Improving Network Generalization through Selection of Examples

Leonardo Franco and Sergio A. Cannas
Facultad de Matemática, Astronomía y Física. Universidad Nacional de Córdoba.
Ciudad Universitaria. (5000). Córdoba. Argentina.
email: {franco,cannas}@fis.uncor.edu

Keywords: Neural Networks, Machine Learning, Generalization.

Abstract

In this work, we study how the selection of examples affects the learning procedure in a neural network and its relationship with the complexity of the function under study and its architecture.

We focus on three different problems: parity, addition of two numbers and bit-shifting implemented on feed-forward Neural Networks.

For the parity problem, one of the most used problems for testing learning algorithms, we obtain the result that only the use of the whole set of examples assures global learning. For the others two functions we show that generalization can be considerably improved with a particular selection of examples instead of a random one.

Improving Network Generalization through Selection of Examples

Introduction

Feed-Forward Neural Networks have been extensively used to solve many kinds of different problems, being applied in a wide range of areas covering subjects such as Prediction of temporal series, Structure prediction of proteins, Speech recognition, etc. [3,4]

One of the fundamental properties making these networks useful is its capacity of learning from examples. Through synaptic modifications algorithms, the network is capable of obtaining a new structure of internal connections, appropiated to solve a determined task.

At present, the general underlying theory of the whole learning process is very poorly understood existing very few general results, especially in the subject concerning generalization.

In this work, we study how the selection of examples affects the learning procedure, leading to an improvement in the generalization ability in most of the cases. We also give some insights about the relationship between the complexity of the function and its architecture, related to the generalization ability.

We study three different problems: addition of two numbers, bit-shifting and parity implemented on feed-forward Neural Networks, starting from small architectures and then evolving to greater sizes.

Since all the problems are non-linearly separable, they are all optimal in depth, having only one hidden layer. The parity network used is taken from [6] and the others two architectures are taken from [1,2].

1 The Sum problem

We study the generalization properties of a network constructed to compute the addition between two binaries operands each of N bits, which gives a result of the same length N. The architecture is optimal in depth, i.e. it has only one hidden layer [1]. It is composed by 2N binary neurons in the input layer corresponding to the two numbers of N-bits to add, N hidden neurons and N binary neurons in the output. To simplify the analysis and to allow comparisons with every function that has at least one output neuron, we study only one output bit, the one having the most significant value. The generalization to the case of N output bits is straightforward, since every output neuron behaves indenpendently. The resulting network has two hidden neurons, one of them fully connected to the input layer, while the other is connected to every input neuron but those corresponding to the most significant bits. Fully connection also exists between the hidden layer and the output neuron. Finally, the two most significant input bits are also directly connected to the output. Further simplification of the problem can be obtained by considering shared synapsis. That means, we impose the constraint that synapsis connecting one hidden neuron

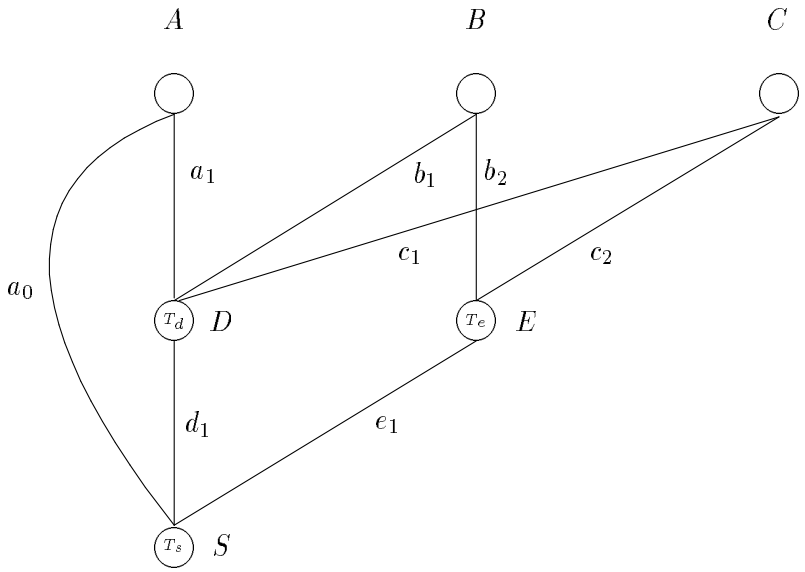

Figure 1: Network Structure to compute the most significant output bit of the Sum function of two 3-bit numbers. The synapses corresponding to input bits with same significant value have been symmetrized in such a way that the six input bits can be replaced by three input bits taking the values {0,1,2}.

