Chapter 1

Reification of Boolean Logic

The modern era of neural networks began with the pioneer work of McCulloch and Pitts (1943). McCulloch was a psychiatrist and neuroanatomist; he spent some 20 years thinking about the representation of an event in the nervous system. Pitts was a mathematical prodigy, who joined McCulloch in 1942.

The McCulloch-Pitts model of the neuron is shown in Fig. 1. The input x_i, for i = 1, 2, …, n, are 0 or 1, depending on the absence or presence of the input impulse at instance k. The output signal of this neuron is denoted as o. The firing rule for this model is defined as follows

$$o^{k+1} = \begin{cases} 1 & if \ \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & if \ \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

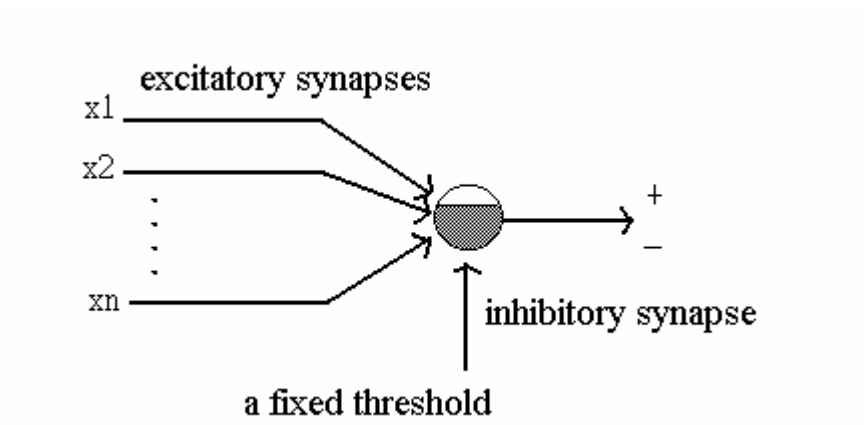


Fig. 1. McCulloch-Pitts neuron model, a binary device (1943).

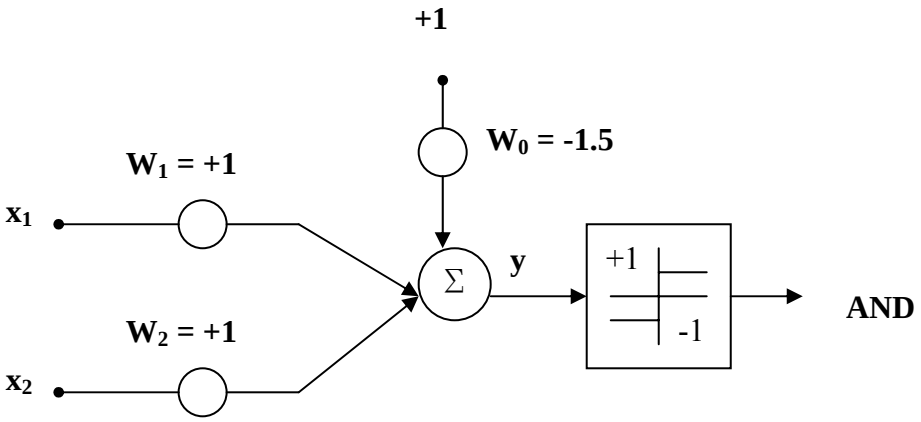
The first paper that refers finite-state machine, AI and recurrent neural network as automaton is written by Kleene (1956).

"Every finite-state machine is equivalent to and can be 'simulated' by some neural net. That is, given any finite-state machine, ***M***, we can build a certain neural net $\boldsymbol{N^M}$ which, regarded as a black-box machine, will behave precisely like ***M***!" by Minsky (1967).

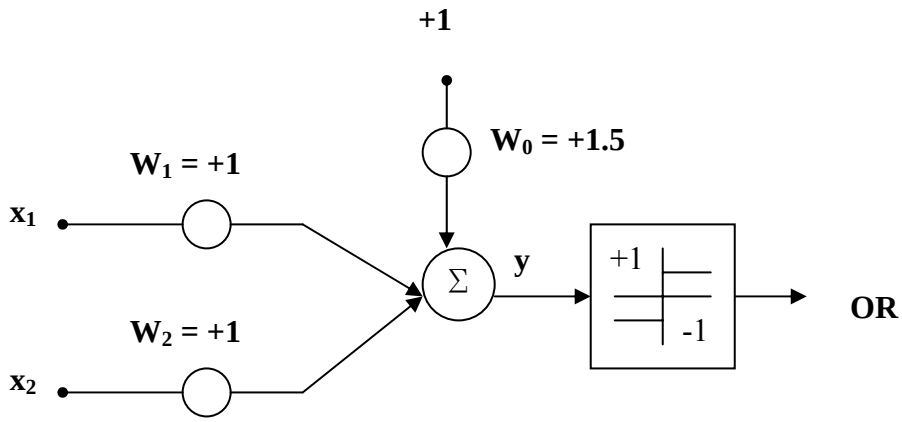
In Kremer (1995) a formal proof is presented that the simple recurrent network has a computation power as great as that of any finite-state machine.

