Extra Output Biased Learning[1]

Yeong-Ho Yu
Robert F. Simmons

Department of Computer Sciences
and
Artificial Intelligence Laboratory
Taylor Hall 2.124, University of Texas at Austin,
Austin, Texas 78712-1188

March 1990 AI90-128

Abstract

One way to view feed-forward neural networks is to regard them as *mapping functions* from the *input space* to the *output space*. In this view, the immediate goal of *back-propagation* in training such a network is to find a correct mapping function among the set of all possible mapping functions of the given topology. However, finding a correct one is sometimes not an easy task, especially when there are local minima. Moreover, it is harder to train a network so that it can produce correct output not only for training patterns but for novel patterns which the network has never seen before. This so-called *generalization* capability has been poorly understood, and there is little guidance for achieving a better generalization. This paper presents a unified viewpoint for the training and generalization of a feed-forward network, and a technique for improved training and generalization based on this viewpoint.

To Appear in the Proceedings of the International Joint Conference on Neural Networks 1990

[1]This work is sponsored by the Army Research Office under contract DAAG29-84-K-0060.

1 Introduction

Training a feed-forward network through back-propagation requires finding a set of weights for the network so that it produces the correct target output when an input in the training set is given. However, in the presence of local minima, sometimes it is very hard to find such a set.

Another important aspect of training a network is to achieve a *generalization* capability of the network so that it may produce the correct output for a novel input. This capability is essential to any kind of learning which goes beyond simple memorization of the training patterns.

However, in practice, there is little guidance for better training and generalization except some empirical results [Kukich 88] concerning various factors such as the topology of the network, encoding schemes, training schedules, and values of various parameters. These results provide us with motivations for trying a different network or a different encoding scheme when the current ones fail. Nevertheless, these results hardly help us in deciding those factors for a given problem in the first place.

The most overlooked factor so far has been the training patterns themselves. As a matter of fact, the most direct way to affect the training and generalization is to change the training patterns. This research presents a viewpoint about training and generalization to explain how the training patterns affect those performances. Then, based on this viewpoint, a technique for a better training and generalization is described.

2 Viewpoint of Mapping Functions

One way to view feed-forward networks[2] is to regard them as *mapping functions* from the *input space* to the *output space* [Nilsson 65]. For instance, a two-layer network of L input units, M hidden units, and N output units has L-dimensional input space, I, and N-dimensional output space, O. Then, the network is thought to be mapping a point in I space to another in O space as below.

$$f : I \longmapsto O \tag{1}$$

Furthermore, since the network has a hidden layer which can be described as a M-dimensional space, H, the mapping function f is the composition of two functions as below.

$$f = h \circ g \tag{2}$$
$$\text{where } g : I \longmapsto H, \;\; h : H \longmapsto O$$

[2]In this paper, two-layer (i.e., one layer of hidden units) feed forward networks are assumed. However, the results are applicable to networks of multiple hidden layers.

Given a network, its topology determines all the possible mapping functions it can represent. In this viewpoint, training such a network is to select one of them, and the way to do it is to provide an appropriate set of training patterns which can be viewed as a *specification* for the desired mapping function. Sometimes, the inputs in the training patterns span the whole input space as do the training patterns of the XOR problem[3]. In these cases, the training patterns provide the full specification and there is only one mapping function f for the specification, even though there may be many combinations of different g's and h's for it. More often, however, the inputs are a subset of the whole input space. In this case, the training patterns are a partial specification and there are many possible mapping functions, $f_1, f_2, \ldots, f_n$ for it.

In the beginning, the initial network represents a mapping function which is not the specified one, and iterative back-propagations modify the current mapping function to one closer to the specified one. If the training finishes successfully, the final mapping function will be the specified one or one very close to it. However, the back-propagation algorithm has a risk of falling into a so-called *local minimum*. The mapping functions of local minima may be viewed as those which are correct only for a partial input space - *partially correct* mapping functions.

If the training finishes successfully for a partial specification, then the network will always produce the correct output for the input used in the training patterns. However, for novel inputs, the network's performance may vary drastically over several trials, since the network has had the freedom of choosing any function from the set of functions belonging to the partial specification.

