Chapter 3

Neural Network Approaches to Generalization and Invariance

In Chapter 2 several general methods for performing invariant pattern recognition were described. In this chapter, we move from the general to the specific, and consider in particular the application of *artificial neural networks* to the problem of invariant pattern recognition. This will involve a clarification of the terms "generalization" and "invariance"; both are frequently used in the neural networks literature, often as if they described the same phenomenon: this is not the case.

From its earliest history, practitioners in the field of neural networks have been interested in the pattern recognition problem. Perhaps this is due to the simple fact that the input layer of an artificial neural network is such a seductive analogue of the the biological retinal array of photoreceptors. This correspondence, coupled with the amount of the cortex devoted to vision, has long made pattern recognition a favoured area of inquiry for neural networks researchers. Indeed it could be argued that pattern recognition has been the major application domain for artificial neural networks since their invention, and some form of invariance has often been sought. The longevity of this quest, and how little it has changed to this day, is exemplified by Pitts and McCulloch (1947, p. 127), who, referring to biological nets, say:

> Numerous nets, embodied in special nervous structures, serve to classify information according to useful common characters. In vision they detect the equivalence of apparitions related by similarity and congruence, like those of a single physical thing seen from various places. . . . The equivalent apparitions in all cases share a common figure and define a group of transformations that take the equivalents into one another but preserve the figure invariant. . . . We seek general methods for designing nervous nets which recognize figures in such a way as to produce the same output for every input belonging to the figure.

Pitts and McCulloch (1947) were especially concerned with designing neural networks which were consistent with what was known about actual mammalian cortical structure. Nevertheless, their models for translation and dilation invariance have much in common with more modern work.

In his seminal paper Rosenblatt (1958, p. 386) starts with the question of "the capability of higher organisms for perceptual recognition, generalization, recall and thinking", but segues seamlessly within two paragraphs to the "stored pattern", as if the storage and recognition of patterns were such a natural way to consider these problems that no explanation was necessary. Moreover, Rosenblatt uses a "photo-perceptron" as his example, again emphasizing the fact that visual pattern recognition has long been considered a prime application for neural networks. Rosenblatt's (1958, p. 404) confidence is exemplified by:

> The question may well be raised at this point of where the perceptron's capabilities actually stop, We have seen that the system described is sufficient for pattern recognition, associative learning, and such cognitive sets as are necessary for selective attention and selective recall. The system appears to be potentially capable of temporal pattern recognition, as well as spatial recognition, involving any sensory modality or combination of modalities.

When Minsky and Papert (1969) showed this confidence to be misplaced, it was done in the context of pattern recognition. Their proofs of the limitations of perceptrons in general were couched in the language of geometry: could finite-order or diameter-limited perceptrons compute *connectedness*? Could they compute *parity* or *convexivity*? Indeed they made explicit their interest in invariance and pattern recognition, saying "we are adopting the mathematical viewpoint of Felix Klein: every interesting geometrical property is an invariant of some transformation group" [Minsky and Papert, 1969, p. 41].

Kohonen's (1972) quite general model for a linear associative memory was demonstrated with character recognition examples. Amari (1977) also used pattern recognition as the problem domain for his more complex self-organizing associator, which addresses the association of non-orthogonal patterns which are a difficulty for correlation-matrix-based associators.

Grossberg (1980), although addressing the general problem of the development of internal representations of the environment through experience, uses geometric pattern-matching as his example. This paper contains much of the theory, including the notion of feedback-induced resonance between input patterns and stored prototypes, which forms the basis of of his later *Adaptive Resonance Theory (ART)*, the subject of many papers [*e.g.* Carpenter and Grossberg, 1987; Carpenter and Grossberg, 1988].

Despite the activity mentioned above, the field of artificial neural network research had in reality been dealt a severe blow by Minsky and Papert's (1969) critique. It was the popularization of the backpropagation algorithm for training non-linear multilayer perceptrons (MLPs) in the mid 1980s that sparked a new explosion of interest in the field. It is the non-linearity of the MLPs that allows them to escape the limitations of perceptrons demonstrated by Minsky and Papert (1969). Backpropagation, in reality a fairly simple application of the chain rule, was independently discovered by several researchers. It is its ingenious distributed implementation which makes it interesting, and applicable to neural networks. In one of the influential early papers on back-propagation, Rumelhart et al. (1986b) demonstrate its ability to solve "rather abstract mathematical problems" [Rumelhart et al., 1986b, p. 631] such as parity and encoding. When, however, they want a more realistic application they turn to a simple invariant pattern recognition problem: discriminating between a "T" and a "C" independent of translation and rotation.[1]

It should be clear that the history of artificial neural networks is inextricably bound up with their application to pattern recognition. There is a very large number of papers and books on this subject, and an attempt to survey them is beyond the scope of this thesis. Excellent treatments exist, such as Bishop (1995). The remainder of this chapter will thus address the application of neural networks to invariant pattern recognition. The somewhat nebulous notion of generalization will also be addressed, since it is often confounded with invariance. Invariance, in fact, is a particularly special case.

3.1 Quantifying Generalization

Since their earliest inception, many claims have been made about the ability of neural networks to generalize. In the context of pattern classification, this usually means that the network is able to assign correct class labels to patterns that were not presented to it during training. Class membership is usually to an extent subjective – whether, for example, a given pattern is an instance of a particular letter of the alphabet is an unsolved problem mathematically, and is in fact context-sensitive even for humans [McGraw, 1992]. Generalization is, in its simplest form, due to the fact that neural networks interpolate: small changes in the input usually produce small changes in the output [Amari, 1990]. This, however, is a far from satisfactory definition of generalization for mathematical analysis.

A network which performs true invariant pattern recognition is a quite different entity to one which simply interpolates in subjectively "sensible" ways. A true invariant pattern recognition network has an output that is *by definition* invariant under the application of certain transformations to the input layer. This property should ideally be independent of the composition of any training data applied to the network. It is possible to imagine a truly invariant network which does not generalize correctly according to certain subjective criteria.

