A Novel Algorithm to Train Multilayer Hardlimit Neural Networks Based on a Mixed Integer Linear Program Model

Jose B. da Fonseca(✉)

Faculty of Sciences and Technology, New University of Lisbon,
2829-615 Caparica, Portugal
jbfo@fct.unl.pt
http://www.dee.fct.unl.pt

Abstract. In a previous work we showed that hardlimit multilayer neural networks have more computational power than sigmoidal multilayer neural networks [1]. In 1962 Minsky and Papert showed the limitations of a single perceptron which can only solve linearly separable classification problems and since at that time there was no algorithm to find the weights of a multilayer hardlimit perceptron research on neural networks stagnated until the early eighties with the invention of the Backpropagation algorithm [2]. Nevertheless since the sixties there have arisen some proposals of algorithms to implement logical functions with threshold elements or hardlimit neurons that could have been adapted to classification problems with multilayer hardlimit perceptrons and this way the stagnation of research on neural networks could have been avoided. Although the problem of training a hardlimit neural network is NP-Complete, our algorithm based on mathematical programming, a mixed integer linear model (MILP), takes few seconds to train the two input XOR function and a simple logical function of three variables with two minterms. Since any linearly separable logical function can be implemented by a perceptron with integer weights, varying them between -1 and 1 we found all the 10 possible solutions for the implementation of the two input XOR function and all the 14 and 18 possible solutions for the implementation of two logical functions of three variables, respectively, with a two layer architecture, with two neurons in the first layer. We describe our MILP model and show why it consumes a lot of computational resources, even a small hardlimit neural network translates into a MILP model greater than 1G, implying the use of a more powerful computer than a common 32 bits PC. We consider the reduction of computational resources as the near future work main objective to improve our novel MILP model and we will also try a nonlinear version of our algorithm based on a MINLP model that will consume less memory.

Keywords: Hardlimit neural networks · Mixed integer linear programming · Training a hardlimit neural network with a MILP model · Solving a MILP model with the cplex solver

I. Rojas et al. (Eds.): IWANN 2015, Part II, LNCS 9095, pp. 477–487, 2015.
DOI: 10.1007/978-3-319-19222-2_40

1 Introduction

In their famous book Perceptrons [2] Minsky and Papert showed that a single perceptron can only solve linearly separable classification problems and to solve a nonlinearly separable classification problem they would need a multilayer perceptron. Since at that time, 1962, there was no algorithm to find the weights of a multilayer perceptron, research on neural networks stagnated until the early eighties with the invention of the Backpropagation algorithm. Nevertheless since the early sixties there appeared many algorithms to train hardlimit neural networks or multilayer threshold gates to implement logical functions that could have been adapted to solve nonlinearly separable classification problems. The first published work on the synthesis of multilayer threshold gates to implement logical functions seems to be [3]. So why to create one more algorithm? Because it is shown that the training of a multilayer hardlimit perceptron is NP-Complete [4,5] and we will show that our solution is more efficient than previous algorithms in terms of runtime, although it consumes a lot of memory that increases exponentially with the network size. Nevertheless our MILP model is simple and beautiful with only two constraints and given a set of possible integer values of weights it always finds all the possible solutions with a given architecture. We could have used a nonlinear model that would consume much less memory since the weights could be, in this case, represented by a simple array of integers, since it is shown that any logical function can be implemented by a multilayer hardlimit neural network with integer weights [4]. Nevertheless the nonlinear model could be trapped in a local optimum and the nonlinear model needs an initial feasible solution. On the contrary our MILP model always finds the global optimum since it is a linear model. We have already done some experiments with a nonlinear MNILP model but the results are very poor: the model did not converge to any solution using all possible nonlinear solvers.

