# Solving arithmetic problems using feed-forward neural networks.

**Leonardo Franco** [1]
**Sergio A. Cannas** [2]
*Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba,*
*Haya de la Torre y Medina Allende S/N, Ciudad Universitaria,*
*5000 Córdoba, Argentina.*
*e-mail : {franco, cannas}@.fis.uncor.edu*

**Abstract**

We design new feed-forward multi-layered neural networks which perform different elementary arithmetic operations, such as bit shifting, addition of $N$ $p$-bit numbers, and multiplication of two $n$-bit numbers. All the structures are optimal in depth and are polinomialy bounded in the number of neurons and in the number of synapses. The whole set of synaptic couplings and thresholds are obtained exactly.

*Keywords:* Neural Networks, Arithmetic operations, Shifter Circuit.

[1] Fellow of the Consejo Provincial de Investigaciones Científicas y Tecnológicas de la provincia de Córdoba. (CONICOR-Argentina).

[2] Member of the National Research Council. (CONICET-Argentina).

## 1 Introduction

One of the most important issues in the Neural Network field is *learning* [15], *i.e.*, synaptic modification algorithms (also known as learning algorithms) that allow an arbitrarily connected network to develop an internal structure appropriated for a particular task. Although learning algorithms can be (in principle) implemented for any kind of neural structure, feed-forward layered networks presents a simple architecture that makes them specially suitable for the study of general learning properties (besides their utility for solving specifical problems, see, for instance, [9]). These networks are basically input-output devices: parallel circuits composed by layers of processing units (neurons) connected by synaptic couplings; the information flows in only one direction from the input to the output layer and eventually passes through one or more hidden layers. The simplest feed-forward network is the so-called *simple perceptron* [12], which possesses only two layers: input and output. Learning in feed-forward networks is achieved by minimizing some cost function in order to associate a set of inputs (stimulus) to a set of desired outputs (responses).

In this work we consider networks composed of binary neurons, *i.e*, neurons whose activity can only take two values: zero and one. The activity of a neuron $\sigma_i$ in a given layer is determined by computing a linear threshold function of the form

$$\sigma_i = \Theta\left[\sum_j W_{ij} S_j - T_i\right] \tag{1}$$

where $\Theta(x)$ is 1 if $x \geq 0$ and 0 otherwise, $W_{ij}$ are the synaptic weights, $S_j = 0, 1$ are the activities of the neurons in a previous layer and $T_i$ is the activation threshold of the neuron $\sigma_i$. In this case, every output neuron computes a boolean function of the input values.

The first problem one faces when trying to apply a learning algorithm to a specific problem is that of solvability. There exists a large class of problems (the so-called *linearly inseparable*) that cannot be solved by simple perceptrons, being the *exclusive-or* (XOR) boolean function the most famous one [5]. Arithmetic problems, the topic of the present work, also fall into this category. On the other hand, linearly inseparable problems can always be solved by adding hidden layers. Moreover, it can be easily shown that any boolean function can be computed using only one hidden layer with a number of neurons which grows exponentially with the number of variables (or input neurons). However, this exponential dependency is highly undesirable in practical implementations. Hence, the important point is to determine the minimum number of hidden layers **polinomially bounded in the number of entries** needed to solve a given function. Let us define the depth of a network as the number of layers with threshold units, *i.e.*, the input layer is not taken into account. The minimal depths for some arithmetic problems are known. It was proven by A. Hajnal *et al* [4] that the addition of $N$ numbers can be computed by networks of depth 2, while multiplication of two numbers needs at least a depth 3 network. In fact, both problems are closely related and after solving the former a solution of the latter can be immediately obtained, as we will show in this work. Generally speaking, arithmetic problems, if soluble by feed-forward neural networks, admit several solutions with different architectures, and for some of them analytical solutions have been obtained (see [8,6,3]). Besides the depth, another important factor to take into account in the choice of a given architecture is the maximum fan-in of the circuit, which is defined as the maximum number of inputs of a single neuron. Both features (depth and fan-in) are very important by the time of chip implementation. The depth of the network gives the overall delay in computation (which is also important for parallel processing software) and the hardware puts an upper limit to the number of inputs of a single gate.

Returning to the general topic of learning in multilayer networks, several optimization algorithms have been introduced in the last years but, up to now, no general convergence theorem has been proved for any of them. Moreover, a current problem in the field is the search of standards for performance comparison of different learning algorithms [11].

Even in cases in which the algorithms are successful, their general underlying theory still remains far from being understood and several important problems remain open. Which is the minimum number of hidden neurons needed to perform a certain task? Which is the best architecture for the hidden layers? How many examples are needed for learning certain input-output mapping and how does the network performance depend on the choice of a particular set of training examples?

Having exact solutions of particular networks that solve non-trivial problems like the arithmetic ones aid significantly for the analysis of such type of questions. They also serve as standards for testing how the different algorithms work. Moreover, statistical mechanics has often illustrated that exactly soluble models could also reveal unexpected new facts.

In this work we obtain analytically soluble neural networks for three arithmetic problems, namely, addition of $N$ numbers of $p$-bit length, multiplication of two $n$-bit numbers and bit-shifting of a $M$-bit number. All these structures are optimal in depth and we analyze their structures in terms of size, connections, maximum fan-in, etc. The results are also compared with other implementations.

The bit-shifting problem has an additional interest. Shifter circuits have been proposed to explain dynamic control of information flow between arrays of neurons at different levels of the visual pathway, related to distinct aspects of perception, such as directed visual attention, stereopsis, and others [2]. Hence, besides its possible utility in neurophysiological modeling, we think that the present network may serve to design efficient visual processing networks, since, to the best of our knowledge, this is the first implementation of a shifter network of depth-2 **for an arbitrary shifting range**. By the way, this is a constructive proof that the lower bound for the depth of this problem is two (it is a linearly inseparable problem).

The paper is organized as follows: in section 2 we present the construction of a depth-2 network for addition of $N$ numbers; these results are applied in section 3 to the construction of a depth-3 network for multiplication of two numbers. In section 4 we construct a network which shifts an arbitrary number of places all the bits of a number of length $M$. This structure is straightforwardly generalized for performing both left and right shifts. Some conclusions and remarks are presented in section 5.

## 2 A depth-2 network for addition of $N$ numbers.

It was proven by Hajnal et Al. [4] that the addition of $N$ numbers can be computed by polynomial size Neural Networks of depth 2. The proof is nonconstructive and up to our knowledge no such network has been constructed. Recently Siu & Roychowdhury [14] showed that a depth 2 net is feasible. We have found a depth-2 neural network composed by linear threshold units, polynomialy bounded in the number of gates and connections, for solving the problem of adding $N$ numbers of $p$-bits length. This problem is very closely related to multiplication, powering and other ones.

The main problem we face in designing such net is the difficulty introduced by the carry from the sum of the bits. In almost all the previous designs the procedure, mainly, consists of transforming the sum of the $N$ numbers to the sum of two numbers, using techniques as the Dortmunder method [6], Carry Save Addition [10] and Block Save Addition [8]. These procedures introduce, at least, 2 intermediate layers of linear threshold units leading to structures that exceed the known lower bound. Hence, we decide not to use those techniques and proceed with the sum of the numbers in just one step.

The network architecture is the following: In the input layer there are $Np$ neurons corresponding to the $N$ $p$-bit numbers to be added. The output layer has $M = \log_2(N(2^p - 1)+1)$ neurons, enough to express the result from the sum that could be, at most, $N(2^p-1)$. The log operation is assumed to give an integer value result, which is greater or equal than the exact real result.

Every input number $S_j$ ($j = 1, \ldots, N$) will be represented by a set of $p$ binary neurons: $S_j = S_j^{p-1}, S_j^{p-2}, \ldots, S_j^0$. The input layer is arranged in $p$ groups of $N$ bits each, depending on its significant value, i.e., the corresponding neurons are arranged in the following order: $S_N^{p-1}, S_{N-1}^{p-1}, \ldots, S_1^{p-1}, S_N^{p-2}, S_{N-1}^{p-2}, \ldots, S_1^{p-2}, \ldots, S_N^0, S_{N-1}^0, \ldots, S_1^0$. The significant value of a bit $S_N^p$ is $2^p$ and it is assumed that the neurons could take the values one and zero, i.e., $S_j^i = \{0, 1\}$. The output neurons are denoted by $S_{M-1}^o, S_{M-2}^o, \ldots, S_0^o$.