[1]A simple problem, since the characters used consisted of only five pixels.

It is important to note that many of the results and claims concerning the ability of neural networks to generalize have been made for *unstructured input data*, frequently in the context of generalization in the presence of noise. This makes plain the essential difference between generalization and invariance: in the invariance case, new input patterns belonging to the same class as a training pattern are produced by a completely deterministic mathematical transformation; in many generalization studies they are produced by random perturbations of the training pattern. Several of these approaches to measuring and improving generalization will now be reviewed.

3.1.1 Hamming Distance

Amari (1990) considers the properties of correlation matrix associative memories trained to associate two sets of independently randomly generated n-dimensional bipolar vectors, $\{\mathbf{s}^\mu\}$ and $\{\mathbf{q}^\mu\}$. The normalized Hamming distance between an input vector $\mathbf{s}_i^\mu$ and a noisy version $\mathbf{s}_i^{\mu\prime}$ is

$$D_{\mathbf{s}^\mu}(\mathbf{s}_i^\mu, \mathbf{s}_i^{\mu\prime}) = \frac{1}{2n}\sum_{i=1}^{n}\left|\mathbf{s}_i^\mu - \mathbf{s}_i^{\mu\prime}\right|. \tag{3.1}$$

Let the output of the network for an input vector $\mathbf{x}$ be $T_W\mathbf{x}$. The generalization ability of the network may be measured by considering the distance $D_{\mathbf{q}^\mu}(\mathbf{q}_i^\mu, T_W\mathbf{s}_i^{\mu\prime})$. Amari's analysis shows that the expected value of this distance is

$$D_{\mathbf{q}^\mu} = \Phi\left(-\frac{1-2D_{\mathbf{s}^\mu}}{\sqrt{r}}\right), \tag{3.2}$$

where

$$\Phi(u) = \int_{-\infty}^{u}\frac{1}{\sqrt{2\pi}}e^{-v^2/2}dv, \tag{3.3}$$

and r is the ratio of the number of training examples to the input dimensionality. This is an explicit expression of the ability of the network to generalize. It shows that the degradation of performance is very graceful, with a catastrophic transition to $D_{\mathbf{q}^\mu} \approx 1$ at $D_{\mathbf{s}^\mu} = 0.5$. A similar analysis is given for recurrent autocorrelation associative memories in the same paper.

It must be remembered, however that this is a statistical result, obtained for random input and output vectors, and n sufficiently large for the central limit theorem to be applied. Moreover, the Hamming distance is a totally inappropriate metric for pattern similarity in the case of transformation invariance. A shifted pattern may have no pixels in common with its prototype. Its normalized Hamming distance from its prototype is thus 1, whereas its "shift invariance difference" is 0. Despite this, statistical measures based on such metrics make up much of the literature on generalization, and are often critically compared to invariance techniques.

3.1.2 Comparison with a Teacher Network

Another way to quantify generalization is to consider the output of a network which is trained using the inputs and outputs of a "teacher" network with an identical architecture [Krogh and Hertz, 1991]. Since the architectures are identical, it is known that the "student" network is capable of emulating the teacher perfectly, given the correct weights. Krogh and Hertz (1991) analyze the dynamics of the training of a single linear neuron with $p = \alpha N$ of the 2^N possible input-output pairs generated by a teacher network. The inputs are generated randomly and independently. They define the generalization error F to be the squared difference between the student and teacher outputs, averaged over all 2^N possible binary inputs:

$$F = \frac{1}{N} \sum_{i=1}^{N} (u_i - w_i)^2, \tag{3.4}$$

where $\mathbf{u}$ and $\mathbf{w}$ are the teacher and student weight vectors respectively. F varies between 1 and 0 (perfect generalization) if the weight vectors are normalized to length $\sqrt{N}$.

Krogh and Hertz (1991) devote much space to the analysis of F as a function of time during training, $F(t)$. For the purposes of this thesis, the asymptotic results at infinite time are those of interest, since we are concerned with the generalization ability of trained networks. The result is:

$$\lim_{t \to \infty} F(t) = \begin{cases} 1 - \alpha & \text{for } \alpha < 1, \\ 0 & \text{for } \alpha \geq 1. \end{cases} \tag{3.5}$$

It may at first seem surprising that perfect generalization occurs when only the potentially-tiny fraction $N/2^N$ of possible inputs are presented, but this is in fact entirely unremarkable, since only N points are required to determine an $(N-1)$-dimensional hyperplane, which is all that is being done in this simple linear system.

Krogh and Hertz (1991) also investigate the effect of adding a weight decay term (see §3.3.1) to the weight-update equation. Without a weight decay term, learning only takes place in the subspace of the weight space spanned by the training input patterns. If the initial weight vector is non-zero, components outside this subspace can cause errors. The use of weight decay causes that part of the initial weight vector orthogonal to the input pattern space to decay exponentially, thus eliminating this source of error.

Winther, Lautrup and Zhang (1995) also investigate generalization performance where the target is defined by an unknown teacher network. They investigate a learning scheme which makes use of some prior knowledge to improve generalization performance. They use a Bayesian approach, in which the prior information available is $\Pr(D|V)$, the probability that the training dataset D was generated by a teacher network with parameters V. They find that the learning rule they derive is optimal in the sense that it gives the best possible expected generalization error (defined as the expected value of the squared output error over all possible inputs) for this formulation.