with two input bits with the same significant value are always equal. With this constraint every pair of input neurons corresponding to the same significant bit can be replaced by a single ternary neuron, which take the values {0,1,2}. Hence, our final architecture contains an input layer of N ternary neurons. We start studying the case of adding two numbers of 3-bit length and the we generalize the results to the case of addition of N-bits numbers. A description of the network architecture for N=3 is depicted in figure 1.

The output bit S receives direct inputs from the 2 hidden neurons $D, E = 0, 1$ and from the input neuron A computing the following function:

$$S = \theta\{a_0 A + d_1\theta[a_1 A + b_1 B + c_1 C - T_d] + e_1\theta[b_2 B + c_2 C - T_e] - T_s\} \qquad (1)$$

where $A, B, C = 0, 1, 2$; $\theta(x)$ is the Heaviside step function and T_α ($\alpha = d, e, s$) are the thresholds parameters.

The values of synapse a_0 and those of the thresholds T_d, T_e are restricted to be greater than zero to reduce the possible internal representations to only one.

Requiring that the network computes the full set of 27 addition examples lead to the following necessary and sufficient conditions:

$$T_d > \quad a_1 \quad \geq \frac{T_d}{2} \qquad (2)$$

$$a_1 + 2b_1 \geq T_d > 2b_1 + 2c_1 \quad (3)$$
$$a_1 + b_1 + 2c_1 \geq T_d > a_1 + b_1 + c_1 \quad (4)$$
$$2b_2 \geq T_e > 2c_2 \quad (5)$$
$$b_2 + 2c_2 \geq T_e > b_2 + c_2 \quad (6)$$
$$e_1 \geq T_s > 0 \quad (7)$$
$$a_0 \geq T_s > 2a_0 + d_1 \quad (8)$$
$$2a_0 + d_1 + e_1 \geq T_s > a_0 + d_1 + e_1 \quad (9)$$

We will show that, if the network just computes a particular set of 12 examples, then the above conditions are satisfied. Therefore, the correct learning of such examples ensures full generalization with less than half of the total number of examples.

We will denote the examples by writing between square brackets the three input values and the correct output separated by a colon.

- From the example [000 : 0] we obtain the right part of Eq. (7).
- From the examples [100 : 1] and [200 : 0] we obtain:

  $$d_1 < -a_0 \quad (10)$$

  and the fulfillment of Eqs. (2) and (8)
- From the example [020 : 1] we obtain:

  $$d_1\theta[2b_1 - T_d] + e_1\theta[2b_2 - T_e] > T_s$$

  which together with $T_s > 0$ and Eq. (10) implies the left part of Eq.(7) and the left part of Eq.[5].
- From the example [120 : 0] we obtain the left part of Eq.(3) together with the right part of Eq.(9).
- From the examples [111 : 1] and [112 : 0] we obtain Eq.[4] and also that $c_1 > 0$
- From the examples [011 : 0], [012 : 1] and [002 : 0] we obtain Eq.[6] together with the right part of Eq.[5].
- From the example [222 : 1] we obtain the left part of Eq. [9].
- Finally, from the example [022 : 1] we obtain the right part of Eq. (3) and consequently the fulfillment of the complete set of Eqs. (2-9).