In this way the total number of neurons will be completely determined by the number of neurons in the intermediate layer, because the size of the input and output layers is clearly defined by the problem itself. The neurons in the intermediate layer will be arranged in $M$ groups. The group $k$ will have $N_k$ neurons, whose outputs are connected only to the output neuron $S_k^o$.

We analyze every output bit separately. Let us first consider the synapses structure which determines the least significant output bit $S_0^o$ (corresponding to the value $2^0$). We shall pay a special attention to this case, since this structure will be essentially repeated with minor changes for the other bits.

The least significant bit, namely $S_0^o$, has to be one if the result of the sum H of the input bits

$$H = \sum_{i=1}^{N} S_i^0 \tag{2}$$

is odd, and zero if it is even:

$$S_0^o = \begin{cases} 1 & \text{if H is odd} \\ \\ 0 & \text{if H is even} \end{cases} \tag{3}$$

In other words, this bit computes the parity of $H$. The solution of this problem will be the key to compute the sum of $N$ numbers. It can be solved as follows: first, we put in this part of the intermediate layer (*group* $k = 0$) $N_0 = N$ neurons $\sigma_i^0 = 0, 1$ ($i = 0, ..., N_0 - 1$). The synaptic connections between these neurons and the $k = 0$ input ones are chosen in such way that more than half of the intermediate neurons are ON (that means $\sigma_i^0 = 1$) when $H$ is odd while less than half of them are OFF ($\sigma_i^0 = 0$) when $H$ is even.

How should we select the value of the connections? Since in the parity problem all the bits have the same weight, it is reasonable to set all the connections from the input neurons to an intermediate one $\sigma_i^0$ equal to a single value $w_i^{0,0}$. Therefore, we have that

$$\sigma_i^0 = \Theta\left( w_i^{0,0} \sum_{j=1}^{N} S_j^0 - T_i \right) = \Theta\left( w_i^{0,0} H - T_i \right). \tag{4}$$

Now we select the following values:

$$\{T_i\} = \left\{ \pm 1, \pm 3, \pm 5, ...., \begin{array}{ll} \pm(N_0-2), N_0 & \text{if } N_0 \text{ is odd} \\ \pm(N_0-1) & \text{if } N_0 \text{ is even} \end{array} \right\} \tag{5}$$

and

$$w_i^{0,0} = \begin{cases} 1 & \text{if } T_i > 0 \\ -1 & \text{if } T_i < 0 \end{cases} \tag{6}$$

Notice that, for even values of $H$, the intermediate neurons whose thresholds have the same absolute value $|T_i|$ are always one ON and the other OFF. This is also true when $H$ is odd for all intermediate neurons except those with $|T_i| = H$, which are **both ON**. Moreover, for odd values of $N$ the neuron with $T_i = N$ is OFF for all even values of $H$. Therefore, we have that

$$\sum_{i=0}^{N_0-1} \sigma_i^0 = \begin{cases} > N/2 & \text{if } H \text{ is odd} \\ \leq N/2 & \text{if } H \text{ is even} \end{cases} \tag{7}$$

for all values of $N$, as desired (see table 1). Finally, we can see from table 1 that setting the synapses from the intermediate group ($k = 0$) toward the output neuron $S_0^o$ all equal to one, i.e., $S_0^o = \Theta\left(\sum_{i=0}^{N_0-1} \sigma_i^0 - T_0\right)$, and taking

$$N_0/2 < T_0 \leq (N_0+1)/2, \tag{8}$$

$S_0^o$ computes the parity problem for all values of $N$. In Fig. 1 we show an example of the resulting net structure for the least significant output bit.

The solution we have found is a particular and simple one. A more general solution can be obtained from the previous one by allowing the thresholds $\{T_i\}$ to take a wider range of values. Following the same ideas as before, any threshold that in the previous solution took the value $m$ ($m = \pm 1, \pm 3, \ldots$) , now is allowed to take an arbitrary value in the interval $(m-1, m]$. For instance, the threshold that was $T_i = 1$ now can take the values in the interval $(0, 1]$ and the one that was $T_i = -1$ now can be in the interval $(-2, -1]$. All the other values of thresholds and synapses remain as before.

Having computed the least significant output bit, we consider the other output bits. To compute the second least significant output bit $S_1^o$, we have to perform the addition of the $N$ input bits belonging to the group $k = 1$ (corresponding to the significant value $2^1$) plus the carry from the $N$ input bits of the group k=0 (significant value $2^0$). We proceed in the same manner as before, but considering that two bits of significant value $2^0$ are necessary to form one with significant value $2^1$. Hence, the bits of significant value $2^0$ are connected to the intermediate neurons by synapses that are half of the value of the synapses that connect these neurons with the corresponding ones with absolute value $2^1$.

Now, we proceed as if we were summing $N_1 = N + int(N/2)$ bits (therefore we have to put this number of intermediate neurons) and we repeat the procedure we used in the first case (the *int* operation takes the integer part of the argument).

Therefore, we have that

$$\sigma_i^1 = \Theta\left[w_i^{1,1} \sum_{j=1}^{N} S_j^1 + w_i^{1,0} \sum_{j=1}^{N} S_j^0 - T_i\right] \tag{9}$$

for $i = 0, \ldots, N_1 - 1$, with

$$\{T_i\} = \left\{ \pm 1, \pm 3, \pm 5, ...., \begin{array}{ll} \pm(N_1 - 2), N_1 & \text{if } N_1 \text{ is odd} \\ \pm(N_1 - 1) & \text{if } N_1 \text{ is even} \end{array} \right\} \tag{10}$$

and

$$w_i^{1,j} = \left\{ \begin{array}{ll} \frac{1}{2^{1-j}} & \text{if } T_i > 0 \\ -\frac{1}{2^{1-j}} & \text{if } T_i < 0 \end{array} \right. \quad (j = 0, 1) \tag{11}$$

This procedure can be further generalized to any output bit. When $k < p$, the output bit $S_k^o$ results from the addition of the $N$ input bits of the group $k$, plus the carrying from the preceding $k - 1$ input groups. This operation is equivalent to the addition of $N_k$ numbers, where

$$\begin{aligned} N_k &= Int\left(N + N/2 + \cdots + N/2^k\right) = \\ &= 2N - Int\left[\frac{N}{2^k}\right] \end{aligned}$$

for $k < p$. On the other hand, when $p \leq k \leq M - 1$, the output bit $S_k^o$ results only from the carrying of the $p$ input groups, because there are no input groups with $k' \geq p$. Therefore, the number of intermediate neurons in the group $k$ will be

$$N_k = \left\{ \begin{array}{ll} 2N - Int\left[\frac{N}{2^k}\right] & \text{if } k < p \\ \frac{N_p}{2^{k-p+1}} = Int\left[\frac{N}{2^{k-p}}(1 - \frac{1}{2^{p+1}})\right] & \text{if } k \geq p \end{array} \right. \tag{12}$$

and

$$\sigma_i^k = \Theta\left[\sum_{l=0}^{k} w_i^{k,l} \sum_{j=1}^{N} S_j^l - T_i\right] \tag{13}$$

for $i = 0, \ldots, N_k - 1$, with

$$\{T_i\} = \left\{ \pm 1, \pm 3, \pm 5, ...., \begin{array}{ll} \pm(N_k - 2), N_k & \text{if } N_k \text{ is odd} \\ \pm(N_k - 1) & \text{if } N_k \text{ is even} \end{array} \right\} \tag{14}$$

and

$$w_i^{k,l} = \left\{ \begin{array}{ll} \frac{1}{2^{k-l}} & \text{if } T_i > 0 \\ -\frac{1}{2^{k-l}} & \text{if } T_i < 0 \end{array} \right. \quad (l = 0, \ldots, k) \tag{15}$$

The total number of neurons in the intermediate layer results:

$$N_I = \sum_{k=0}^{p-1} \left\{ 2N - Int\left[\frac{N}{2^k}\right] \right\} + \sum_{k=p}^{M-1} Int\left[\frac{N}{2^{k-p}}(1 - \frac{1}{2^{p+1}})\right] \simeq$$

$$\simeq 2Np - 2 + Int\left[\frac{N}{2^p} - \frac{1}{2^p - 1}\right] \tag{16}$$

Finally, we set

$$S_k^o = \Theta(\sum_{i=0}^{N_k - 1} \sigma_i^k - T_0^k) \tag{17}$$

with

$$N_k/2 < T_0^k \leq (N_k + 1)/2 \tag{18}$$

for $k = 0, \ldots, M - 1$.

