-47, Sloan School of Management, MIT, Cambridge, Mass.

Patarnello, S. and Carnevali, P (1987). Learning networks of neurons with boolean logic *Europhysics Letters*, 4(4).503–508

Press, W H , Flannery, B. P., A., T. S., and T., V. W. (1988) *Numerical Recipes* Cambridge University Press, Cambridge.

Rumelhart, D. E., Hinton, G E., and Williams, R. J. (1986) Learning internal representations by error propagation. In *Parallel distributed processing. Explorations in the microstructure of cognition*, volume I Bradford Books, Cambridge, MA.

Waibel, A , Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1988). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing.*

Werbos, P. (1974). *Beyond Regression.* Phd thesis, Harvard University.