Although this neuron model is very simplistic, it has substantial computing potential. It can perform the basic logic operations **NOT**, **OR**, and **AND**, with appropriately selected weights and thresholds. As we know, any multivariable combinational function can be implemented using either the **NOT** and **OR**, or alternatively the **NOT** and **AND**, Boolean operations.

(a)



(b)



(c)

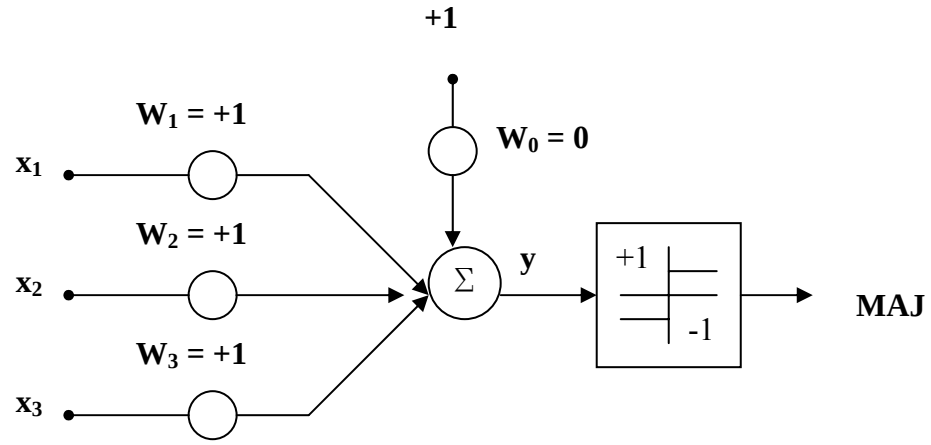


Fig. 2(a)~(c). Neuronal implementation of **AND**, **OR**, and **MAJ** logic functions.

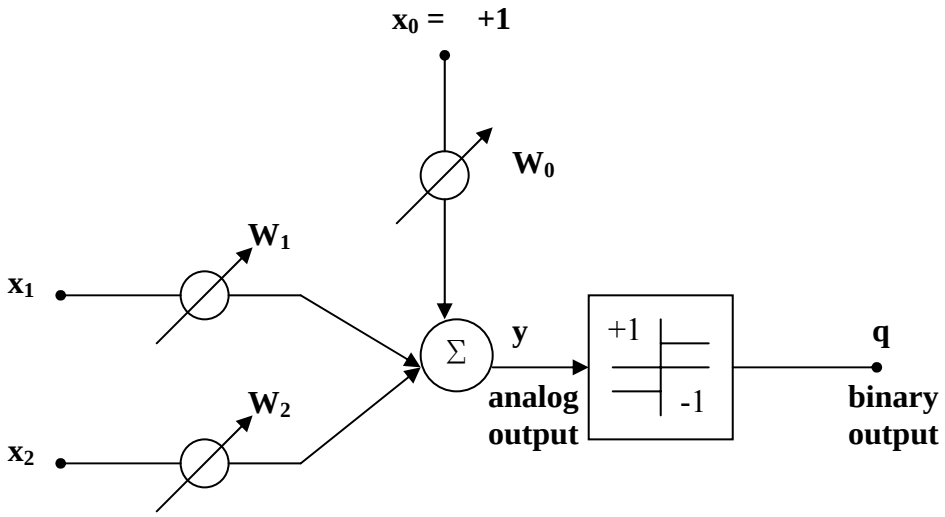


Fig. 3. A two-input neuron.