3.1.3 The Effective Number of Parameters

Yet another definition of generalization performance is the expected test set error $\langle E_{\text{test}}(\lambda)\rangle_{\xi\xi'}$, where λ is a regularization parameter (see §3.3), ξ is the training set and ξ' is the test set [Moody, 1992]. Moody (1992) gives an analysis of the relationship between the expected test error and the expected training set error for nonlinear learning systems such as MLPs. His main result is

$$\langle E_{\text{test}}(\lambda)\rangle_{\xi\xi'} \approx \langle E_{\text{train}}(\lambda)\rangle_{\xi\xi'} + 2\sigma_{\text{eff}}^2 \frac{p_{\text{eff}}(\lambda)}{n}, \tag{3.6}$$

where n is the size of the training set, σ_{eff} is the effective noise variance in the response, and p_{eff} is the *effective number of model parameters*. The expectation values are taken over possible training and test sets, and the result is to second order. Equation 3.6 can be compared with the exact result for linear models using a sum of squares error function and no regularizer:[2]

$$\langle E_{\text{test}}\rangle_{\xi\xi'} = \langle E_{\text{train}}\rangle_{\xi\xi'} + 2\sigma^2 \frac{p}{n}, \tag{3.7}$$

It is of interest that p_{eff} is in general not equal to the true number of model parameters p. The effective number of parameters depends upon the amount of model bias (*e.g.* network architecture), model nonlinearity and on prior model preferences determined by the regularization parameter λ and the form of the regularizer.

Many techniques for improving generalization are based on an Occam's Razor heuristic: simple models are best. Weight decay (see §3.3.1) is motivated by the intuitive notion that it causes unnecessary weights (*i.e.* parameters) to be removed. Moody's (1992) analysis confirms this, and indeed the result is applicable to all regularized unbiased linear models: p_{eff} is a decreasing function of λ, with $p_{\text{eff}}(0) = p$ and $p_{\text{eff}}(\infty) = 0$. If the model is biased and nonlinear, then in general $p_{\text{eff}}(0) \neq p$.

[2]For a particular input vector sampling probability density. See Moody (1992, p. 851).

3.1.4 The Vapnik-Chervonenkis Dimension

Another measure frequently used in the analysis of generalization in artificial neural networks for classification problems is the Vapnik-Chervonenkis (VC) dimension. Here we will follow the development of [Holden and Anthony, 1992]. Quite a few definitions are required before we proceed.

Without loss of generality, we consider a neural network with one output node, with takes on the values 0 or 1. Any given such network computes a class $\mathscr{F}$ of functions $f_{\mathbf{w}} : \mathbb{R}^n \to \{0, 1\}$, the actual function computed depending on the weight vector $\mathbf{w}$. These functions effectively split the input space $\mathbb{R}^n$ in two.

Definition 3.1 *The* hypothesis $h_{\mathbf{w}}$ *associated with a function* $f_{\mathbf{w}}$ *is defined as the subset of* $\mathbb{R}^n$ *for which* $f_{\mathbf{w}} = 1$,

$$h_{\mathbf{w}} = \{\mathbf{x} \in \mathbb{R}^n | f_{\mathbf{w}} = 1\} . \tag{3.8}$$

The hypothesis space H *computed by the network is the set*

$$H = \left\{h_{\mathbf{w}} | \mathbf{w} \in \mathbb{R}^W\right\} \tag{3.9}$$

of all hypotheses where W *is the total number of weights in the network.*

The VC dimension can be regarded as a measure of the expressive capacity of the hypothesis space of a network.

Definition 3.2 *Given a set* $S \subseteq \mathbb{R}^n$ *and some function* $f_{\mathbf{w}} \in \mathscr{F}$, *the* dichotomy (S^+, S^-) *of* S *induced by* $f_{\mathbf{w}}$ *is defined to be the partition of* S *into the disjoint subsets* S^+ *and* S^- *where* $S^+ \cup S^- = S$ *and* $\mathbf{x} \in S^+$ *if* $f_{\mathbf{w}} = 1$, $\mathbf{x} \in S^-$ *if* $f_{\mathbf{w}} = 0$.

Definition 3.3 *Given a hypothesis space* H *and* $S \subseteq \mathbb{R}^n$, *we define* $\triangle_H(S)$ *as the set*

$$\triangle_H(S) = \{h \cap S | h \in H\} . \tag{3.10}$$

We say that S *is* shattered *by* H *if* $\triangle_H(S) = 2^S$ *where* 2^S *is the set of all subsets of* S.

We are now in a position to define the *growth function* and the VC dimension.

Definition 3.4 (Growth Function) *The growth function is defined on the set of positive integers as*

$$\triangle_H(i) = \max_{S \subseteq \mathbb{R}^n, |S| = i} |\triangle_H(S)| . \tag{3.11}$$

Definition 3.5 (Vapnik-Chervonenkis dimension) *The VC dimension* $\mathscr{V}(H)$ *of a hypothesis space* H *is the largest integer* i *such that* $\triangle_H(i) = 2^i$, *or infinity if no such* i *exists.*

The growth function thus gives the maximum number of dichotomies induced by $\mathscr{F}$ for any set of i points. The VC dimension is the size of the largest set of points which can be shattered by H. As an example, consider the set of simple linear discriminant functions:

$$\mathscr{F}_L = \left\{ \mathrm{sign}\left[w_0 + w_1 x_1 + \cdots + w_n x_n\right] \;\mid\; \mathbf{w} \in \mathbb{R}^{n+1} \right\}. \tag{3.12}$$

It is well-known that $\mathscr{V}(\mathscr{F}_L) = n+1$. Such exact results are difficult to obtain for more complex systems such as neural networks, though some bounds have been found. For multi-layered perceptrons with W weights and N hidden nodes, the class of functions $\mathscr{F}_{\mathrm{MLP}}(W, N)$ has the bound [Baum and Haussler, 1989]:

$$\mathscr{V}(\mathscr{F}_{\mathrm{MLP}}(W, N)) \leq 2W \log_2(eN). \tag{3.13}$$

Another result, commonly known as *Sauer's lemma*, provides an upper bound for the growth function, given the VC dimension of a class of functions $\mathscr{F}$ [Holden and Anthony, 1992].