2 What is Mathematical Programming? What is a MILP Model?

A mathematical programming model is a set of constraints over the variables of the model subjected to a maximization or minimization of an objective variable also defined in terms of the model variables. A linear mathematical program model is a mathematical program where all constraints are linear and also the expression that defines the objective variable. A mixed integer linear program model (MILP) is a linear mathematical programming model with integer and binary variables. A linear model can neither have the multiplication or division of two variables nor a nonlinear operation over a model variable. To solve the MILP model we use a black box algorithm, the solver, normally the Cplex solver [6], through a modelling language, the GAMS software, that makes transparent the use of the solver by a non specialist.

3 Description of a MILP Model to Train Hardlimit Multilayer Neural Networks

The main idea behind our work is expressed by the following theorem:

Theorem 1. Any linearly separable logical function of n variables can be computed by a Rosenblatt neuron or threshold gate with integer weights such that

$$|w_i| \leq \frac{(n+1)^{(n+1)/2}}{2^n}, i = 0, ..., n \tag{1}$$

For n=3, this theorem guarantees that all linearly separable logical functions of three variables can be implemented by a single Rosenblatt perceptron or threshold gate with integer weights varying between -2 and 2. You can find a demonstration of theorem 1 in [4]. Since there are only 104 linearly separable functions over 256 possible functions of three variables, we considered an architecture with two layers, two hidden neurons in the first layer and an output neuron, to implement a two input XOR and a three variable function. Preliminary results seem to show that we can implement almost all logical functions of three variables with this architecture with the integer weights varying only between -1 and 1 to reduce the model size, but we found one counter example f=m_1+m_2+m_3+m_4 where the MILP did not find any solution for this architecture. Varying the integer weights between -2 and 2 we exhausted the memory because the model takes a dimension of 1.8G and the GAMS software on a 32 bits machine only uses 2G RAM. In a near future we plan to install GAMS on a 64 bits machine that will not have this memory limit.

The *brain* of our MILP model is an indexed binary variable, $weights(w_i,bias_j)$, that assumes the value 1 for all combinations of integer weights that are considered solutions of the implementation of the logical function. For the case of a two input XOR defined by (2) the solutions are generated by the set of linear constraints (3).

$$xor(b_0, b_1) \equiv b_0 \neq b_1 \equiv b_0\bar{b_1} + \bar{b_0}b_1 \tag{2}$$

$$\forall w_{ij}, bias_i, b_0, b_1 : \tag{3}$$
$$weights(w_{ij}) f(w_{ij}, bias_i, b_0, b_1) = xor\_2in(b_0, b_1) weights(w_{ij}, bias_i)$$

The set of *linear* constraints (3) are equivalent to the simpler set of *nonlinear* constraints (4), since we make a restriction based on a logical operation over a model variable. In this sense (3) results from a linearization of (4).

$$\forall w_{ij}, bias_i, b_0, b_1 \setminus \{weights(w_{ij}, bias_i) = 0\} : \tag{4}$$
$$f(w_{ij}, bias_i, b_0, b_1) = xor(b_0, b_1)$$

Since we want to obtain all possible solutions of the XOR with a given architecture and integer weights varying between -1 and 1, we must maximize the number of solutions, the objective variable which is defined by (5).

$$obj = \sum_{w_{ij}, bias_i} weights(w_{ij}, bias_i) \tag{5}$$

For the case of a XOR function (5) is implemented by the following line of GAMS code: calc_obj.. sum((w11, w12, w13, w14, w21,w22, bias1, bias2, bias3), weights(w11, w12, w13, w14, w21, w22, bias1, bias2, bias3))=e=obj;

So we linearized a very nonlinear set of constraints. But the price we paid is very high: since the binary variable weights and the constraint calc_ws1 are indexed by all the weights and input variables in this simple example we will have a search space with dimension 39x4=78732, assuming only three possible values for the weights -1, 0, 1 which generates a model with 18M. But if we increase the domain of variation of weights to -2,-1,0,1,2 we will have a search space with dimension 59x4= 7812500 which corresponds to a model with 1.3G and we got an out of memory message in a machine with 2G RAM. The set of linear constraints (4) is implemented by the following GAMS code:

calc_ws1(w11,w12,w13,w14,w21,w22,bias1,bias2,bias3,b0,b1).. weights(w11,w12, w13, w14, w21, w22, bias1, bias2, bias3) * f(w11, w12, w13, w14, w21, w22, bias1, bias2, bias3, b0, b1) =e= xor_2in(b0,b1) * weights(w11, w12, w13, w14, w21, w22, bias1, bias2, bias3);