The whole structure of a net for summing $N$ numbers of $p$ bits involves $\mathcal{O}(3Np)$ neurons and a number of synapses $\mathcal{O}(N^2p^2)$.

As seen, all the sum procedure could be carried out just knowing how to compute the **parity** problem.

The full structure of a net for the particular case $N = 3$ $p = 2$ is shown in figure 2. Some network parameters for different values of $N$ and $p$ are shown in table 2.

## 3 Depth-3 Neural Multiplier

The two $n$-bit number multiplier consists of a 4-layer neural network. The multiplication is performed in two transformation steps. The first one transforms the product of the multiplicand and the multiplier to a sum of $n$ numbers of $2n$-1 bit length. The second transformation adds all these $n$ numbers.

We begin by arranging the two $n$-bit numbers in the input layer.

The first transformation step consists in multiplying the bits of the multiplicand by every bit of the multiplier, in the same way as in hand calculations. We need no intermediate layer for this process, since it involves binary products, which are equivalent to perform a bit to bit boolean operation AND, which is indeed linearly separable [5]. Hence, the first step can be carried out by a simple perceptron. We obtain $n$ numbers of $2n$-1 bits that have to be added. We arrange them in the first hidden layer. Notice that just $n$ out of the $2n$-1 bits of every number are different from zero. This fact allows us to reduce the number of neurons from $2(n^2 - n)$ down to $n^2$ in this layer. The neurons from this layer are connected to the input by two synapses, one to a bit from the multiplicand and one to the multiplier. The values of the synapses and the thresholds can be chosen in a simple form. It is enough to set all the synaptic weights equal to $+1$ and all the thresholds to $+2$ to compute the above operation. This is a simple particular solution, but a more general form is to put every threshold ($T$) to a positive value and the two correspondent synapses ($W_1, W_2$) such that:

$$W_1, W_2 < T \leq W_1 + W_2. \tag{19}$$

In this way the first transformation step involves $2n$ neurons in the input layer, $n^2$ in the first hidden layer and $2n^2$ synapses between both layers.

The second and last transformation consists in adding the $n$ numbers of $2n$-1 bits, we have obtained from the first step. We have solved this problem in the previous section in a more general form of adding $N$ numbers of $p$ bits. For this case, in which the numbers come from a product, a minimization can be done in the number of neurons, due to the fact that $n$-1 bits of every number are zero. Thus, this second step needs $2n^2 - n$ neurons in the second hidden layer that, together with the $2n$ in the output layer totalizes $3(n^2 + n)$ neurons for the whole net. The number of required connections is of order $n^4$.

An example of the structure of a multiplier for the case of two numbers of 2 bits length is shown in figure 3.

In table 3 some parameters of the network for some classical sizes, as well as for the general case, are shown.

## 4 A bit shifting network

Given an input number of $M$-bits $S = S_{M-1}S_{M-2}....S_1S_0$, with $S_i = \{0,1\}$, a 1-place left bit-shifting consists in obtaining a new $M$-bit number in which all the bits from the input number are carried one place to the left and a zero is put in the remaining empty site (see figure 4). The leftmost bit $S_{M-1}$ is simply thrown away.

Generalizing this procedure, a c-bit shifting consists on the transformation in which we carry all the input digits c places to the left and put zeros in the c empty places from the right, while the c bits $S_{M-1}, \ldots, S_{M-c}$ are discarded.

We constructed a depth-2 network which performs this operation. It is optimal in depth because it can be shown that this is a linearly inseparable problem.

The structure is the following. The input layer contains the $M$ bits of the input number $S$ plus $K = log_2(M+1)$ neurons indicating the number $c$ of places to be shifted ($c \leq M$), *i.e.*, $c = S^c_{K-1} \ldots S^c_1 S^c_0$. The output layer contains just $M$ neurons corresponding to the shifted number $S^o = S^o_{M-1} \ldots S^o_1 S^o_0$. In our model the intermediate layer acts as a set of gates letting pass the correspondent values of the input bits towards the output, depending on the values of the indicating neurons.

This network is straightforwardly generalized to a bi-directional shifter which allows to perform both left and right shift in a single structure.

In order to explain the functioning of the intermediate layer, let us start presenting the simple case of a 1-bit shifting. This structure will be generalized later (with minor changes) to a c-bit shifter.

### 4.1 The 1-bit shifter

For the 1-bit shifting it is necessary to put just one indicating neuron $S^c = S^c_0$. When $S^c_0 = 0$ the output will be equal to the input $S^o_i = S_i \; \forall i$ (no shift is performed), while for $S^c_0 = 1$ we shift all the input bits one place, *i.e.*, $S^o_i = S_{i-1}$ for $i = 1, \ldots, M-1$ and $S^o_0 = 0$. We see that the state of the output neuron $S^o_i$ depends only on the values of $S_i$, $S_{i-1}$ and $S^c_0$. Hence, we put only two intermediate neurons $\sigma_{i,0}$ and $\sigma_{i,1}$ in every group $i$ (see Fig.5) for $i = 1, \ldots, M-1$ (the $i = 0$ case will be considered later). The leftmost neuron $\sigma_{i,0}$ will carry the information about the value of $S_i$ toward $S^o_i$, letting pass this value or not depending on the state of $S^c_0$. Therefore, it will be connected only to $S_i$ (via a synapse $u_{i,i}$) and to $S^c_0$ (via a synapse $v^0_{i,0}$). On the other hand, the neuron $\sigma_{i,1}$ will carry the information about the state of the **previous** neuron $S_{i-1}$ toward $S^o_i$ , and therefore it will be connected only to $S_{i-1}$ (with synapse $u_{i,i-1}$) and to $S^c_0$ (with synapse $v^0_{i,1}$). Denoting by $T_{i,j}$ the thresholds of $\sigma_{i,j}$ ($j = 0, 1$) we have that

$$\sigma_{i,j} = \Theta\left[u_{i,i-j}S_{i-j} + v^0_{i,j}S^c_0 - T_{i,j}\right] \qquad (i = 1, \ldots, M-1\,;\, j = 0, 1) \tag{20}$$

The intermediate group $i = 0$ has to be considered separately. Since there is no previous input bit to $i = 0$ we need only one intermediate neuron $\sigma_{0,0} = \Theta\left[u_{0,0}S_0 + v^0_{0,0}S^c_0 - T_{0,0}\right]$. Finally, both neurons $\sigma_{i,0}$ and $\sigma_{i,1}$ will be connected to $S^o_i$ with synapses $w_{i,0}$ and $w_{i,1}$:

$$S^o_i = \Theta\left[w_{i,0}\sigma_{i,0} + w_{i,1}\sigma_{i,1} - T_i\right] \tag{21}$$

In figure 5 a schematic structure of a group $i$ is shown.