Sauer's Lemma *Given a class $\mathscr{F}$ of functions for which $\mathscr{V}(\mathscr{F}) = d \geq 0$ and $d < \infty$,*

$$\triangle_{\mathscr{F}}(k) \leq 1 + \sum_{i=1}^{d} \binom{k}{i}. \tag{3.14}$$

For finite $\mathscr{V}(\mathscr{F})$, another useful bound is

$$\triangle_{\mathscr{F}}(k) \leq k^{\mathscr{V}(\mathscr{F})} + 1. \tag{3.15}$$

The VC dimension can be applied to the analysis of the ability of neural networks to generalize. As discussed above, a neural network may be considered to compute a class $\mathscr{F}$ of functions. Training of the network can be regarded as a process which tries to find some $f_{\mathbf{w}} \in \mathscr{F}$ which is in some sense a "good approximation" to a target function f_T on the given set of training examples. If an input vector $\mathbf{x} \in \mathbb{R}^n$ is chosen randomly according to an arbitrary distribution P, we can define $\pi_{f_{\mathbf{w}}}$ to be the probability that the network agrees with the target function for this input vector:

$$\pi_{f_{\mathbf{w}}} = \Pr\left[f_{\mathbf{w}}(\mathbf{x}) = f_T(\mathbf{x})\right]. \tag{3.16}$$

Now consider a sequence of k training examples, $\{(\mathbf{x}_1, f_T(\mathbf{x}_1)), \ldots, (\mathbf{x}_k, f_T(\mathbf{x}_k))\}$, chosen at random according to P. Let $v_{f_{\mathbf{w}}}$ be the fraction of the inputs for which $f_{\mathbf{w}}$ agrees with f_T. Training consists of choosing a weight vector $\mathbf{w}$ according to the value of $v_{f_{\mathbf{w}}}$. Consequently, it is necessary to know if $v_{f_{\mathbf{w}}}$ converges to $\pi_{f_{\mathbf{w}}}$ uniformly for all $f_{\mathbf{w}} \in \mathscr{F}$

as k becomes large. If it does not, we may choose a $v_{f_\mathbf{w}}$ for which $\pi_{f_\mathbf{w}}$ is relatively low, and the probability of good generalization is thus also low.

An inequality due to Vapnik[3] gives a bound for the probability that there is a function $f_\mathbf{w} \in \mathscr{F}$ for which $\pi_{f_\mathbf{w}}$ and $v_{f_\mathbf{w}}$ are significantly different. For a given threshold value α,

$$\Pr\left[\sup_{f_\mathbf{w} \in \mathscr{F}} \frac{v_{f_\mathbf{w}} - \pi_{f_\mathbf{w}}}{\sqrt{1-\pi_{f_\mathbf{w}}}} > \alpha\right] \leq 4\triangle_{\mathscr{F}}(2k)e^{\frac{-\alpha^2 k}{4}}. \tag{3.17}$$

The significance of this result arises because for finite $\mathscr{V}(\mathscr{F})$, by Equation 3.15, the growth function $\triangle_{\mathscr{F}}(k)$ is bounded above by a polynomial function of k. Since the right-hand side of Equation 3.17 decays exponentially as a function of k, the generalization error can be made arbitrarily small by choosing a sufficiently large k. Moreover, Equation 3.17 provides a bound on the rate of convergence during training.

Summary

Although these are useful and powerful ways of characterizing the ability of neural networks to generalize, they all address the generalization problem from the perspective of a learning environment in which the input and output vectors are essentially drawn at random from some population, according to some probability distribution. This may be appropriate when attempting to estimate the ability of a network to generalize in the presence of noise with known statistics, or when trying to quantify the representational power of a given network architecture. Theoretical results such as these, however, would seem to have little bearing on the design of networks where the transformations of the input patterns are known *a priori*. This serves to emphasize the difference between generalization and invariance.

3.2 Improving Generalization

3.2.1 Large Training Sets

The most basic approach to improving the generalization performance of a neural network, however it might be defined, is to increase the size of the training set. Many of the results described in §3.1 provide estimates or bounds for the number of training examples required to achieve a specified generalization performance. In the most basic sense, it is desirable to have sufficient training examples to span the parameter space of the network (see §3.1.2).

The use of large training sets is such a ubiquitous feature of neural network applications that it would be possible to cite thousands of references. As examples of the sizes

[3]This is from Holden and Anthony (1992).

of training sets often employed, Plaut and Hinton (1987) use 250,000 training examples for a network with 54 input nodes and 2 output nodes. This is discussed in detail in §4.3. Fontaine and Shastri (1992) use 5450 training images for their digit recognition system. Khotanzad and Lu (1990) use 24 slightly perturbed images of each of the 26 letters of the alphabet for their character recognition system. This approach is almost universal in the neural networks literature.

One of the aims of the Model-Based Neural Network approach is to obviate the need for such large training sets. By designing networks with guaranteed invariance properties, the need to include transformed and perturbed images in the training set can be removed. This aim will form a part of much that follows in this thesis.

3.2.2 Cognitron and Neocognitron

One of the most durable families of neural networks for pattern recognition has been that of Cognitron [Fukushima, 1975] and its successor, Neocognitron [Fukushima, 1980]. Cognitron is a self-organizing multi-layered network that is trained using a Hebbian scheme [Hebb, 1949]. Much of the inspiration for Cognitron was biological, and Fukushima made some claims now known to be false (*e.g.* "it is known that the capability of a layered neural network is greatly enlarged if the number of the neural layers is increased." [Fukushima, 1975, p. 121].

The basic structure of the Cognitron is a series of cascade layers consisting of neurons with limited "receptive fields": a neuron in the layer immediately following the input layer would have connection only to input neurons in a limited spatial area of the input layer. As one progresses through the network, the receptive fields of neurons increase in size, the idea being that neurons in the final layer respond to features on the entire input layer. Cognitron was trained using a small number of simple synthetic digits and numerals, and some associative memory properties were demonstrated.