When weights(w11,w12,w13,w14,w21,w22,bias1,bias2,bias3)=0 the constraint is relaxed and equivalent to 0=e=0 and when weights(w11, w12, w13, w14, w21, w22, bias1, bias2, bias3)=1 the constraint imposes that the output of the neural network, f(w11, w12, w13, w14, w21, w22, bias1, bias2, bias3, b0, b1), must be equal to the desired output xor_2in(b0, b1) for all combinations of (b0, b1), i.e. (w11, w12, w13, w14, w21, w22, bias1, bias2, bias3) is a solution of the weights that implement a two input xor. We have already developed a first version of a nonlinear model to solve this problem but although the model got smaller, about 5M, all nonlinear solvers we have access did not reach any solution.

Function *f*, the output of the multilayer perceptron, is defined as a parameter given by (6).

$$\begin{aligned} f = ((\quad & \\ (((b_0 - 1)(w_{11} - 2) + (b_1 - 1)(w_{12} - 2) + bias_1 - 2 \geq 0)(w_{21} - 2) + & \\ (b_0 - 1)(w_{13} - 2) + (b_1 - 1)(w_{14} - 2) + bias_2 - 2 \geq 0)(w_{22} - 2) + & \\ bias_3 - 2)) \geq 0 & \end{aligned} \tag{6}$$

Note that the subtraction of the weights by 2 is only correct for weights varying in {-1,0,1}. For a greater set of variation we must increase this constant. Equation (6) is implemented by the single line of GAMS code:
f(w11,w12,w13,w14, w21,w22, bias1, bias2, bias3, b0, b1)= (((((ord(b0)-1) * (ord(w11)-2) + (ord(b1)-1)*(ord(w12)-2) + ord(bias1)-2) ge 0) * (ord(w21)-2) + (((ord(b0)-1) * (ord(w13)-2) + (ord(b1)-1)*(ord(w14)-2) + ord(bias2)-2) ge 0) * (ord(w22)-2) + ord(bias3)-2)) ge 0);

3.1 Implementing a Three Input Logical Function

As a second example we will show how to obtain the weights of a two layer hardlimit perceptron, with two neurons in the first layer, to implement a three input logical function defined by (7) and the weights varying between -1 and 1 to reduce the combinatorial explosion. We use two layers of neurons because from the 256 possible functions of three variables only 106 are linearly separable and so the remaining cannot be implemented by a single neuron. Augmenting the number of neurons in the first layer and the weight variation domain to -2..2 we got an out of memory with a 2G RAM PC and so there are lots of three input logical functions that still cannot be implemented with this architecture since to implement all the nonlinearly separable logical functions of three variables we will need more neurons in the first layer and to augment the variation domain to -2..2 to guarantee the implementation of minterms in the first layer following theorem 1. The sets of constraints (8) impose that for all solutions the output of the network must be equal to the desired function.

$$f(b_0, b_1, b_2) = m_1 + m_4 = \bar{b_2}\bar{b_1}b_0 + b_2\bar{b_1}\bar{b_0} \tag{7}$$

$$\begin{aligned} &\forall w_{ij}, bias_i, b_0, b_1, b_2 : \\ &weights(w_{ij}, bias_i) f(w_{ij}, bias_i, b_0, b_1, b_2) = \\ &weights(w_{ij}, bias_i) f3in(w_{ij}, bias_i, b_0, b_1, b_2) \end{aligned} \tag{8}$$

4 Conclusions and Future Work

Although there exist many published algorithms to train multilayer threshold gates or hardlimit neural networks to implement logical functions, we showed that our algorithm, to our knowledge the first in the literature to be based on a MILP model and solved by the Cplex solver, has better runtimes and is simpler than previous works and can also solve classification problems. In a near future we plan to implement a nonlinear MINLP model that will consume much less memory, although it will have the drawbacks of the possibility of getting trapped in a local optimum and the necessity of an initial feasible suboptimal solution.