For the most general case of an input of N bits, we can see that, for any bit we increase the input in size, it is enough to add four examples to obtain generalization. For example, for the case of N=4 we take the twelve examples corresponding to the 3-bit case but now converted to the new problem by adding a zero in the new right-most input place. For example, the input pattern [112:0] in the 3-bit problem now would be the input pattern [1120:0]. It is also necessary to add 4 new examples, two pairs of examples having different output when we modify the new right-most input

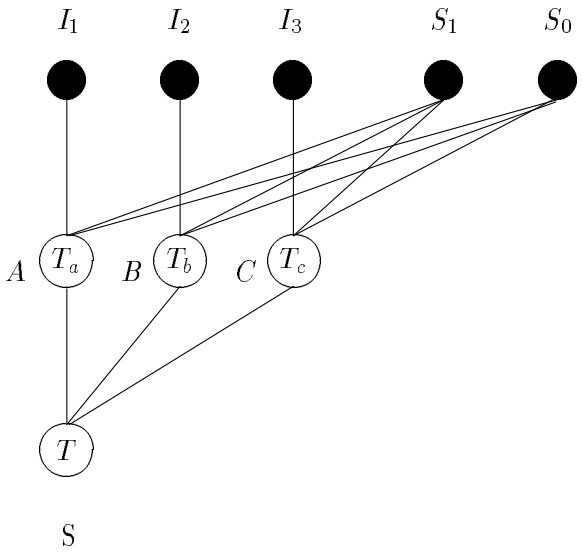

Figure 2: Network Structure to compute the left-most output bit of a bit shifting operation.

bit. For the case of N=4 these two pairs are the examples {[0111:0] [0112:1];[1111:1] [1112:0]}. The same procedure is repeated as we increase the number of input bits. Thus, we obtain an upper bound for the minimal set of examples equal to $4N$. Hence, for N output neurons such bound becomes $2N(N+1)$

2 Bit Shifting

The bit-shifting function is a basic operation in computers circuits and also was used in modelling vision devices (see Ref.[2] and references therein). The network structure we considered here was derived in Ref.[2]. Since it has only one hidden layer it is optimal in depth. The input layer has the N input bits to shift plus $log_2(N+1)$ indicating bits given the places to shift. There are N binary neurons in the hidden layer, every one connected to one input neuron and to all the indicating neurons. The problem has an output of N bits but, to simplify the study, we analyze just one output bit. We start the analysis with the particular case of having 3 input bits with its corresponding two indicating bits, to later generalize the results for the case of N input bits.

For the case of 3 input bits, the structure of the network is shown in figure 2, where the output neuron S computes the following function:

$$S = \theta[J_a\theta(a_1 I_1 + a_2 S_1 + a_3 S_0 - T_a) + J_b\theta(b_1 I_2 + b_2 S_1 + b_3 S_0 - T_b) + \\ + J_c\theta(c_1 I_3 + c_2 S_1 + c_3 S_0 - T_c) - T] \qquad (11)$$

$I_1, I_2, I_3 = 0, 1$ being the input bits; $S_0, S_1 = 0, 1$ are the two indicating bits, J_a, J_b, J_c are the synapses between the three hidden neurons A, B, C and the output

S, and a_i, b_i, c_i are the synapses between the respectively hidden neurons (A,B,C) and the input and indicating bits.

As in the preceeding section, we impose the constraint that the thresholds of the hidden neurons T_a, T_b, T_c to be always positive. In order to simplify the generalization to the N input bits case we also impose that the synapsis J_a, J_b and J_c has to be greater than the threshold of the output neuron T. These constraints makes no substantial changes to the results.

Requiring that this network computes efficiently the full set of 32 examples lead to the following necesary and sufficient conditions:

$$T > 0 \tag{12}$$
$$a_1 \geq T_a > a_1 + a_3 \tag{13}$$
$$a_1 + a_2 < T_a \tag{14}$$
$$b_1 < T_b \tag{15}$$
$$b_1 + b_3 \geq T_b > b_3 \tag{16}$$
$$b_1 + b_2 + b_3 < T_b \tag{17}$$
$$c_1 < T_c \tag{18}$$
$$c_1 + c_2 \geq T_c > c_2 \tag{19}$$
$$c_1 + c_2 + c_3 < T_c \tag{20}$$

We denote the examples by writing between square brackets the three input values plus the two indicating bits and the correct output separated by a colon.

As in the previous section, it is easy to verify that the correct learning of the ten examples: $\{[000-00:0], [100-00:1], [100-01:0], [100-10:0], [010-01:1], [010-11:0], [010-00:0], [001-00:1], [001-10:1], [001-11:1]\}$ ensures the correct computation of the full set of examples. For instance, from $[000-00:0]$ we derive $T > 0$ (Eq. 10); from $[100-00:1]$ we obtain the left-side of Eq.(11), and so on.