The structure works as follows: when $S_0^c = 0$ we have that $\sigma_{i,0} = S_i$ and $\sigma_{i,1} = 0 \; \forall i$. On the other hand, when $S_0^c = 1$ we have that $\sigma_{i,0} = 0$ and $\sigma_{i,1} = S_{i-1}$ for $i = 1, \ldots, M-1$. This results are summarized in table 4 and replacing them into Eq.(20) we find that

$$\begin{aligned} u_{i,i} &\geq T_{i,0} > 0 && (22) \\ v_{i,0}^0 &< T_{i,0} - u_{i,i} && (23) \end{aligned}$$

for $i = 0, \ldots, M-1$ and

$$\begin{aligned} 0 < u_{i,i-1} &< T_{i,1} && (24) \\ 0 < v_{i,1}^0 &< T_{i,1} && (25) \\ u_{i,i-1} + v_{i,1}^0 &\geq T_{i,1} && (26) \end{aligned}$$

for $i = 1, \ldots, M-1$. From table 4 we see that $S_i^o$ must satisfy $S_i^o = \sigma_{i,0}$ if $S_0^c = 0$ and $S_0^c = \sigma_{i,1}$ if $S_0^c = 1$. Since $\sigma_{i,0}$ and $\sigma_{i,1}$ can never be **both** equal to one at the same time, the above condition can be satisfied if $S_i^o = (\sigma_{i,0} \, \mathrm{OR} \, \sigma_{i,1})$ (logic OR operation). This last operation can be performed by taking $T_i \geq 0$ and

$$w_{i,j} \geq T_i \qquad (j = 0, 1) \tag{27}$$

for all $i$.

We have completed the construction of the 1-bit shifting net, obtaining a structure which, for an input number of $M$-bits, is composed by $4M-1$ neurons, $6M-3$ synaptic connections and a maximum number of inputs per neuron equal to two.

## 4.2 The $c$-bit shifter

We will now consider the most general case of a c-bit shifter, which allows to shift from none to all the bits of the input number.

In this case the intermediate group $i$ has to be able of carrying information about the state of any one of the $i+1$ input neurons $\{S_i, S_{i-1}, \ldots, S_1, S_0\}$ towards the output $S_i^o$, letting pass only the value of $S_{i-c}$. Therefore, we put in this group $i+1$ neurons $\sigma_{i,j}$ with $j = 0, \ldots, i$, which give a total number of $M(M+1)/2$ intermediate neurons; $\sigma_{i,j}$ will be connected to $S_{i-j}$ through a synapse $u_{i,i-j}$ and to all the $K$ indicating neurons $S_l^c$ through synapses $v_{i,j}^l$ $(l = 0, \ldots, K-1)$:

$$\sigma_{i,j} = \Theta \left[ u_{i,i-j} S_{i-j} + \sum_{l=0}^{K-1} v_{i,j}^l S_l^c - T_{i,j} \right] \quad (i = 1, \ldots, M-1\,;\, j = 0, \ldots, i). \tag{28}$$

Now, the synapses $\{u_{i,i-j}, v_{i,j}^l\}$ will be set in such a way that, for a c-bit shifting we will have, in every group $i$, $\sigma_{i,j} = 0$ for $j \neq c$ and $\sigma_{i,c} = S_{i-c}$.

Let start with the first intermediate neuron of every group $\sigma_{i,0}$. We have that

$$\sigma_{i,0} = \begin{cases} S_i & \text{if } S_l^c = 0 \; \forall l \\ 0 & otherwise. \end{cases} \tag{29}$$

Replacing into Eq.(28) we find that

$$\begin{aligned} u_{i,i} &\geq T_{i,0} > 0 & (30)\\ v_{i,0}^l &< T_{i,0} - u_{i,i} \quad \forall l & (31)\end{aligned}$$

for $i = 0, \ldots, M-1$. Note that all the synapses are inhibitories except $u_{i,i}$.

Let us now consider the rest of the intermediate neurons. Since $\sigma_{i,j} = 0\ \forall j > 0$ if $S_l^c = 0$ $\forall l$, we find from Eq.(28) that $T_{i,j} > 0$ for all values of $i$ and $j$, and

$$u_{i,i-j} < T_{i,j} \qquad \forall j > 0 \tag{32}$$

For the second neuron of every group $\sigma_{i,1}$ we have that

$$\sigma_{i,1} = \begin{cases} S_{i-1} & \text{if } S_0^c = 1 \textit{ and } S_l^c = 0\ \forall l \neq 0 \\ 0 & \textit{otherwise} \end{cases} \tag{33}$$

for $i = 1, \ldots, M-1$. Replacing into Eq.(28) we find that

$$v_{i,1}^0 < T_{i,1} \tag{34}$$

$$u_{i,i-1} + v_{i,1}^0 \geq T_{i,1} \tag{35}$$

$$T_{i,1} - (u_{i,i-1} + v_{i,1}^0) > v_{i,1}^l \quad \forall l \neq 0. \tag{36}$$

From Eqs.(32), (34) and (35) we see that $u_{i,i-1} > 0$ and $v_{i,1}^0 > 0$, *i.e.*, both synapses are excitatory while the rest are all inhibitory. Notice that a similar set of inequalities will be encountered for the synapses associated with any intermediate neuron that can be ON when only one indicating bit $S_l^c$ is ON, *i.e.*, when $c = 2^m$ with $m = 0, 1, \ldots, K-1$. In other words, for any group $i \geq 2^m$ we have that

$$\sigma_{i,2^m} = \begin{cases} S_{i-2^m} & \text{if } S_m^c = 1 \textit{ and } S_l^c = 0\ \forall l \neq m \\ 0 & \textit{otherwise} \end{cases} \tag{37}$$

and therefore

$$0 < v_{i,2^m}^m < T_{i,2^m} \tag{38}$$

$$u_{i,i-2^m} + v_{i,2^m}^m \geq T_{i,2^m} \tag{39}$$

$$T_{i,2^m} - (u_{i,i-2^m} + v_{i,2^m}^m) > v_{i,m}^l \quad \forall l \neq m. \tag{40}$$

Then, in all these cases only the two synapses $u_{i,i-2^m}$ and $v_{i,2^m}^m$ are excitatories and less than their respective threshold, while all the rest of the synapses are inhibitories according to (39) and (40).

The above outlined procedure can be applied to any one of the intermediate neurons. Then, given $\sigma_{i,j}$, suppose that there are $p$ bits $S_l^c = 1$ when $c = j$. Let $\{l_1, \ldots, l_p\}$ the set of index for which $S_l^c = 1$. Then

$$0 < v_{i,j}^l < T_{i,j} \quad \forall l \in \{l_1, \ldots, l_p\} \tag{41}$$

$$u_{i,j} + v^l_{i,j} \geq T_{i,j} \quad \forall l \in \{l_1, \ldots, l_p\} \tag{42}$$

$$T_{i,j} - (u_{i,j} + \sum_{\forall l \in \{l_1,\ldots,l_p\}} v^l_{i,j}) > v^{l'}_{i,m} \quad \forall l' \notin \{l_1, \ldots, l_p\}. \tag{43}$$

Finally, note that for any value of $c$ only the neuron $\sigma_{i,c} = S_{i-c}$ for every group $i \geq c$, while the rest of the intermediate neurons $\sigma_{i,j} = 0 \ \forall j \neq c$. Hence, the output $S^o_i$ can be calculated as $S^o_i = (\sigma_{i,0}$ OR $\sigma_{i,1}$ OR$\ldots$ $\sigma_{i,i-1}$ OR $\sigma_{i,i})$. This operation can be performed by setting $T_i > 0 \ \forall i$ and $w_{i,j} \geq T_i$ for $j = 0, \ldots, i$.

This procedure leads to a 3-layer structure for a $M$-bit shifter with a total number of neurons of order $\mathcal{O}(M^2)$, $2(M^2 + M)$ synapses and with a Fan-in Max equal to $M$.

A scheme of a left bit shifter for the case M=3 is shown in figure 6, and some net features for different sizes are shown in table 5.

## 4.3 Bi-directional shifter

The previous structures can be generalized in order to allow both left and right shifting. The functioning scheme is esentially the same as the previous one, but with the addition of new intermediate neurons to perform the right shift.

A net for right shifting would have the same structure described in the preceding subsection, with the difference that the intermediate neurons have to be arranged in a reverse way: the intermediate neurons corresponding to the first output bit have to be swapped with the ones corresponding to the last output bit and so on.