References

1. Barahona da Fonseca, J.: Are Rosenblatt multilayer perceptrons more powerfull than sigmoidal multilayer perceptrons? From a counter example to a general result. In: Proceedings of ESANN 2013, Ciaco, Belgium, pp. 345–350 (2013)
2. Minsky, M., Papert, S.: Perceptrons. MIT Press, Massachusetts (1965)
3. Gonzalez, R., Lawler, E.L.: Two-level threshold minimization. In: Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design, 41–4 (1965)

4. Orponen, P.: Computational Complexity of Neural Networks: a Survey. NC-TR-94-010 Technical Report, Department of Computer Science, University of London, England (1994)
5. Blum, A.L., Rivest, R.L.: Training a 3-node neural network is NP-complete. Lecture Notes in Computer Science, 661 (1993)
6. ILOG: Cplex Solver (2005)

A GAMS code of the First MILP Model to Implement a two input XOR with a two layer Threshold Gate

```
* XOR 2in

sets b0 /0*1/;
alias(b1,b0);

sets w11 /1*3/;
alias( w12, w13, w14, w21, w22, bias1, bias2, bias3, w11);


Parameter xor_2in(b0,b1), f(w11,w12,w13,w14, w21,w22, bias1, bias2, bias3, b0, b1);

*Definition of a 2 Input XOR:
xor_2in(b0,b1)=( ord(b0) ne ord(b1));

* Output of the Neural Network:
f(w11,w12,w13,w14, w21,w22, bias1, bias2, bias3, b0, b1)=
( ((
( ( ( ord(b0)-1 ) * (ord(w11)-2) + (ord(b1)-1)*(ord(w12)-2) +
ord(bias1)-2 )  ge 0) * (ord(w21)-2) +
( ( ( ord(b0)-1 ) * (ord(w13)-2) + (ord(b1)-1)*(ord(w14)-2) + ord(bias2)-2 )  ge 0) * (ord(w22)-2) +
ord(bias3)-2 ) ) ge 0 ) ;

variable obj;

binary variable
weights(w11,w12,w13,w14, w21,w22, bias1, bias2, bias3);

**CONSTRAINTS**

calc_ws1(w11,w12,w13,w14,w21,w22,bias1,bias2,bias3,b0,b1)..

calc_ws1(w11,w12,w13,w14,w21,w22,bias1,bias2,bias3,b0,b1)..
weights(w11,w12,w13,w14,w21,w22,bias1,bias2,bias3)*
f(w11,w12,w13,w14, w21,w22, bias1, bias2, bias3, b0, b1)=e=
xor_2in(b0,b1)*weights(w11,w12,w13,w14,w21,w22,bias1,bias2,bias3);

calc_obj.. obj=e=
sum( (w11,w12,w13,w14, w21,w22, bias1, bias2, bias3),
weights(w11,w12,w13,w14, w21,w22, bias1, bias2, bias3) );

Model Xor2in /all/;

Solve Xor2in using mip maximizing obj;

display weights.l, obj.l;
```