Extending this result to the case of having N bits involves all the examples that have only one Input bit ON and all its possibles combinations of indicating bits, plus the patterns with all the input bits OFF and its combinations with the indicating bits. This procedure gives that for N input bits is enough $(N+1)^2$ examples to obtain generalization.

3 The Parity Problem

The parity function is one of the most used problems for testing learning algorithms because its simple definition and its great complexity given by the fact that the most similar patterns (those differing by a single bit) have different outputs. The parity function has only one output neuron that indicates when it is ON that an odd number of the N input bits are ON, while it is OFF if this number is even. The simplest architecture known to compute this function using linear threshold units (l.t.u.) and having no direct input-output connections consists of a network with one hidden layer with N units, fully connected to the N input neurons (See Ref.[5]).

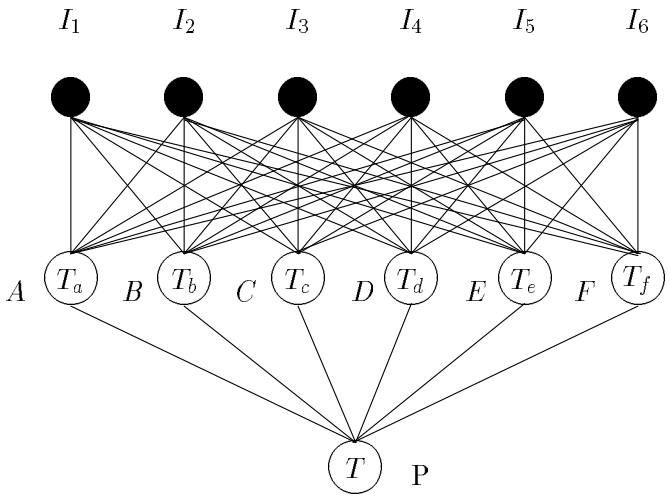

Figure 3: Network Structure to compute the 6-bit Parity function.

As we have done in the previous sections we first analyze a particular case with 6 input neurons (see Fig.3) and then we generalize the result to N bits.

We start explaining the functioning of the network in terms of the synapses values. The functioning is mainly determined by the behaviour of the six hidden neurons. Half plus one of these neurons 4 are ON if the input pattern has an odd number of input bits equal to one; otherwise, only half of them are ON.

This is achieved by the following conditions:

1. The hidden neuron A, has to be ON when one or more input bits are ON, otherwise A has to be OFF.

2. The hidden neuron B, has to be OFF when two or more input bits are ON, otherwise B has to be ON.

3. The hidden neuron C, has to be ON when three or more input bits are ON , otherwise C has to be OFF.

4. The hidden neuron D, has to be OFF when four or more input bits are ON, otherwise D has to be ON.

5. The hidden neuron E, has to be ON when five or more input bits are ON, otherwise E has to be OFF.

6. The hidden neuron F, has to be OFF when the six input bits are ON, otherwise F has to be ON.

7. Finally, the output neuron P, has to be ON when four or more hidden neurons are ON, otherwise P has to be OFF.

Denoting by $P = 0, 1$ the output value, we have that:

$$P = \theta \left\{ a\theta \left[\sum_{i=1}^{6} (a_i I_i - T_a) \right] + b\theta \left[\sum_{i=1}^{6} (b_i I_i - T_b) \right] + c\theta \left[\sum_{i=1}^{6} (c_i I_i - T_c) \right] + \right.$$

$$+d\theta\left[\sum_{i=1}^{6}(d_iI_i-T_d)\right]+e\theta\left[\sum_{i=1}^{6}(e_iI_i-T_e)\right]+f\theta\left[\sum_{i=1}^{6}(f_iI_i-T_f)\right]-T\Bigg\} \tag{21}$$

where $I_i=0,1(i=1,\ldots,6)$ denote the state of the input neurons.
From the conditions (1-7) we obtain the following set of inequalities:

From condition 1 we obtain that:

$$a_i \geq T_a \ \ \forall i \tag{22}$$

From condition 2 we obtain that:

$$b_i + b_j < T_b \ \ \forall\ i,j \tag{23}$$

From condition 3 we obtain that:

$$c_i + c_j + c_k \geq T_c \ \ \forall\ i,j,k \tag{24}$$

From condition 4 we obtain that:

$$d_i + d_j + d_k + d_l < T_d \ \ \forall\ i,j,k,l \tag{25}$$

From condition 5 we obtain that:

$$e_i + e_j + e_k + e_l + e_m \geq T_e \ \ \forall\ i,j,k,l,m \tag{26}$$

From condition 6 we obtain that:

$$f_i + f_j + f_k + f_l + f_m + f_n < T_f \ \ \forall\ i,j,k,l,m,n \tag{27}$$

From condition 7 we obtain that:

$$w + x + y + z \geq T \tag{28}$$

where w,x,y,z are different and could be any of the synapses (a,b,c,d,e,f).

Fixing the values of the thresholds T_a,T_b,T_c,T_d,T_e,T_f, Eqs. (22-28) lead to a set of 2^6-1 *independent* inequalities for the synapsis parameters $\{a_i\},\{b_i\},\ldots,\{f_i\}$. On the other hand, the total number of examples is 2^6. Therefore, only imposing the correct learning of the full set of examples except the most simple one [000000 : 1], which determines the sign of the output threshold, ensures the fulfillment of inequalities (22-28). Hence, we cannot guarantee generalization with any particular subset of examples.

Finally, the generalization of this result to a fully connected network with N inputs and N hidden neurons is straightforward, because in this case we have 2^N examples and 2^N-1 inequalities of the type of Eqs. (22-28).

	Addition	Bit-Shifting	Parity
Number of Synapsis	$2N+4$	$N(2+log_2(N+1))$	N^2+N
Total Number of Examples	3^N	$2^N(N+1)$	2^N
Examples Needed for Generalization	$4N$	$(N+1)^2$	2^N-1

Table 1: Features of the networks used to compute the Addition of two Numbers, Bit-Shifting and the N-bit Parity functions.

Conclusions

We analyzed neural network architectures that exactly solve three different problems: addition, bit shifting and parity. All these architectures have in common one hidden layer and only one output neuron.

We showed how several properties for networks with an arbitrary number of inputs N can be simply obtained by generalizing the analysis of the small network equations. In this way, several properties can be exactly obtained for N arbitrary. They are compared in Table 1.

For the addition and bit shifting problems we showed that generalization can be greatly improved with a particular selection of examples, compared with a random one. We also found how an upper bound for the minimal number of examples needed for full generalization scales with N.

In the case of the parity problem we showed that the most frequently used architecture (i.e., the full connected one with N hidden neurons) may be completely inadequate for studying generalization, since there is no subset of examples that guarantees it. This property may be considered a pathological one resulting from the use of a very simple architecture (only one hidden layer) in a very complex problem. Probably this pathology may disappear in networks with more complex architectures.

Finally, a comparison of the different properties shown in table 1 for the three cases suggests that the minimum number of examples needed for full generalization may serve as a criterion for defining the complexity of a problem.

Acknowledgments

This work was partially supported by the following agencies: CONICET (Argentina), CONICOR (Córdoba, Argentina) and Secretaría de Ciencia y Técnica de la Universidad Nacional de Córdoba (Argentina).

References

1. Cannas, S.A. 1995. Arithmetic Perceptrons, *Neural Computation* **7** (1) 173-181.

2. Franco, L. and Cannas, S.A. 1998. Solving arithmetic problems using feed-forward neural networks, *Neurocomputing* **18**, 61-79.

3. Haykin, S. 1994. *Neural Networks: A comprehensive foundation.* McMillan.

4. Hertz, J., Krogh, A., and R. Palmer. 1991.*Introduction to the Theory of Neural Computation*, Addison Wesley, Santa fe Institute.

5. Impagliazzo, R., Paturi R., and Saks, M.E. 1997. Size-Depth Trade-offs for Threshold Circuits. *SIAM Journal on Computing* **26** (3), 693-707.

6. Rumelhart, D.E., McClelland, J.L. 1986. *Parallel Distributed Processing*, MIT Press, Cambridge, MA.