The relevance of these networks to this study dates from Neocognitron [Fukushima, 1980] Fukushima (1980, p. 193) claims "The network is self-organized by "learning without a teacher", and acquires an ability to recognize stimulus patterns based on the geometrical similarity (Gestalt) of their shapes without effected by their positions."(sic). With Neocognitron, the series of cascaded layers used by Cognitron was retained, but the connection patterns and weights of neurons were constrained to be identical within a given plane, each connected to a spatially-shifted region of the layer above. There could be several planes in a given layer. "Hence, all the cells in a single cell-plane have receptive fields of the same function, but at different positions" [Fukushima, 1980, p. 195]. This is conceptually similar to the MBNN systems introduced in Chapter 4. Neocognitron was trained with a single noise-free example of each of the digits {0,1,2,3,4}, and was shown to be able to classify some distorted and noisy examples correctly.

This work was extended to larger datasets in Fukushima, Miyake and Ito (1983) and Miyake and Fukushima (1984). Supervised learning was introduced into the model, along with other variations. The authors continued to make strong claims for the invariant pattern recognition capabilities of their systems, *e.g.* "each cell of the deepest layer of the network responds selectively to a specific stimulus pattern and is not affected by the distortion in shape or the shift in position of the pattern" [Fukushima et al., 1983, p. 826].

The claims for the invariant properties of the Neocognitron were made without rigorous mathematical analysis. Barnard and Casasent (1990) show that the performance of the Neocognitron *is not intrinsically shift-invariant*, and that any invariance comes as a trade-off with classification sensitivity. In fact, Neocognitron's apparent shift invariance is basically the result of summing the total energy of the input layer. Despite this, work has continued based on the assumption that Neocognitron *is* shift-invariant: Himes and Iñigo (1992) combine log-polar mapping with the Neocognitron in an attempt to produce a shift-, scale- and translation-invariant system, and [Li and Wu, 1993] introduce layers that produce rotated versions of the input pattern to add rotation invariance to Neocognitron.

3.2.3 Cascade-Correlation

The Cascade-Correlation algorithm, introduced by Fahlman and Lebiere (1990), is a combined architecture and supervised learning algorithm. Rather than just adjusting the connection weights in a pre-determined network structure, Cascade-Correlation starts with a minimal network (no hidden nodes), and adds new hidden nodes one after another. Once a node is trained, its input weights are never trained again. New nodes have all existing nodes as inputs, and are connected to all the outputs. This means that no backpropagation is necessary, since the only weights ever adjusted are between the output units and all the other nodes in the network: the simple Widrow-Hoff delta rule can be used [Widrow and Hoff, 1960].

When a new hidden unit is needed a pool of *candidate units* is created. The candidate units have input connections from all the input units and existing hidden units. The candidate units are trained to maximize the correlation between the candidate units' outputs and the residual output error of the network. When training is deemed to have converged, the candidate unit with the highest correlation with the output error is selected and installed as the new hidden unit. Its input-side weights are fixed, and the process is then repeated until the output error of the network is acceptable. The architecture is also extensible to recurrent networks [Fahlman, 1991].

Fahlman and Lebiere (1990, p. 12) state that "by training one unit at a time instead of training the whole network at once, we can speed up the learning process considerably, while still creating a reasonably small net that generalizes well". In terms of generalization, Cascade-Correlation belongs to the “less is more” school. It builds a network that is near-minimal in terms of the number of nodes and connections necessary to characterize the training set. Many techniques for improving generalization (weight decay, pruning, *etc.*) are based on the notion that fewer network parameters leads to better performance.

The drawback is that in Cascade-Correlation *everything* about the network is determined by the training set. Such a technique must be vulnerable to overfitting noise in training sets unless the training sets are large. Cascade-Correlation does not address the problem of invariance at all. It is worth noting that the notion that smaller networks generalize better has been challenged. Lawrence, Giles and Tsoi (1996), for instance, report that oversize networks can result in lower training and generalization error for certain problems.

3.2.4 Massive Weight-Sharing

Lautrup, Hansen, Law, Mørch, Svarer and Strother (1994, p. 1) state the generalization problem very nicely:

> The aim of learning is to match a model to data in such a way that generalization ability ensues. Whether this is possible depends intricately on the training procedure and on the architecture of the learning machine. If the model is overly restrictive, it cannot “capture the rule”, hence it fails to implement the the training set. On the other hand if we train a model with too high capacity for a given data set, it is unlikely that the model will generalize. The reason is that there will be many different ways to implement the training set in the model, *i.e.* to generalize from it.

Lautrup et al. (1994, p. 1) consider generalization for extremely ill-posed problems: problems with a vast number of highly-correlated inputs, but only a small number of available training patterns. Such problems are common in image and spectral analysis. Their solution uses a similar approach to that of Singular-Valued Decomposition [Press et al., 1992]: the problem is transposed from a high-dimensional input space to a low-dimensional “signal-space”. In neural network terms, the effect is to induce *massive weight-sharing* by constraining the network weights to a low-dimensional subspace, rather than letting training explore the entire space of dimensionality equal to the number of weights. This is done by expanding the weights $\{\mathbf{w}\}$ so that they are expressed as linear combinations of the set of vectors $\{\mathbf{x}\}$ which span the signal space of training inputs.

$$\mathbf{w}_j = \sum_{\alpha=1}^{p} \gamma_j^{\alpha} \mathbf{x}_{\alpha} \tag{3.18}$$

Optimization is then performed on the coefficients $\{\gamma_j^\alpha\}$ rather than on the weights.

This "trick" is only possible if there are strong correlations between the components of the input vector. In fact this approach is directly related to Principal Components Analysis. Principal Components Analysis gives direct access to a set of orthogonal vectors which span the input space, and it is possible to formulate Massive Weight-Sharing to use these vectors as its basis. Principal Components Analysis also ranks these vectors in terms of their contributions to the input variance. It would be possible to reduce the dimensionality of the effective weight space by truncating the spanning set according to this ranking.