B Output of a Run of the First MILP Model

```
----     49 VARIABLE weights.L

INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3
               1
2.3       1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1
               1
3.2       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 1  INDEX 6 = 3
               2
2.1       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 3  INDEX 6 = 1
               2
```

```
1.2       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 1
              3
2.2       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 3
              1
1.1       1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 1  INDEX 6 = 1
              3
2.2       1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 3  INDEX 6 = 3
              1
1.1       1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3
              2
2.1       1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1
              2
1.2       1.000
----     49 VARIABLE obj.L                =       10.000

These 10 solutions correspond to the following sets of weight values, respectively:

w11= -1  w12= -1  w13= -1  w14= -1  w21= -1  w22= 1
bias1=0 bias2=1 bias3=-1

w11= -1  w12= -1  w13= -1  w14= -1  w21= 1  w22= -1
bias1=1 bias2=1 bias3=-1

w11= -1  w12= 1  w13= -1  w14= 1  w21= -1  w22= 1
bias1=0 bias2=-1 bias3=0

w11= -1  w12= 1  w13= -1  w14= 1  w21= 1  w22= -1
bias1=-1 bias2=0 bias3=0

w11= -1  w12= 1  w13= 1  w14= -1  w21= -1  w22= -1
bias1=0 bias2=0 bias3=1

w11= -1  w12= 1  w13= 1  w14= -1  w21= 1  w22= 1
bias1=-1 bias2=-1 bias3=-1
This solution corresponds to the first direct implementation of the XOR, the two neurons in the first layer
implement the two minterms, m2 and m1 respectively, and the third neuron implement the OR of these minterms
resulting in the two input xor

w11= 1  w12= -1  w13= -1  w14= 1  w21= -1  w22= -1
bias1=0 bias2=0 bias3=1

w11= 1  w12= -1  w13= -1  w14= 1  w21= 1  w22= 1
bias1=-1 bias2=-1 bias3=-1
This solution also corresponds to the second direct implementation of the XOR, the two neurons in the first layer
implement the two minterms, m1 and m2 respectively,  and the third neuron implement the OR of these minterms.
This solution is equivalent to the previous since it is irrelevant which neuron in the first layer implements
m1 or m2.

w11= 1  w12= -1  w13= 1  w14= -1  w21= -1  w22= 1
bias1=0 bias2=-1 bias3=0

w11= 1  w12= -1  w13= 1  w14= -1  w21= 1  w22= -1
bias1=-1 bias2=0 bias3=0
```

C GAMS code of the Second MILP Model to Implement a Three Variable Logical Function, with a single Perceptron

```
* F 3in=m6

sets b0 /0*1/;
alias(b1,b2,b0);

sets w11 /1*5/;
alias( w12, w13, w14, w15, w16,w21, w22, bias1, bias2, bias3, w11);

Parameter f_3in_d(b0,b1,b2), f(w11,w12,w13, bias1, b0, b1,b2);

f_3in(b0,b1,b2)=(ord(b0)=1) * (ord(b1)=2) * (ord(b2)=2);

f(w11,w12,w13, bias1, b0, b1,b2)=
( ( ( ord(b0)-1 ) * (ord(w11)-3) + (ord(b1)-1)*(ord(w12)-3) + (ord(b2)-1)*(ord(w13)-3)
+ ord(bias1)-3 )  ge 0) ;

variable obj;
```

```
binary variable weights(w11,w12,w13, bias1);

calc_ws1(w11,w12,w13, bias1,b0,b1,b2)..
weights(w11,w12,w13, bias1)*
f(w11,w12,w13, bias1,b0,b1,b2)=e= f_3in(b0,b1,b2)*weights(w11,w12,w13, bias1);

calc_obj.. obj=e=sum( (w11,w12,w13, bias1), weights(w11,w12,w13, bias1));

Model f3in /all/;

Solve f3in using mip maximizing obj;

display weights.l, obj.l, f_3in_d;
```

D Output of the Second MILP Model to Implement a Three Variable Logical Function, f =m6, with a Single Perceptron with Weights between -2 and 2

```
----     48 VARIABLE weights.L
INDEX 1 = 1
                1
4.4       1.000
INDEX 1 = 2
                1
4.4       1.000
----     48 VARIABLE obj.L                =        2.000
----     48 PARAMETER f_3in
                1

0.1       1.000

These two solutions correspond to the following weights that implement m6= :

w1=-2, w2=1, w3=1, bias=-2
w1=-1, w2=1, w3=1, bias=-2
```