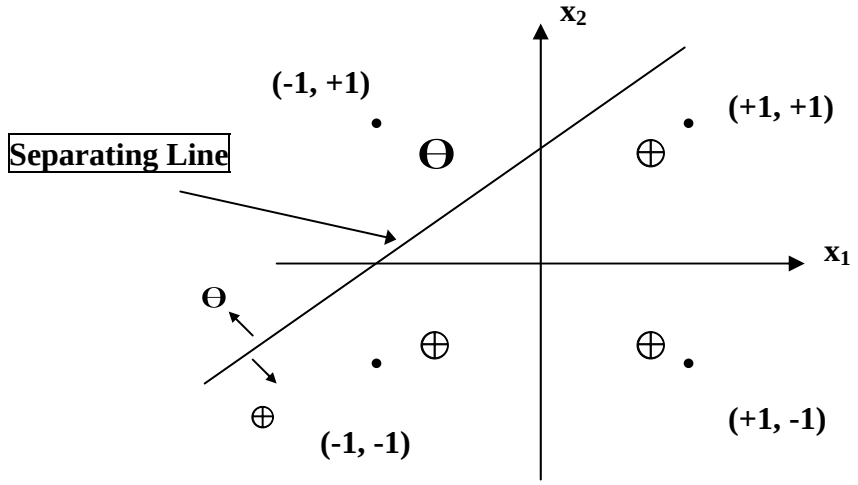
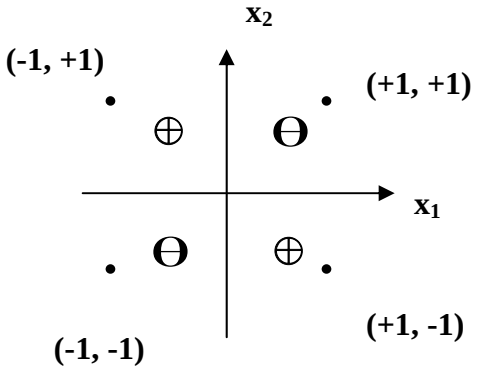


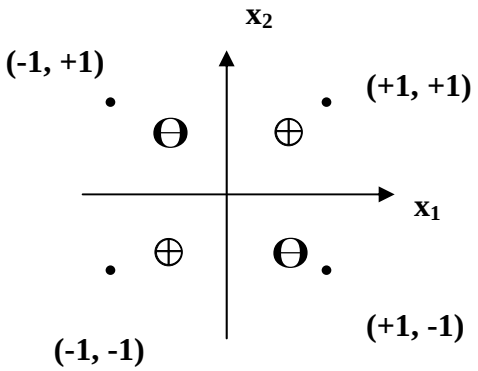
Fig. 4. Separating line in pattern space.

Fig. 3 is a two-input neuron. In pattern space, the neuron can be represented as a separating hyperplane (see Fig. 4). By adjusting the position of the separating line, a single neuron can simulate 14 Boolean functions, except **XOR** and **XNOR** (see Fig. 5).



x_1	x_2	XOR
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Fig. 5(a). The pattern space and truth table of **XOR**.



x_1	x_2	XNOR
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

Fig. 5(b). The pattern space and truth table of **XNOR**.

We need a single hidden layer with two neurons to solve the **XOR** problem.