Finally, both structures can be combined into a single bi-directional shifter with the addition of a new input neuron $S_{sh}$ that indicates which operation have to be carried out: $S_{sh} = 0$ corresponds to a left shift and $S_{sh} = 1$ to a right one. In this case every intermediate group $i$ contains $M$ neurons $\sigma_{i,j}$ with $j = i-(M+1), i-M, \ldots, -1, 0, 1, \ldots, i$, each one connected to one input neuron $S_{i-j}$ and to all the indicating neurons $S^c_l$. $S_{sh}$ has an inhibitory connection with all the neurons $\sigma_{i,j}$ for $j = 1, \ldots, 0$, so that $\sigma_{i,j} = 0$ when $S_{sh} = 1$ independently of the input. This effect can be seen as a dynamic raising of all the thresholds for this subgroup of neurons when $S_{sh} = 1$. The reverse effect can be obtained for the intermediate neurons $\sigma_{i,j}$ with $j = i - (M+1), i - M, \ldots, -1$ as follows. We raise all their thresholds so that $\sigma_{i,j} = 0$ for any input when $S_{sh} = 0$ and we put an excitatory connection from $S_{sh}$ to all of them that supress this effect when $S_{sh} = 1$. This dynamic switch of the thresholds can be modelled as follows:

$$\sigma_{i,j} = \Theta\left[u_{i,i-j} S_{i-j} + \sum_{l=0}^{K-1} v^l_{i,j} S^c_l - T'_{i,j}(S_{sh})\right], \tag{44}$$

for $= 1, \ldots, M-1$, $j = i - (M+1), i - M, \ldots, -1, 0, 1, \ldots, i$, with

$$T'_{i,j}(S_{sh}) = \begin{cases} T_{i,j} + J_{i,j}(1 - S_{sh}) & \text{for } j = i - (M-1), \ldots, -1 \\ T_{i,j} + J_{i,j} S_{sh} & \text{for } j = 1, \ldots, i \end{cases} \tag{45}$$

where $J_{i,j} > u_{i,i-j} + \sum_{l=0}^{K-1} |v^l_{i,j}| - T_{i,j}$ and all the parameters $\left\{u_{i,i-j}, v^l_{i,j}, T_{i,j}\right\}$ remain as before. The intermediate neurons $\sigma_{i,0}$ are the only ones that are not connected to $S_{sh}$ since they only carry information towards the output neuron $S^o_i$ when no shift is performed. This network has $M^2$ intermediate neurons and a total number of $M^2(K+3) - M$ synapses.

In Fig. 7 we present an example of a bi-directional shifter with 5 input neurons. (the connections between the input and the intermediate layers are not shown for clarity). In this example the input is 01010 and the network performs a 1-place right shift giving the ouput 00101.

## 5 Conclusions and Remarks

We have presented the construction of 3 neural nets for the multiplication of two n-bit integers, the addition of $N$ integer numbers and for the bit-shifting problem. The design of the first two compared favorably with the previous designs [8,6], with the remarkable advantage of reducing the numbers of layers down to the optimal depth, 4 and 3 layers respectively. The adder network presents a backward connection structure, i.e., hidden neurons in a given group $i$ are only connected to neurons of the corresponding input group and to all the preceding input neurons. The corresponding synaptic weights decay exponentially with the distance to the input neuron (see eq. (15)). This structure results from the carry process and it has been already encountered in other networks for a related problem [3]. Different features of a network for adding $N$ integers numbers of $N$ bits by different methods are shown in table 6.

We constructed a depth-2 bi-directional shifter. The depth of this network is independent on the shifting range $2^k$. This is, to the best of our knowledge, the first solution of the shifting problem with this characteristic. This result has to be compared to that proposed by Anderson and Van Hessen [2], which is composed of $k$ hidden layers.

The same ideas used for the construction of our shifter can be readily extended to two dimensions.

It is worth noting that, being the shifting problem a linearly inseparable one, the construction of a depth-2 network proves that this is the optimal depth.

## 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) N. Alon and J. Bruck, Explicit constructions of depth-2 majority circuits for comparison and addition, SIAM J. Discrete. Math. 7 (1991) 1-8.

(2) C.H. Anderson and D.C. Van Essen, Shifter circuits: A computational strategy for dynamic aspects of visual processing, Proc. Natl. Acad. Sci. USA 84 (1987) 6297-6301.

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

(4) A. Hajnal, W. Maass, P. Pudlak, M. Szegedy and G. Turan, Threshold Circuits of bounded depth, IEEE Symp. Found. Comp. Sci. 28 (1987) 99-110.

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

(6) T. Hofmeister, W. Hohberg and S. Kohling, Some notes on Threshold Circuits and multiplication in depth 4, Inform. Proccesing Lett. 39 (1991) 99-110.

(7) E. Klingman, Microprocessor systems design (Prentice Hall, New Jersey, 1977) 54-65.

(8) R. Lauwereins and J. Bruck, Efficient Implementation of a Neural Multiplier, IBM Research, Tech. Report RJ 8138 (1991).

(9) B. Muller and J. Reinhardt, Neural Networks: An introduction. (Springer Verlag, Berlin, 1991).

(10) M. Paterson, N. Pippenger and U. Zwick, Optimal Carry Save Networks, Research Report RR166, Dept. of Computer Science, Univ. of Warwick, Coventry, UK. (1990)

(11) L. Prechelt, A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice, Neural Networks 9 (3) (1996) 457-462.

(12) F. Rosenblatt, Principles of Neurodynamics (Spartan, New York, 1962).

(13) K. Siu, J. Bruck and T. Kailath, Depth Efficient Neural Networks for Division and Related problems, IEEE Transactions on information Theory 39 (3) (1993).

(14) K. Siu and V. Roychowdhury. On optimal depth threshold circuits for multiplication and related problems. SIAM J. Discrete Math. 7 (2) (1994)

(15) T.L.H. Watkin, A. Rau, and M. Biehl, The statistical mechanics of learning a rule, Rev. Mod. Phys. 65 (2) (1993) 499-556.

# Captions for figures and tables

**Fig. 1:** Net structure example for solving the addition of the rightmost bit (corresponding to $2^0$) of three binary numbers $S_j$ $(j = 1, 2, 3)$. The numbers inside the circles indicate the thresholds values of the corresponding neurons.

**Fig. 2:** Net structure for adding three 2-bit numbers. The values inside the circles indicate the thresholds of the corresponding neurons. (see the text for the synapses values).

**Fig. 3:** Schematic network architecture for computing the product of the two 2-bit numbers: $\sigma_a = \sigma_a^1 \sigma_a^0$ and $\sigma_b = \sigma_b^1 \sigma_b^0$ $(\sigma_{a,b}^i = 0, 1)$, where $\sigma^0 = \sigma_a \times \sigma_b$ is the output. Every neuron in the first hidden layer performs a binary product between one bit of the multiplicand $\sigma_a$ and one bit of the multiplier $\sigma_b$. The second hidden and output layers are designed, as described in Section 2 to perform the addition of the previous results, like in hand calculations. Filled and open circles indicate active (ON) and rest (OFF) neurons respectively. In this particular example the input $\sigma_a = 10$ and $\sigma_b = 11$ is transformed into the numbers $h_o = 10$ and $h_1 = 100$, which in turn are added to give the output $S^0 = 0110$.

**Fig. 4:** Schematic representation of a 1-bit shifting transformation. In this example the input number $S = S_3 S_2 S_1 S_0$ $(S_i = 0, 1)$ is transformed into $S^o = S_3^o S_2^o S_1^o S_0^o$ with $S_i^o = S_{i-1}$ for $i = 1, 2, 3$ and $S_0^o = 0$.

**Fig. 5:** Network connectivity associated with the $i^{th}$ output neuron $S_i^o$ for a 1-bit (left) shifter. The variables inside the open circles are the thresholds of the corresponding neurons, while the variables besides the solid lines are the synaptic weights.

**Fig. 6:** Network architecture for a bit shifter of a 3-bit number $S = S_2 S_1 S_0$. The number of shifting places is given by $S^c = S_1^c S_0^c$. (see text for details).

**Fig. 7:** Example of a 1-place right shift with a bi-directonal shifter. Filled and open circles indicate active (ON) and rest (OFF) neurons respectively. In this example the input is 01010, $S_{sh} = 1$, and $S^c = 01$, so the network performs a 1-place right shift.