E GAMS code of the Third MILP Model to Implement a Three Variable Logical Function with a Two Layer Perceptron

```
* F 3in
sets b0 /0*1/;
alias(b1,b2,b0);
sets w11 /1*3/;
alias( w12, w13, w14, w15, w16,w21, w22, bias1, bias2, bias3, w11);

Parameter f_3in_d(b0,b1,b2),
f(w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3, b0, b1,b2);

*f_3in_d=m1 + m4
f_3in_d(b0,b1,b2)=(ord(b0)=1) * (ord(b1)=1) * (ord(b2)=2) +
(ord(b0)=2) * (ord(b1)=1) * (ord(b2)=1);

* Output of the Neural Network, f():

f(w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3, b0, b1, b2)=
(  ((   ( ( ord(b0)-1 ) * (ord(w11)-2) + (ord(b1)-1)*(ord(w12)-2) + (ord(b2)-1)*(ord(w13)-2) + ord(bias1)-2 )  ge 0) * (ord(w21)
-2) + ( ( ( ord(b0)-1 ) * (ord(w14)-2) +
(ord(b1)-1)*(ord(w15)-2)+ (ord(b2)-1)*(ord(w16)-2) + ord(bias2)-2 )  ge 0) * (ord(w22)-2) +
 ord(bias3)-2 ) )  ge 0) ;

variable obj;

binary variable
weights (w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3);

**CONSTRAINTS**
calc_ws1(w11,w12,w13,w14,w15,w16,w21,w22, bias1, bias2, bias3, b0,b1,b2)..
weights(w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3)*
f(w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3,b0,b1,b2)=e=
```

```
weights(w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3)*
 f_3in(b0,b1,b2);

calc_obj.. obj=e=sum(
(w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3),
weights( w11,w12,w13,w14,w15,w16, w21,w22, bias1, bias2, bias3) );

Model f3in /all/;

Solve f3in using mip maximizing obj;
```