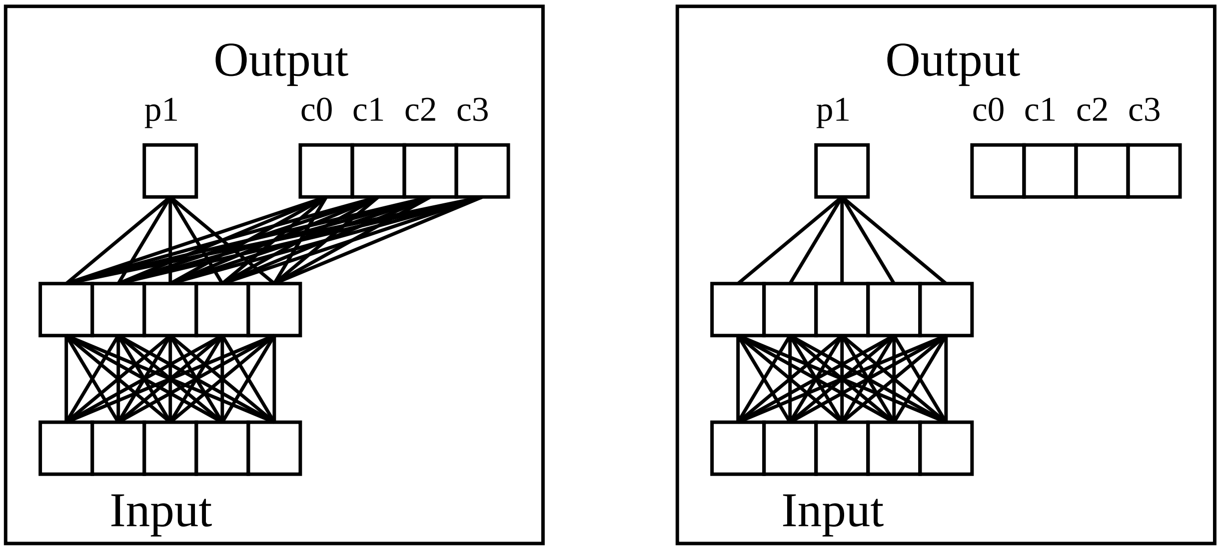
Therefore, one way to get a better generalization is to provide training patterns which cover as much input space as possible. However, this is impractical for a large number of input units [Touretzky 89], or unsuccessful in the presence of local minima which correctly cover only a part of the input space given in the training patterns.

Another way, which is described in the next two sections, is to provide directly additional relevant information about the desired mapping function as a part of the outputs in the training patterns.

3 Parallel Training of Multiple Mapping Functions

Input and output patterns are usually direct mappings; one output pattern corresponding to each input pattern. For parity and XOR, one output bit is on or zero according to the input pattern. In addition or subtraction, the output units represent a single binary number corresponding to each pair of input patterns. However, there is no absolute reason for this simplicity. It is possible to design a network which has different semantics for different parts of outputs. For instance, in Figure 1(a), one output unit is for the parity of the input while the other 4 units represent the number of 1's in the input.

[3]This is true only when we limit the values of input units to 0's and 1's.



(a) Parity and Counter (b) Detaching Counter

Figure 1: N-Bit Parity and Counter

In this case, it is as if we merge two networks into one where two networks share the connections from the input units to hidden units. In terms of our mapping-function viewpoint and equations (1) and (2) above, the network has two parallel mapping functions f_1 and f_2 where $f_1 = g \circ h_1$ and $f_2 = g \circ h_2$. The difference in training such a merged network from training two separate networks is that the merged network shares g together while two separate networks may have different g's. In other words, training a merged network will allow fewer degrees of freedom in picking the mapping functions for g. Since g and h are mutually dependent, the merged network results in fewer degrees of freedom for h, too. Also, in the cases of partial specifications, fewer degrees of freedom for h and g mean fewer degrees of freedom for f. In some extreme cases where two mappings interfere with each other — i.e., no common g, the network will lead one or both mapping functions into local minima.

This argument suggests that fewer degrees of freedom for f can be achieved by specifying another function, f', mapping the same input space through the same hidden units to an augmented output specification. For parity problems, f' can be a counter that indicates the number of 1's in the input; for language mapping problems, f' might be added syntactic or semantic information.