**Table 1:** Number of neurons OFF and ON in the intermediate group $(k = 0 : \sigma_i^o$, $i = 0, \ldots, N_0 - 1)$, for different parity combinations of $N_0$ and the sum of the input bits $H$ (Eq. 2). The last column indicates the desired values of the output $S_0^o$, in order to compute the parity of H (Eq. 3).

**Table 2:** Several network parameters for classical size values of a Neural adder.

**Table 3:** Several network parameters for classical size values of a Neural multiplier.

**Table 4:** Activation states of the intermediate neurons associated with the output bit $S_i^0$ for a 1- bit shifter.

**Table 5:** Several Networks parameters for classical size values of a bit-shifter.

**Table 6:** Comparison between the principal parameters of different methods for constucting a network that computes the Sum of $N$ numbers of $N$-bit length.

(†) Lauwereins & Bruck (1991).

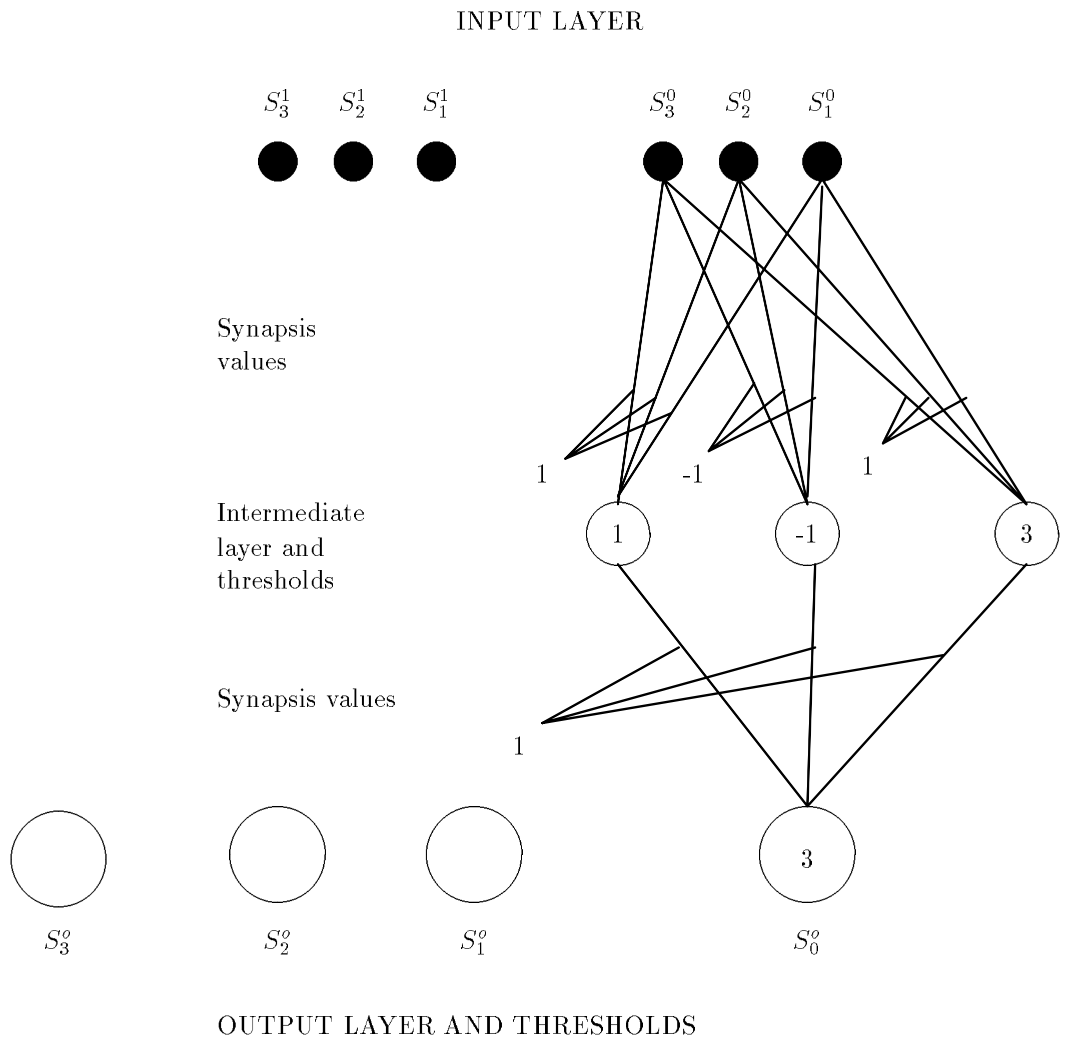


Figure 1 - Franco

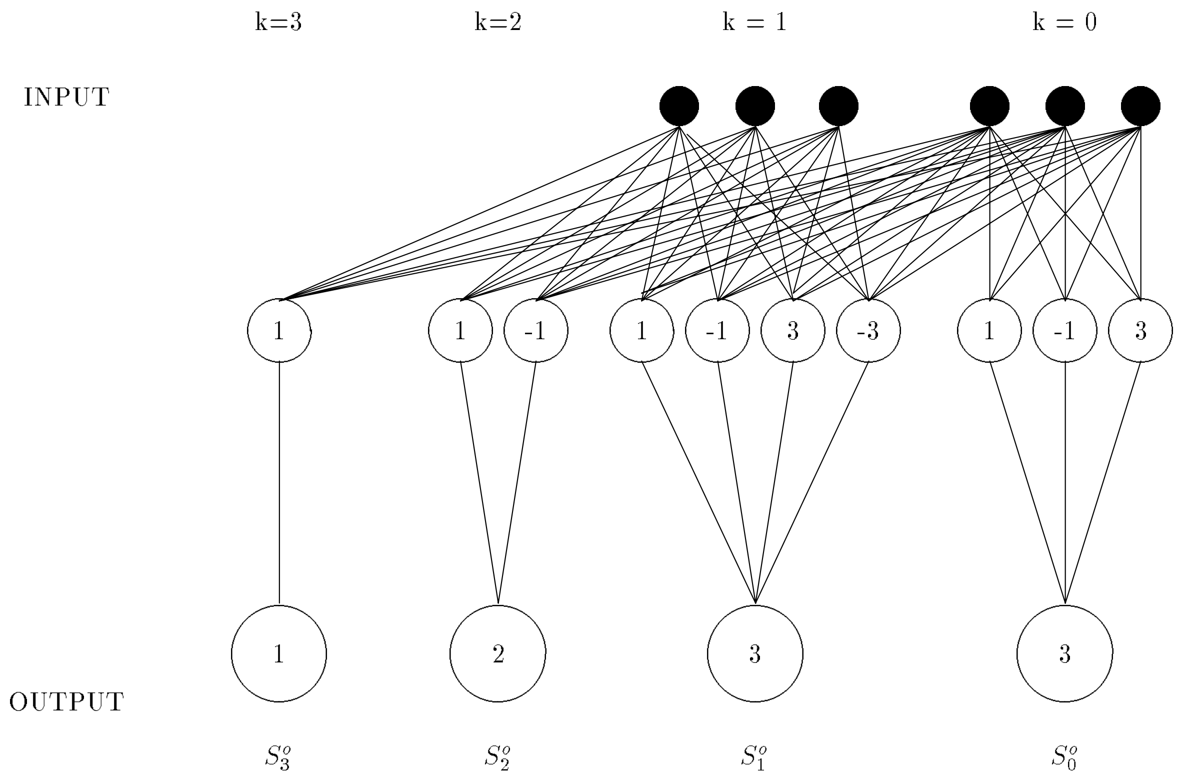
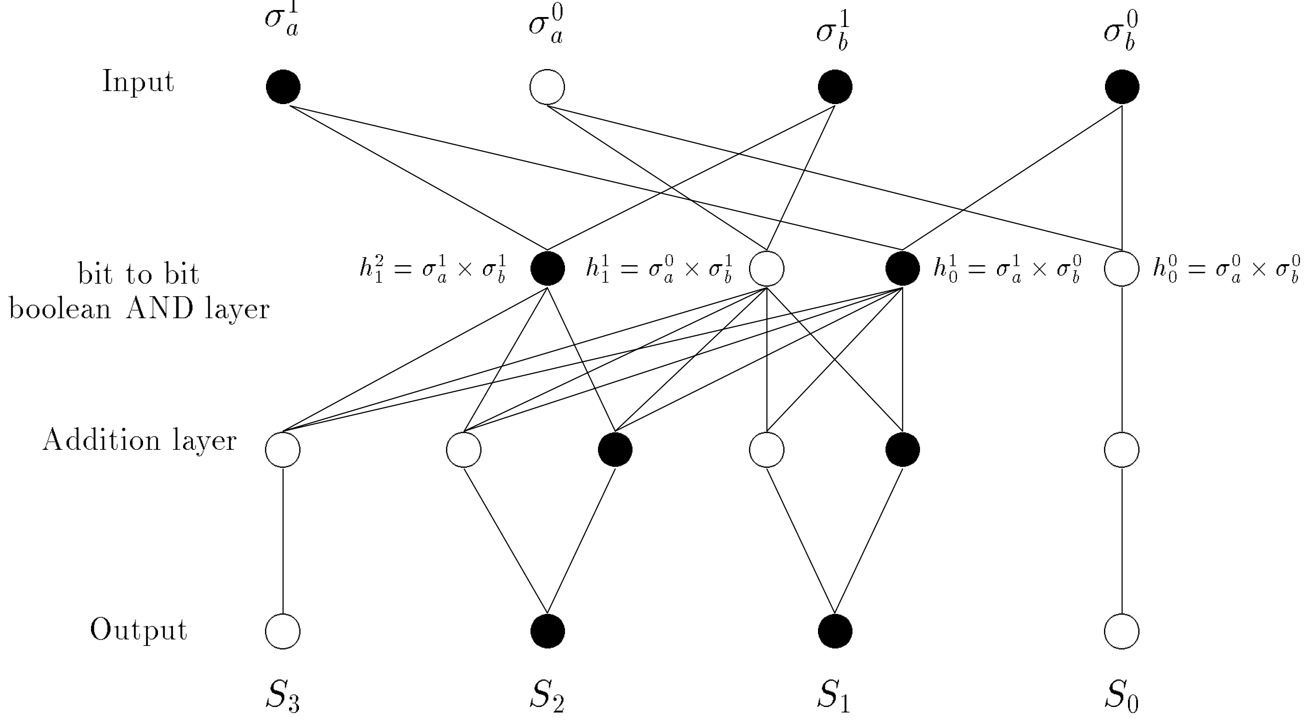