F Output of a Run of the Second MILP Model for $f=m_1+m_4$

```
----     47 VARIABLE weights.L

INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 2  INDEX 6 = 1   INDEX 7 = 3  INDEX 8 = 1

              1
3.2       1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1   INDEX 7 = 3  INDEX 8 = 1
              1
3.2       1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 3   INDEX 7 = 3  INDEX 8 = 1
              2
1.2       1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 3  INDEX 5 = 1  INDEX 6 = 1  INDEX 7 = 3  INDEX 8 = 3
              1
1.1       1.000
INDEX 1 = 1  INDEX 2 = 2  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 1  INDEX 7 = 1  INDEX 8 = 3
              1
2.3       1.000
INDEX 1 = 1  INDEX 2 = 2  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1   INDEX 7 = 3  INDEX 8 = 1
              1
3.2       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 1   INDEX 7 = 1  INDEX 8 = 3
              1
2.3       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 2  INDEX 6 = 1   INDEX 7 = 1  INDEX 8 = 3
              1
2.3       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3   INDEX 7 = 1  INDEX 8 = 3
              2
2.1       1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 3  INDEX 4 = 3  INDEX 5 = 3  INDEX 6 = 1   INDEX 7 = 1  INDEX 8 = 1
              3
2.2       1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3   INDEX 7 = 3  INDEX 8 = 3
              1
1.1       1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 3  INDEX 6 = 1   INDEX 7 = 3  INDEX 8 = 1
              2
1.2       1.000
INDEX 1 = 3  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 3   INDEX 7 = 1  INDEX 8 = 1
              3
2.2       1.000
INDEX 1 = 3  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 1  INDEX 6 = 1 INDEX 7 = 1  INDEX 8 = 3
              2
2.1       1.000

----     47 VARIABLE obj.L                =       14.000

----     47 PARAMETER f_3in_d
              0           1

0.0                   1.000
1.0       1.000

These 14 solutions correspond to the following sets of weight values, respectively:

1. w11= -1  w12= -1  w13= -1  w14= -1 w15= 0  w16= -1   w21= 1  w22= -1
bias1=1 bias2=0 bias3=-1
2. w11= -1  w12= -1  w13= -1  w14= -1 w15= 1  w16= -1   w21= 1  w22= -1
bias1=1 bias2=0 bias3=-1
3. w11= -1  w12= -1  w13= 1  w14= -1 w15= 1  w16= 1   w21= 1  w22= -1
bias1=-1 bias2=0 bias3=0
4. w11= -1  w12= -1  w13= 1  w14= 1 w15= -1  w16= -1   w21= 1  w22= 1
bias1=-1 bias2=-1 bias3=-1**
This solution corresponds to the direct implementation of the two minterms by the two neurons of the first layer and the OR of the
two minterms by the third neuron

5. w11= -1  w12= 0  w13= -1  w14= -1 w15= -1  w16= -1   w21= -1  w22= 1
bias1=0 bias2=1 bias3=-1
```

```
6. w11= -1  w12= 0  w13= -1  w14= -1 w15= 1  w16= -1   w21= 1  w22= -1
bias1=1 bias2=0 bias3=-1
7. w11= -1  w12= 1  w13= -1  w14= -1 w15= -1  w16= -1   w21= -1  w22= 1
bias1=0 bias2=1 bias3=-1
8. w11= -1  w12= 1  w13= -1  w14= -1 w15= 0  w16= -1   w21= -1  w22= 1
bias1=0 bias2=1 bias3=-1
9. w11= -1  w12= 1  w13= 1  w14= -1 w15= -1  w16= 1   w21= -1  w22= 1
bias1=0 bias2=-1 bias3=0
10. w11= -1  w12= 1  w13= 1  w14= 1 w15= 1  w16= -1   w21= -1  w22= -1
bias1=0 bias2=0 bias3=1
11. w11= 1  w12= -1  w13= -1  w14= -1 w15= -1  w16= 1   w21= 1  w22= 1
bias1=-1 bias2=-1 bias3=-1
This solution also corresponds to the direct implementation of the two minterms by the two neurons of the first layer and the OR
of the two minterms by the third neuron
12. w11= 1  w12= -1  w13= -1  w14= 1 w15= 1  w16= -1   w21= 1  w22= -1
bias1=-1 bias2=0 bias3=0
13. w11= 1  w12= 1  w13= -1  w14= -1 w15= -1  w16= 1   w21= -1  w22= -1
bias1=0 bias2=0 bias3=1
14. w11= 1  w12= 1  w13= -1  w14= 1 w15= -1  w16= -1   w21= -1  w22= 1
bias1=0 bias2=-1 bias3=0
```

G of a Run of the Second MILP Model for f=m_1+m_7

```
----     48 VARIABLE weights.L

INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 1  INDEX 8 = 1
             2
2.2 1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 2  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 1  INDEX 8 = 3
             1
2.2 1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 2  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 1  INDEX 8 = 1
             2
2.1 1.000
INDEX 1 = 1  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 3  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 1  INDEX 8 = 3
             1
2.1 1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 1  INDEX 8 = 1
             2
2.2 1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 2  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 1  INDEX 8 = 1
             2
2.1 1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 2  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 1  INDEX 8 = 3
             1
2.2 1.000
INDEX 1 = 1  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 1  INDEX 8 = 3
             1
2.1 1.000
INDEX 1 = 2  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 3  INDEX 8 = 1
             1
2.2 1.000
INDEX 1 = 2  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 1  INDEX 8 = 1
             2
1.2 1.000
INDEX 1 = 2  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 3  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 1  INDEX 8 = 3
             1
1.1 1.000
INDEX 1 = 2  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 1  INDEX 8 = 1
             2
1.2 1.000
INDEX 1 = 2  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 3  INDEX 8 = 1
             1
2.3 1.000
INDEX 1 = 2  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 3  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 1  INDEX 8 = 3
             1
1.1 1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 1  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 3  INDEX 8 = 1
             1
```

```
1.2 1.000
INDEX 1 = 3  INDEX 2 = 1  INDEX 3 = 3  INDEX 4 = 2  INDEX 5 = 1  INDEX 6 = 3
  INDEX 7 = 3  INDEX 8 = 1
             1
1.1 1.000
INDEX 1 = 3  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 1  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 3  INDEX 8 = 1
             1
1.2 1.000
INDEX 1 = 3  INDEX 2 = 3  INDEX 3 = 1  INDEX 4 = 2  INDEX 5 = 3  INDEX 6 = 1
  INDEX 7 = 3  INDEX 8 = 1
             1
1.1 1.000

----     48 VARIABLE obj.L                =       18.000

----     48 PARAMETER f_3in_d

             0           1

1.0       1.000
1.1                   1.000
```