Lautrup et al. (1994) report that this technique was used to reduce the dimensionality of a problem from Positron Emission Tomography from 141,375 to 47. The generalization results reported are in fact not very good. This is attributed to the fact that most of the variance (the first 7 principal components) is in fact inter-subject variance.

3.2.5 Pruning Techniques

Another approach to reducing the dimensionality of a network's weight space in the hope that this will improve generalization is to "prune" weights which are deemed to be insignificant.[4] Rather than simply removing small weights by analogy with Weight Decay (see §3.3.1), pruning techniques take a more sophisticated approach. Optimal Brain Damage [Le Cun, Denker and Solla, 1989] removes the weights that have the least effect on the training error, based on a diagonal approximation of the Hessian of the network. The pruned network must be retrained, introducing additional computational cost. The diagonal assumption is also inaccurate [Asriel U. Levin and Moody, 1994]. Other pruning techniques exist, such as Principal Components Pruning [Asriel U. Levin and Moody, 1994], but all are training set-driven, and none address the invariance problem.

3.2.6 Using Prior Information

One of the aims of the Model-Based Neural Networks approach is to provide a mechanism by which the network designer can utilize his prior expert knowledge of desired network performance and the problem domain to constrain the construction of networks. Several existing approaches to improving network performance make use of prior information, in a variety of ways.

Omlin and Giles (1992) discuss the training of recurrent networks using "hints". By this they mean that some weights are set to large absolute values before training starts, rather than the usual initialization of all weights to small random values. Their

[4] To prune a weight means to remove the corresponding connection from the network.

application is the recognition of regular languages defined by transition grammars. Hints are inserted by setting weights corresponding to known allowed or forbidden transitions to large or small values respectively. Only partial knowledge of the target grammar is included as hints. They found that this prior knowledge could indeed improve convergence time during training, but that generalization performance with hints was unpredictable, and "did not suffer significantly by using hints" [Omlin and Giles, 1992, p. 6].

The ability of the backpropagation algorithm to make use of prior information about data from known distributions is investigated by Barnard and Botha (1993). They take a Bayesian perspective: the prior knowledge to which they refer is the probability that an unknown input pattern belongs to a particular class. They report that linear least mean-squares classifiers can be shown to place their decision boundaries sub-optimally on this basis: they tend to guess the most likely class too often. In this case the prior information is implicit in the training set, in the form of the frequency of class examples. They find that backpropagation on multi-layered networks tends to employ such prior information sub-optimally, although the difference from a theoretically-determined Bayes optimal classifier was slight. Experimental results indicate that network performance approaches the Bayes limit as the size of the training set increases. This is in contrast to the linear classifier, where the performance is a function of the values of the prior probabilities.

Prem, Mackinger and Dorffner (1993) introduce a method for inserting *a priori* symbolic knowledge into a neural network called *Concept Support*. Rather than relying upon setting particular weights, knowledge is inserted by pre-training the network on concepts or rules thought to be important for the eventual task to be performed by the network. It is hoped that this will address problems such as the fact that not all necessary knowledge might be present in the training data (for generalization), and that pre-programming might help the network to avoid spurious local minima during subsequent training. Prior knowledge inserted in this way can always be "over-written" during later training.

As an example of Concept Support, Prem et al. (1993) considered the problem of diagnosing coronary artery disease from thallium-201 scintigrams of the heart. Prior information could take the form of statements such as "there is a difference between the scans of men and women". A network was initially trained to classify scans as belonging to a man or a woman. The output units were then removed, and replaced by a new unit. The network was then trained to classify the input for the presence of disease on the left or the right side of the heart. The correct test set prediction rate rose from 60% to 73% as a result of this Concept Support. The experiments in Prem et al. (1993) all show improved performance as a result of this technique, but the treatment is entirely empirical.

3.3 Regularization Approaches to Invariance

The most common method used to attempt to improve generalization is to augment the usual least squares cost function E (Equation 1.3) with some auxiliary function of the weights, producing a new cost function C:

$$C = \frac{1}{2} \sum_{l} \sum_{c} \left(y_{lc} - t_{lc}\right)^2 + \lambda g\left(w_{ij}\right). \tag{3.19}$$

The choice of this auxiliary function $g\left(w_{ij}\right)$ often seems to be based more on intuition, or empirical knowledge of network performance, than any specific knowledge of the task that the network is supposed to perform. The training process then becomes a compromise between minimizing Equation 1.3 and minimizing the auxiliary function. The auxiliary function is often a "complexity measure" of the network. Consequently, there is an underlying heuristic in this approach: less "complex" networks generalize better.

3.3.1 Weight Decay

Probably the most common auxiliary function is the sum of the squares of all the weights in the network [Krogh and Hertz, 1992; Plaut and Hinton, 1987]:

$$C = \frac{1}{2} \sum_{l} \sum_{c} \left(y_{lc} - t_{lc}\right)^2 + \lambda \frac{1}{2} \sum_{i} w_{ij}^2. \tag{3.20}$$

This cost function leads to a simple "weight decay" term in the weight update equation (Equation 4.14):

$$w_k\left(t+1\right) = w_k\left(t\right) - \varepsilon \frac{\partial E}{\partial w_k\left(t\right)} - \lambda w_k. \tag{3.21}$$

Here E is the sum of squared errors from Equation 1.3, and the "momentum" term has been omitted. This is easy to implement, and limits the development of any weights with "large" absolute value. The trade-off with the minimization of E is determined by the parameter λ. The argument for using this cost function is that it prevents the network from "over-fitting" the data, since weights that are unimportant for the task (and consequently have $\frac{\partial E}{\partial w_k} \approx 0$) will decay to zero at a rate determined by λ. Another justification that is claimed for this approach is that it minimizes the sensitivity of the output to noise in the input [Nowlan and Hinton, 1992a], although this claim is based on results derived for a linear system. This approach can also be interpreted, from a Bayesian perspective, as specifying a particular prior probability distribution of the weights. This is discussed below as a special case of "Soft Weight-Sharing".