Figure 2 - Franco


$\sigma_a^1$
$\sigma_a^0$
$\sigma_b^1$
$\sigma_b^0$
Input
bit to bit boolean AND layer
$h_1^2 = \sigma_a^1 \times \sigma_b^1$
$h_1^1 = \sigma_a^0 \times \sigma_b^1$
$h_0^1 = \sigma_a^1 \times \sigma_b^0$
$h_0^0 = \sigma_a^0 \times \sigma_b^0$
Addition layer
Output
$S_3$
$S_2$
$S_1$
$S_0$

$$\begin{array}{cccccl} & \times & 1 & 0 & = & \sigma_a \\ & & 1 & 1 & = & \sigma_b \\ \hline & & 1 & 0 & = & \sigma_a \times \sigma_b^0 = h_0 \\ & 1 & 0 & & = & \sigma_a \times \sigma_b^1 = h_1 \\ \hline 0 & 1 & 1 & 0 & = & S \end{array}$$

Figure 3 - Franco

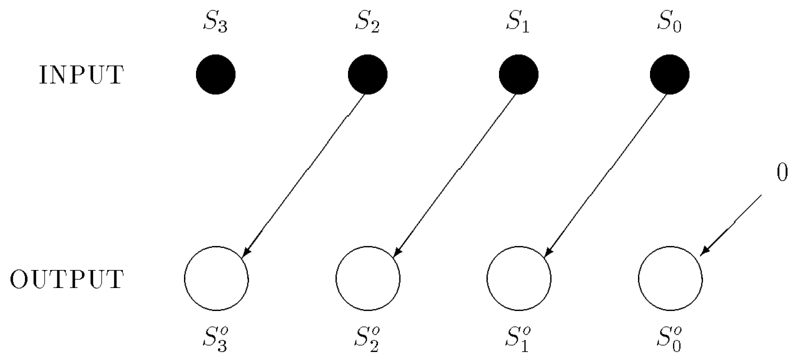


Figure 4 - Franco

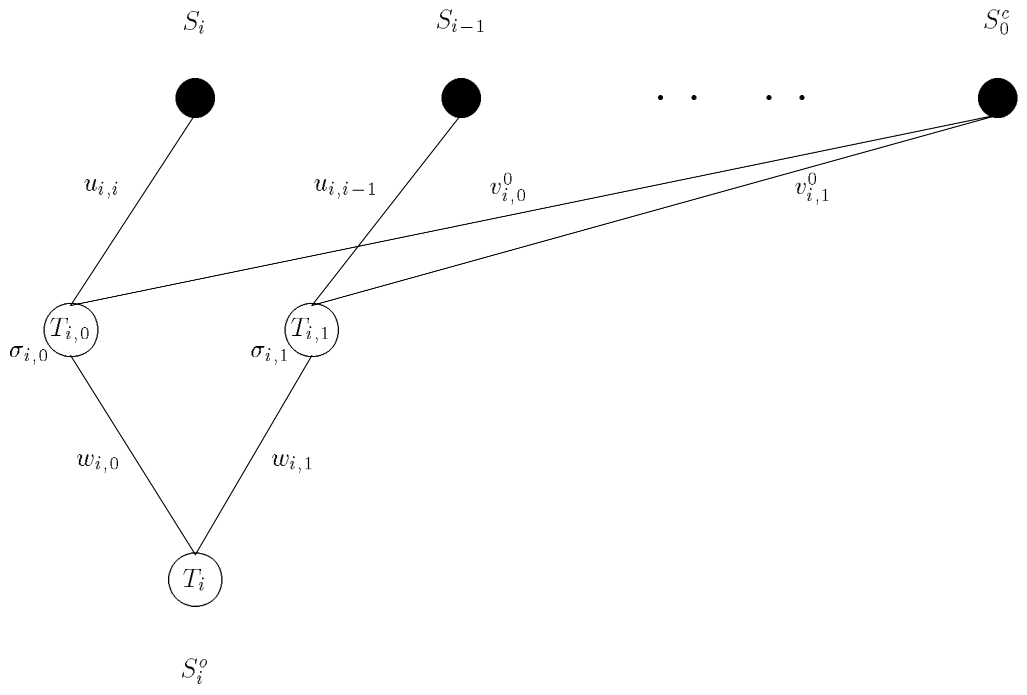


Figure 5 - Franco

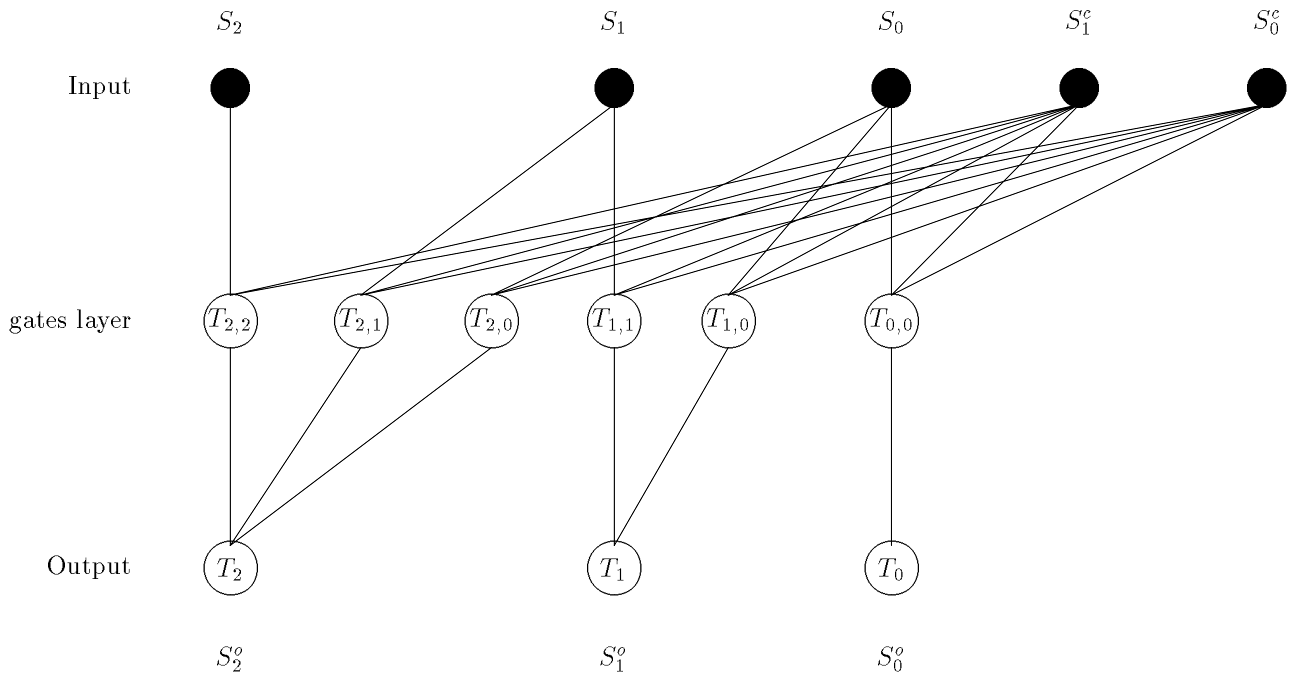


Figure 6 - Franco

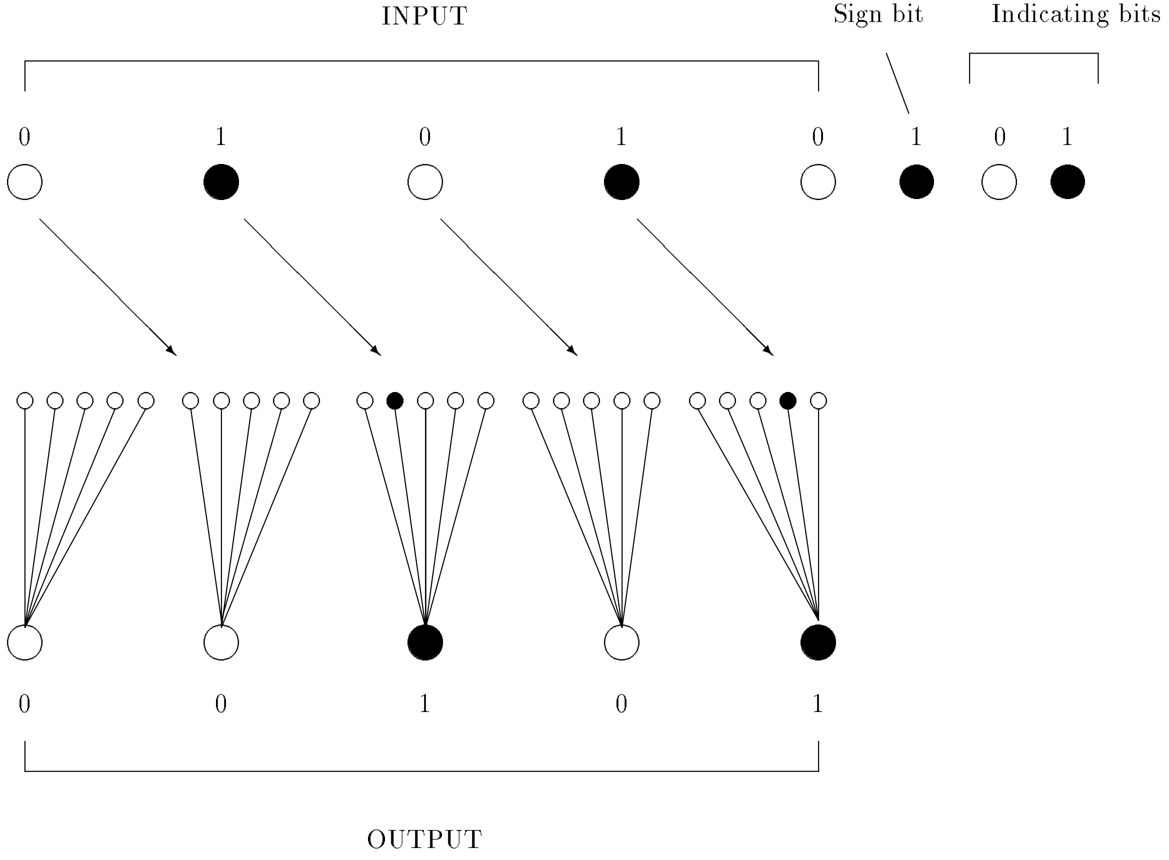


Figure 7 - Franco

| $H$=Sum Result | $N_o$ | Neurons ON | Neurons OFF | $S_0^o$ |
|---|---|---|---|---|
| odd | even | $(N_o/2) + 1$ | $(N_o/2) - 1$ | ON |
| odd | odd | $(N_o + 1)/2$ | $(N_o - 1)/2$ | ON |
| even | even | $N_o/2$ | $N_o/2$ | OFF |
| even | odd | $(N_o - 1)/2$ | $(N_o + 1)/2$ | OFF |

Table 1 - Franco

| Size $(N,p)$ | Neurons | Synapses | Depth | Fan-in Max |
|---|---|---|---|---|
| ( 4 , 4 ) | 50 | 352 | 2 | 15 |
| ( 8 , 8 ) | 195 | 5016 | 2 | 63 |