Now does not exist direct solutions, i.e. where the first two neurons implement m1 and m7 and the third neuron implementing the OR of these two minterms, because the implementation of m7 by a perceptron of the first layer with integer weights would imply a bias of -3 and weights equal to 1, i.e. a three input AND, and the weights only vary between -2 and 2.

These 18 solutions correspond to the following sets of values of weights, respectively:

```
1. w11= -1  w12= -1  w13= 1  w14= -1 w15= 1  w16= -1   w21= 1  w22= -1
bias1=0 bias2=0 bias3=0
2. w11= -1  w12= -1  w13= 1  w14= 0 w15= -1  w16= 1   w21= -1  w22= 1
bias1=0 bias2=0 bias3=-1
3. w11= -1  w12= -1  w13= 1  w14= 0 w15= 1  w16= -1   w21= -1  w22= -1
bias1=0 bias2=-1 bias3=0
4. w11= -1  w12= -1  w13= 1  w14= 1 w15= -1  w16= 1   w21= -1  w22= 1
bias1=0 bias2=-1 bias3=-1
5. w11= -1  w12= 1  w13= -1  w14= -1 w15= -1  w16= 1   w21= -1  w22= -1
bias1=0 bias2=0 bias3=0
6. w11= -1  w12= 1  w13= -1  w14= 0 w15= -1  w16= 1   w21= -1  w22= -1
bias1=0 bias2=-1 bias3=0
7. w11= -1  w12= 1  w13= -1  w14= 0 w15= 1  w16= -1   w21= -1  w22= 1
bias1=0 bias2=0 bias3=-1
8. w11= -1  w12= 1  w13= -1  w14= 1 w15= 1  w16= -1   w21= -1  w22= 1
bias1=0 bias2=-1 bias3=-1
9. w11= 0  w12= -1  w13= 1  w14= -1 w15= -1  w16= 1   w21= 1  w22= -1
bias1=0 bias2=0 bias3=-1
10. w11= 0  w12= -1  w13= 1  w14= -1 w15= 1  w16= -1   w21= -1  w22= -1
bias1=-1 bias2=0 bias3=0
11. w11= 0  w12= -1  w13= 1  w14= 1 w15= -1  w16= 1   w21= -1  w22= 1
bias1=-1 bias2=-1 bias3=-1
12. w11= 0  w12= 1  w13= -1  w14= -1 w15= -1  w16= 1   w21= -1  w22= -1
bias1=-1 bias2=0 bias3=0
13. w11= 0  w12= 1  w13= -1  w14= -1 w15= 1  w16= -1   w21= 1  w22= -1
bias1=0 bias2=1 bias3=-1
14. w11= 0  w12= 1  w13= -1  w14= 1 w15= 1  w16= -1   w21= -1  w22= 1
bias1=-1 bias2=-1 bias3=-1
15. w11= 1  w12= -1  w13= 1  w14= -1 w15= -1  w16= 1   w21= 1  w22= -1
bias1=-1 bias2=0 bias3=-1
16. w11= 1  w12= -1  w13= 1  w14= 0 w15= -1  w16= 1   w21= 1  w22= -1
bias1=-1 bias2=-1 bias3=-1
17. w11= 1  w12= 1  w13= -1  w14= -1 w15= 1  w16= -1   w21= 1  w22= -1
bias1=-1 bias2=0 bias3=-1
18. w11= 1  w12= 1  w13= -1  w14= 0 w15= 1  w16= -1   w21= 1  w22= -1
bias1=-1 bias2=-1 bias3=-1
```