3.3.2 Soft-Weight Sharing

Soft Weight-Sharing [Nowlan and Hinton, 1992a; Nowlan and Hinton, 1992b] is, in essence, a generalization of the Weight Decay approach. Rather than introducing an auxiliary function that causes all weights to have a tendency to decay to zero, they model the weight space as being the result of having selected each weight from a population having a prior probability distribution given by a mixture of Gaussian probability distributions, with nonzero means. The complexity measure is then based on how probable the current state of the network is, given this prior distribution. The aim of this technique is to cause the weights in the network to cluster in "families" close to the means of the Gaussians. Weight decay can be viewed as the result of this technique applied using a prior distribution consisting of a single zero-mean Gaussian with variance $\frac{1}{\lambda}$. The means, variances and mixing proportions of the Gaussians are adjusted simultaneously with the weights during training.

Like weight decay, Soft Weight-Sharing is based on assumptions about the desired properties of the weights that are not derived from the task at hand, and consequently is also vulnerable to small training sets.

3.3.3 Tangent Prop

The aim of Tangent Prop [Simard, Victorri, Cun and Denker, 1992] is to incorporate *a priori* knowledge of the desired invariances of a network into the training procedure. If the response of the network is invariant under some distortion (*e.g.*, rotation, translation, dilation) of the input pattern, then the derivative of its output with respect to such distortions should be zero. For a truly invariant system, this derivative would vanish everywhere. Tangent Prop seeks neural networks invariant under *small* spatial distortions of the training patterns. It thus seeks weight configurations for which the derivative vanishes *for the given training examples*. This approach can lead to "local" invariance (in the neighbourhood of the training patterns), but can only produce global variance by chance.

When a pattern P is transformed by a transformation s depending on n parameters, the set of all possible transformed patterns $S(P)$ is a manifold of at most n dimensions. The patterns in $S(P)$ obtained from *small* transformations of P (*i.e.* the part of $S(P)$ close to P) can be approximated by the tangent plane to the manifold $S(P)$ at point P. Small transformations of P can be obtained by adding to P a linear combination of the vectors that span the tangent plane.

Tangent Prop approximates the vectors corresponding to such small transformations by taking the finite difference between actual training patterns and distorted versions computed from them. The usual least squares error function is augmented with a regularizing term E_r that is proportional to the square of the error of the derivatives of the outputs with respect to the known desired invariances. The weight-update rule becomes:

$$\Delta w_{ij} = -\varepsilon \frac{\partial}{\partial w_{ij}}(E + \lambda E_r), \tag{3.22}$$

where, for an input pattern x, network output $G(x)$ and a transformation operator parameterized by α, $s_i(\alpha, x)$,

$$\begin{aligned} E_r &= \sum_{x \in \text{training set}} E_r(x) \\ E_r(x) &= \sum_i \left\| K_i(x) - \left(\frac{\partial G(s_i(\alpha, x))}{\partial \alpha} \right)_{\alpha=0} \right\|^2, \end{aligned} \tag{3.23}$$

where $K_i(x)$ is the desired directional derivative of G in the direction induced by transformation s_i applied to pattern x. For the case of *local invariance*, $K_i(x)$ is simply set to zero. The directional derivative can be written

$$\left. \frac{\partial G(s_i(\alpha, x))}{\partial \alpha} \right|_{\alpha=0} = J_G(x) \left. \frac{\partial s_i(\alpha, x)}{\partial \alpha} \right|_{\alpha=0}, \tag{3.24}$$

where $J_G(x)$ is the Jacobian of the network transformation G. This directional derivative can be computed by a forward propagation of the estimated tangent vector through a linearized version of the network.

Training a network using Tangent Prop is thus similar to using plain backpropagation, with the addition that tangent vectors are also propagated through the network. The technique provides a means of introducing *a priori* known desired invariances directly as a constraint on the training of the network, but can only achieve invariance in the neighbourhood of its training patterns. It still requires a large training set if invariance is desired over an entire input space.

Leen (1995) makes explicit the relationship between the use of the regularizer proposed in Tangent Prop and simply augmenting the training set with the distorted versions of the input patterns used to compute the tangent vectors. He shows that learning is equivalent if the weighting term λ in Equation 3.22 is chosen to be the variance of the distortions introduced into the original training set.[5] This shows that regularization in this form really belongs to the family of large training set methods, rather than being a true constraint on the network architecture or weight space.

[5]The equivalence only holds up to order $\mathcal{O}(\sigma^2)$. This is reasonable since Tangent Prop's approximations only hold for small distortions.

3.4 Invariant Representations and Features

3.4.1 Invariant Representations

Associative Memory and ART techniques

In this thesis, many of the Model-Based Neural Networks proposed consist of a neural system which pre-processes the input pattern to extract invariant features, or transforms the input pattern into a representation which is invariant under certain transformations. Such networks have been proposed before, although usually restricted to very simple transformations and input patterns. Several are described below.

Associative Memories

The generalization properties of associative memories have been discussed in §3.1.1. Some attempts have been made to construct associative memories that can correctly associate transformed versions of the input patterns with which they were trained. Kree and Zippelius (1988) propose a network that can recognize topologically-equivalent graphs. The system consists of two coupled networks: a Hopfield network to store and retrieve that data, and a preprocessor to transform the input data. Graph-matching is performed by generating isomorphisms of the input graph and using these to try to retrieve a learnt graph. The use of a network that transforms the input data to an invariant representation makes this a MBNN in the sense of this thesis.