| ( 16 , 16 ) | 772 | 73936 | 2 | 255 |
| ( 32 , 32 ) | 3077 | 1121664 | 2 | 1023 |
| ( $N$ , $p$ ) | $\mathcal{O}(3Np)$ | $\mathcal{O}(N^2p^2)$ | 2 | $Np-1$ |

Table 2 - Franco

| Size | Neurons | Synapses | Depth | Fan-in Max |
|---|---|---|---|---|
| 4x4 | 60 | 341 | 3 | 16 |
| 8x8 | 216 | 4743 | 3 | 64 |
| 16x16 | 816 | 70539 | 3 | 256 |
| 32x32 | 3168 | 1087763 | 3 | 1024 |
| $n$x$n$ | $3(n^2+n)$ | $\mathcal{O}(n^4)$ | 3 | $n^2$ |

Table 3 - Franco

| $S_0^c$ | $\sigma_{i,0}$ | $\sigma_{i,1}$ | $S_i^0$ |
|---|---|---|---|
| 0 | $S_i$ | 0 | $S_i$ |
| 1 | 0 | $S_{i-1}$ | $S_{i-1}$ |

Table 4 - Franco

| Size | Neurons | Synapses | Depth | Fan-in Max |
|---|---|---|---|---|
| 4 | 21 | 40 | 2 | 4 |
| 8 | 56 | 144 | 2 | 8 |
| 16 | 177 | 544 | 2 | 16 |
| 32 | 597 | 2112 | 2 | 32 |
| M | $\mathcal{O}(M^2)$ | $2(M^2+M)$ | 2 | $M$ |

Table 5 - Franco

| Method | Neurons | Synapses | Fan-in Max | Depth |
|---|---|---|---|---|
| Block Save Addition (BSA) † | $\mathcal{O}(N^3)$ | $\mathcal{O}(N^4 log N)$ | $\mathcal{O}(N^2)$ | 3 |
| BSA optimized & Dortmunder † | $\mathcal{O}(N^2)$ | $\mathcal{O}(N^3 log N)$ | $\mathcal{O}(N log N)$ | 3 |
| Our | $\mathcal{O}(N^2)$ | $\mathcal{O}(N^4)$ | $\mathcal{O}(N^2)$ | 2 |

Table 6 - Franco