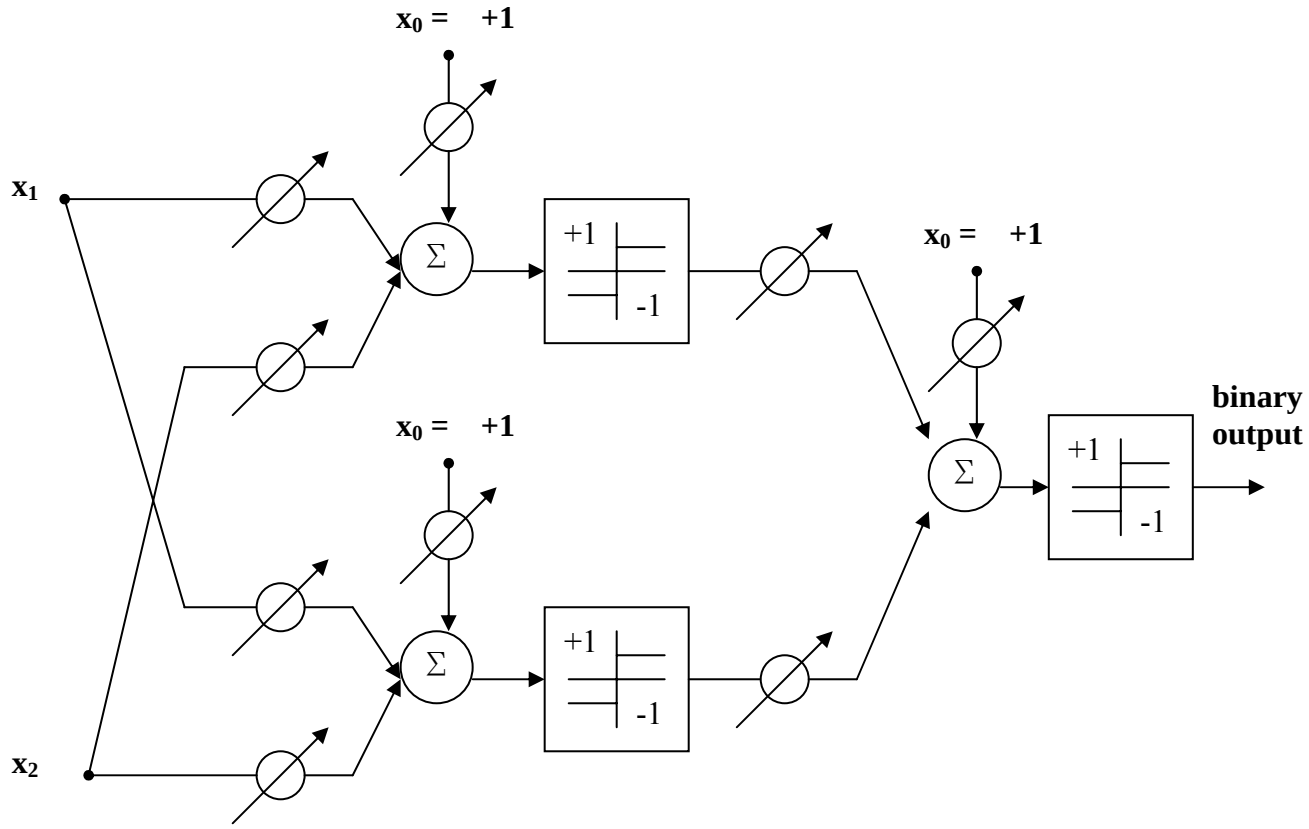


Fig. 6(a). The form of a single hidden layer of two neurons and one output neuron.