Another scheme is a neural network implementation of cross-correlation [Austin, 1989b; Austin, 1989a]. The input patterns used are n-dimensional binary vectors: a simple straight line of 1s and 0s. A preprocessing system is used which computes all possible shifts of the linear input pattern, and computes the Hamming distance between these and the stored patterns. The minimum Hamming distance indicates the best match. It is shown how a more efficient pattern encoding scheme can reduce storage requirements. Importantly for this thesis, Austin shows how this may be implemented in a neural network, making this a MBNN in our sense, albeit for a very simple problem. Since this network computes all possible transformed versions of the input pattern, it is in fact doing a form of cross-correlation.

Adaptive Resonance Theory (ART) Networks

Another system which uses a pre-processing network to produce an invariant representation of its input pattern is proposed by Srinivasa and Jouaneh (1992). Here the classification stage used is an ART network [Carpenter and Grossberg, 1987; Carpenter and Grossberg, 1988]. The invariance net is designed to be invariant under translations and 90° rotations of an 8×8 input layer. The invariance net thus consisted of 256 "slabs" of 8×8 neurons each, each slab corresponding a a particular combination of shift and rotation parameters. This approach is again just multidimensional cross-correlation, and is extremely expensive in both space and computation. Each slab is connected with equal weights to the classifier. Since all possible allowed transformations of the patterns are computed for the input pattern, this 16384-dimensional representation of the 64-dimension input pattern is indeed invariant, though far from efficient. This system is shown to be able to classify correctly transformed versions of 4 uppercase alphabetic characters. A version of this system that has better noise tolerance is given in [Srinivasa and Jouaneh, 1993].

3.4.2 Invariant Features

The simplest way to do invariant pattern recognition with a neural network is to present the network not with the input pattern itself, but with an invariant representation of the pattern computed by some other system. Training and recognition take place entirely with the invariant representation, and the neural network is used only as a classifier. Any of the invariant pattern representations discussed in Chapter 2 could be used in this way.

Some studies have been done which investigate the performance of neural network classifiers for such data. For example, Khotanzad and Lu (1990) compare MLPs trained with backpropagation trained with geometrical moments (see §2.2.1) and Zernike moments (see §2.2.2). They found that performance with Zernike moment features was superior, and in particular more robust to noise. Since the invariance in this approach does not arise from the networks themselves, it is not considered at greater length here.

3.4.3 Higher-Order Neural Networks

An important class of neural networks in the context of this thesis are Higher-Order Neural Networks (HONNS). HONNs can be considered to be a form of MBNN specifically designed for invariant pattern recognition. They differ from any of the MBNNs introduced in this thesis, but they share the same goal: to create a neural network architecture which gives a response with guaranteed invariance properties. Several forms of HONN exist which address various combinations of shift, rotation and scale invariance [Perantonis and Lisboa, 1992; Redding, Kowalczyk and Downs, 1993; Schmidt and Davis, 1993; Spirkovska and Reid, 1994; Delopoulos, Tirakis and Kollias, 1994]. Although these networks achieve some success on this task, they have several severe disadvantages, most importantly the storage requirements and scalability of the approach.

The layer immediately following the input layer of a HONN contains nodes which compute weighted sums of *products* of the activation of the input nodes. For a third order network, the output of a hidden node y_i is

$$y_i = f\left(\sum_j \sum_k \sum_l w_{ijk} x_j x_k x_l\right), \tag{3.25}$$

where f is the transfer function, and $\{x_j, x_k, x_l\}$ are the activations of three input nodes. Invariance is introduced into HONNs by choosing the weights for pairs (2nd order HONNs) or triples (3rd order HONNs) of input nodes with a given geometrical relationship to be the same. Shift invariance could be obtained with a 2nd order HONN by ensuring that all pairs of nodes joined by a line of a given length and orientation have the same weight. Shift, rotation and scale invariance can be obtained with a 3rd order HONN by ensuring that all triples of nodes corresponding to similar triangles have the same weight.

The most significant problem is the combinatorial explosion of the number of weights required. Input layers for pattern recognition networks typically have hundreds of nodes. A hidden node in a 3rd order HONN would thus have to sum $\binom{100}{3} = 161,700$ product terms. For the optical character recognition problem addressed in Chapter 7, which uses a 56×57 node input layer, each hidden node would have 1.08×10^{10} weights. Not only must these weights be stored, but all of the corresponding triples of input nodes must be searched to find those sharing the required geometrical relationships. This is clearly prohibitive.

There are other problems relating to sensitivity to noise, and sampling problems. On discrete lattices, for instance, there are many more possible similar triangles at large scales than at small ones.

It is not surprising that applications of HONNs do not usually use them in this raw form. Perantonis and Lisboa (1992) present a technique for reducing the required number of weights for third order HONNs based on "approximately similar triangles", defined by angles falling within certain specified tolerances. This allows triangles to be coarsely binned into approximately similar classes, but still requires approximately 10^7 weights for a 400 pixel input layer. This is further restricted by only storing values for triangles with a vertex at a certain point: other triangles are translated there as needed (not a neural operation). They found that this system performed better than a system using Zernike moment features for the recognition of typed computer-transformed numerals, both with and without noise. The best test set performance for patterns which had been both rotated and scaled was 79% correct for the HONN and 63% correct using Zernike moments.

Spirkovska and Reid (1994) investigate several other schemes for reducing the dimensionality of the HONN weight space. The consider local connectivity, where triples are only connected if all the inter-pixel distances fall below some bound, this sacrificing a degree of scale invariance. Other options include sampled connectivity and regional connectivity. They finally settle upon a coarse coding scheme, where overlaying fields of coarse pixels are used to represent an input field composed of smaller pixels. For example, a 10×10 input layer may be represented without loss of information by 4 layers with 5×5 pixels of twice the size, each offset by one (small) pixel horizontally or vertically. The advantage is that $\binom{10}{3} = 120$, whereas $4\binom{5}{3} = 40$. The HONN is constructed using triples taken from the coarse layers. At larger input layer sizes the difference is even more dramatic.