4 Extra Output Biased Training

The conclusion of the previous section may well be used in training a network when there are many local minima. For instance, training a network for *N-bit parity* gets increasingly harder due to local minima as N gets larger, even if theoretically there is a N-N-1 network for it [RHW 86].

task	ε	α	*topology*	*trials*	*finished*	*avg cyc*	*avg RMS*	*avg correct*
parity-3	0.3	0.9	3-3-1	25	22	2212	0.1	98.6
counter-3	0.3	0.9	3-3-3	25	24	390	.07	99.6
parity-3-counter	0.3	0.9	3-3-4	25	25	217	.07	100
parity-5	0.3	0.9	5-5-1	10	0	10000+	.19	95.7
counter-5	0.1	0.9	5-5-5	10	8	2159	.06	99.2
parity-5-counter	0.1	0.9	5-5-6	10	9	1712	.06	99.7

ε: learning rate; α: momentum; *finished*: number of trials finished 100% correct within the maximum cycle.
avg cyc: average cycle of finished trials; *avg RMS, correct*: average of all trials; *RMS*: Root-Mean-Square error
maximum cycle: 5,000 for 3 bit experiments and 10,000 for the others.

Figure 2: Extra Output Biased Training

Input (40 units)		*Output (20 units)*				*Extra (16 units)*			
		agent	*action*	*object*	*instrument*				
Mary	fixed the car with hammer	Mary	fixed	car	hammer	Mary	fixed	car	hammer
00011	01000100 ... 00101	00011	...	...	00101	0001	...	...	0001

Figure 3: Sample Training Patterns

However, such a network can be trained more easily by specifying another mapping function relevant to the parity mapping. One such a mapping function is the *counter* which counts the number of 1's in the input. Note that one way to solve the parity problem is to count the number of 1's in the input and decide if it is odd or even. Therefore, it is definitely relevant to the parity problem. Therefore, a network like Figure 1(a) can be constructed for the N-bit parity problem. The output unit p is for the parity while the extra N output units $(c_1, \ldots, c_n)$ work as the counter in which the i-th unit is 1 while the other units are 0 for i 1's in the input. Therefore, the training patterns now have the extra information about the number of 1's.

Figure 2 shows the comparison of the performances for the training 3-bit and 5-bit parity, counter, and parity-counter networks. Note, in both categories of *finished trials* and *average cycles*, training two mapping functions — i.e. parity + counter — outperforms training each of them, which means that the two mapping functions help each other to find a common solution.

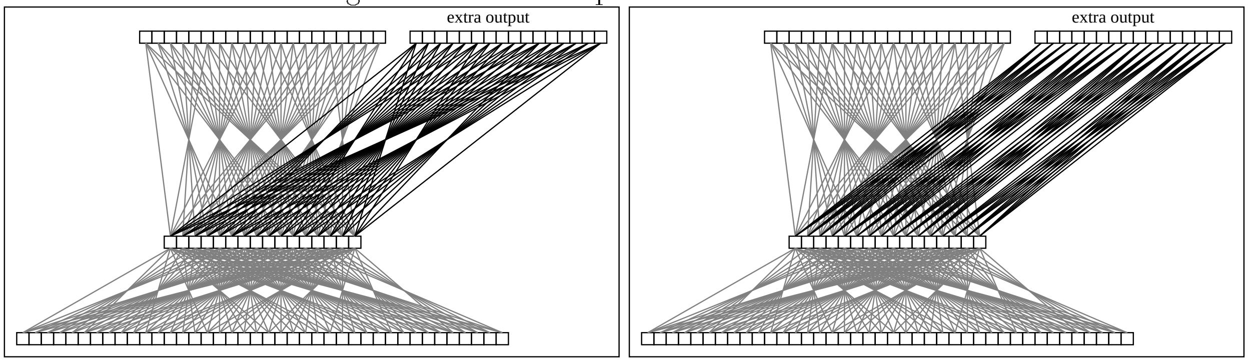
One nice thing about the extra output-biased training is that, after the training, the extra output units can be easily removed without affecting the performance for the original mapping function. For instance, the extra output units, $c_1, \ldots, c_n$, of Figure 1(a) can be removed after the training, leaving the desired network as in Figure 1(b) for the N-bit parity problem.

5 Extra Output Biased Generalization

In training a network and making a set of training patterns, a teacher often has a desired mapping function in mind. As mentioned earlier, a partial specification for the desired

task	ε	α	*topology*	*trials*	*finished*	*avg cyc*	*avg RMS*	*avg correct*	*generalztn*
case-dist	0.1	0.9	40-16-20	5	5	77	.03	100	84.0
case-dist-local	0.1	0.9	40-16-36	5	5	85	.03	100	89.4

Figure 4: Extra Output Biased Generalization



(a) Full Connections for Extra output (b) Partial Connections for Extra Output

Due to the readability, not all the connections are shown.

Figure 5: Networks for Case Analysis

mapping function may result in a mapping function very different from the desired one, since the network has the freedom of choosing any mapping functions that meet the partial specification. In this sense, the terms, *wrong/correct* generalizations, are more revealing than the commonly-used terms, *poor/good* generalization. Traditionally, the only way to get the correct generalization is by covering the input space as much as possible, sometimes more than 95%, like [Allen 87], which makes the generalization task somewhat trivial.

The technique described in the previous section also can be used for obtaining the correct generalization. That is, by giving *direct* information about the desired mapping function as extra outputs, the chance of computing the correct generalization is increased even with the same number of training patterns.

To illustrate this point, a natural language understanding task was used. The inputs are simple English sentences such as "Mary fixed the car with hammer" The outputs are divided into four fields; *agent, action, object,* and *instrument.* Each field takes a value from the input sentence. Through a set of simple grammar rules and lexicon, 360 sentences are generated. Each word is represented by 5 units in a *distributed encoding* scheme [Kukich 88]. The longest sentence has 8 words and, therefore, the input requires 40 units while the output requires $5 \times 4 = 20$ units. 25% of all the sentences (i.e. 90) are used in training and the remaining 75% (i.e. 270) are reserved as novel inputs.

The first row of the table in Figure 4 shows the performances when 16 hidden units are used without any extra output. It was trained up to the point that it produces correct outputs for all the training patterns and for 84% of novel patterns.

To get a better generalization by using extra output units, another set of outputs which are relevant to the desired mapping function should be chosen. In our experiment, instead of finding a wholly new set, a variation of the original output — the values for the four case

agent	*action*	*object*	*instrument*
Mary (0 0 0 1)	fixed (0 0 0 1)	car (0 0 0 1)	hammer (0 0 0 1)
George (0 0 1 0)	cleaned (0 0 1 0)	room (0 0 1 0)	vacuum (0 0 1 0)
Bill (0 1 0 0)	chopped (0 1 0 0)	wood (0 1 0 0)	sponge (0 1 0 0)
Sally (1 0 0 0)	washed (1 0 0 0)	table (1 0 0 0)	ax (1 0 0 0)

Figure 6: Semi-Local Encoding of Values

fields — is used. While the values in the original output are encoded by using a distributed encoding scheme, the variation uses a *semi-local* encoding scheme [Kukich 88] in which, for each field, the possible values for it are encoded by a *local encoding* scheme like the example in Figure 6. Note that, since there are only four possible values for each field, each field requires 4 units.

In the first experiment in which all the extra output units were fully connected to the hidden units as in Figure 5(a), there was no improvement in generalization. This is because the additional information does not restrict the network much more than the original output. Therefore, to get a tighter restriction, the extra output units were connected to hidden units as in Figure 5(b). Note that, the extra output units for a field are exclusively connected to the same number of hidden units. This divides the hidden layer into four parts each of which is mainly concerned with a field in the output. The second row of Figure 4 shows the performance for this network. The generalization is about 89%, a 5% improvement over the network without the extra output.

6 Conclusions

A method for improving back-propagation based training by augmenting the output patterns with additional relevant information is presented. We account for the method's success by arguing that the augmented output provides additional constraints that more precisely specify the allowable function. This results in faster training and better generalization.

Of course, it may not always be possible to provide another mapping function relevant to the desired mapping function. However, even very simple information sometimes results in improvement in training and generalization. Improvement depends on the size of the intersection of the two classes of possible mapping functions. In so far as the intersection is not empty, performance may improve.

An empirical advantage of the "extra output" technique is that after a network has been trained to realize a desired function, the extra output units may be detached. The resulting network is a version that computes more rapidly in that it has fewer connections to manipulate.

References

[Allen 87] R. Allen, Several Studies on Natural Language and Back-Propagation, *IEEE International Conference on Neural Networks*, 1987.

[Kukich 88] K. Kukich, Variations on a Back-Propagation Name Recognition Net, *Advanced Technology Conference*, 1988.

[Nilsson 65] N.J. Nilsson, *Learning Machines: Foundations of Trainable Pattern Classifying Systems*, McGraw-Hill, New York, 1965.

[RHW 86] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning Representations by Error Propagation. In D.E. Rumelhart, J.L. McClelland and the PDP Research Group (Eds.), *Parallel Distributed Processing*, Vol 1, pp. 318-362, MIT Press, 1986.

[Touretzky 89] D.S. Touretzky, Connectionism and Compositional Semantics, Technical Report, CMU-CS-89-147, Carnegie Mellon University, 1989.