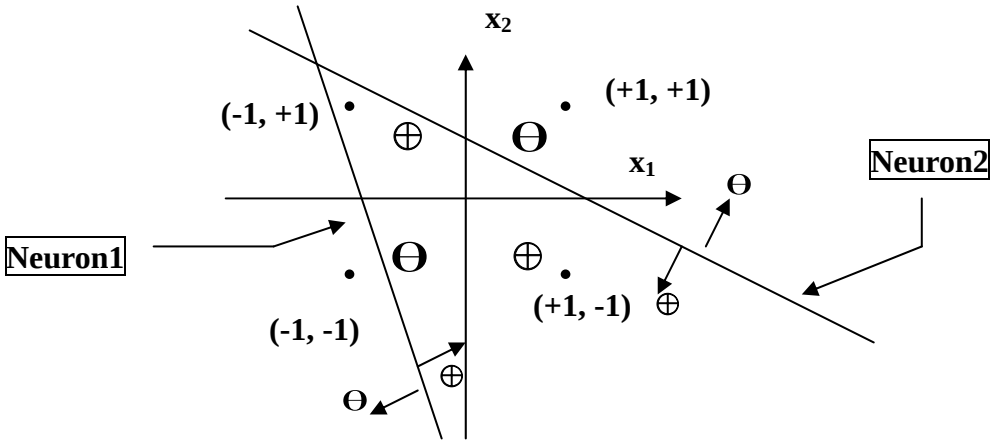


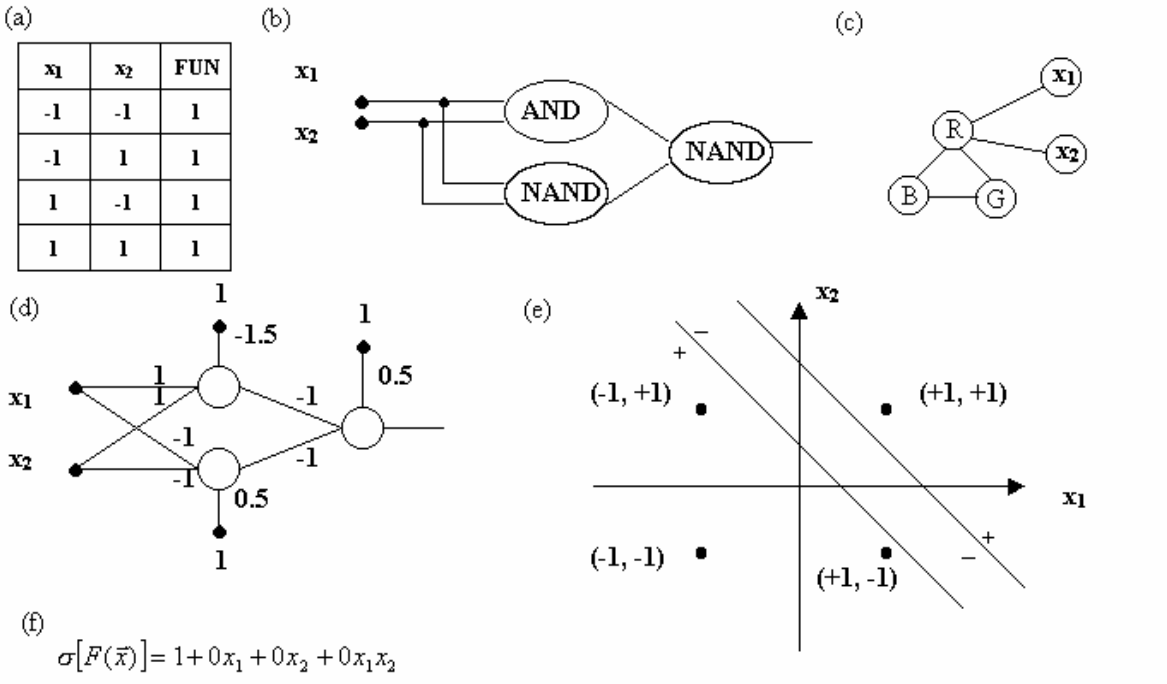
Fig. 6(b). Separating lines in pattern space of **XOR**.

We list eight kinds of two-input Boolean function for reference and other functions can be obtained by transfer the sign or inferred from the tips we have.

1.

(a)

x_1	x_2	FUN
-1	-1	1
-1	1	1
1	-1	1
1	1	1



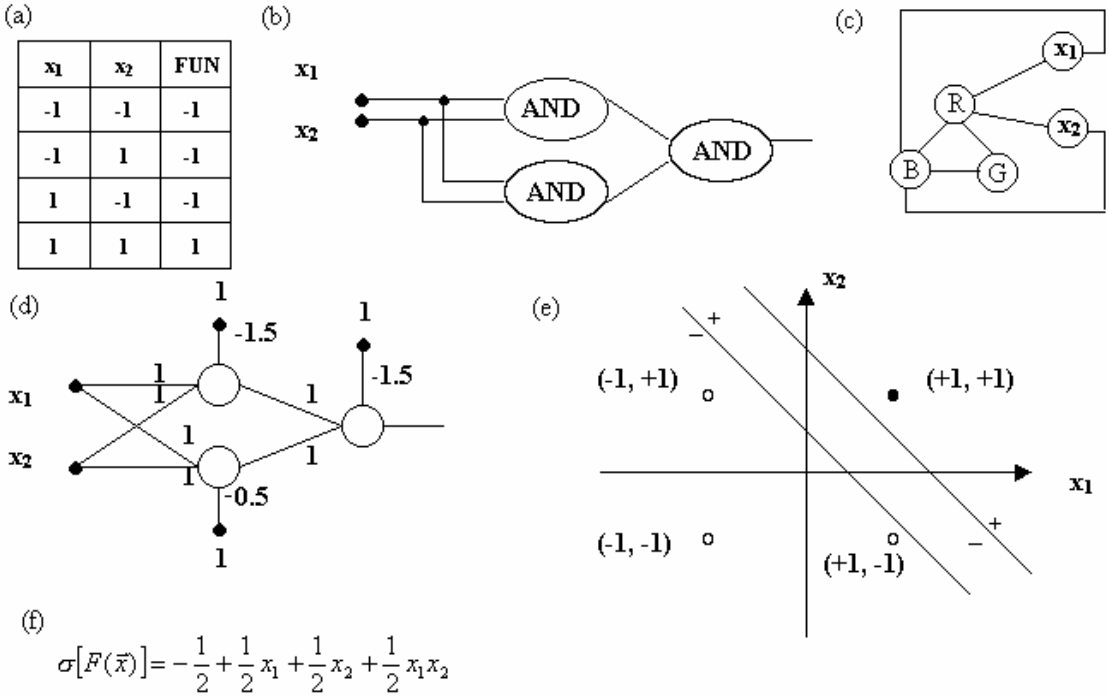
(f)

$$\sigma[F(\vec{x})] = 1 + 0x_1 + 0x_2 + 0x_1x_2$$

2.

(a)

x_1	x_2	FUN
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1



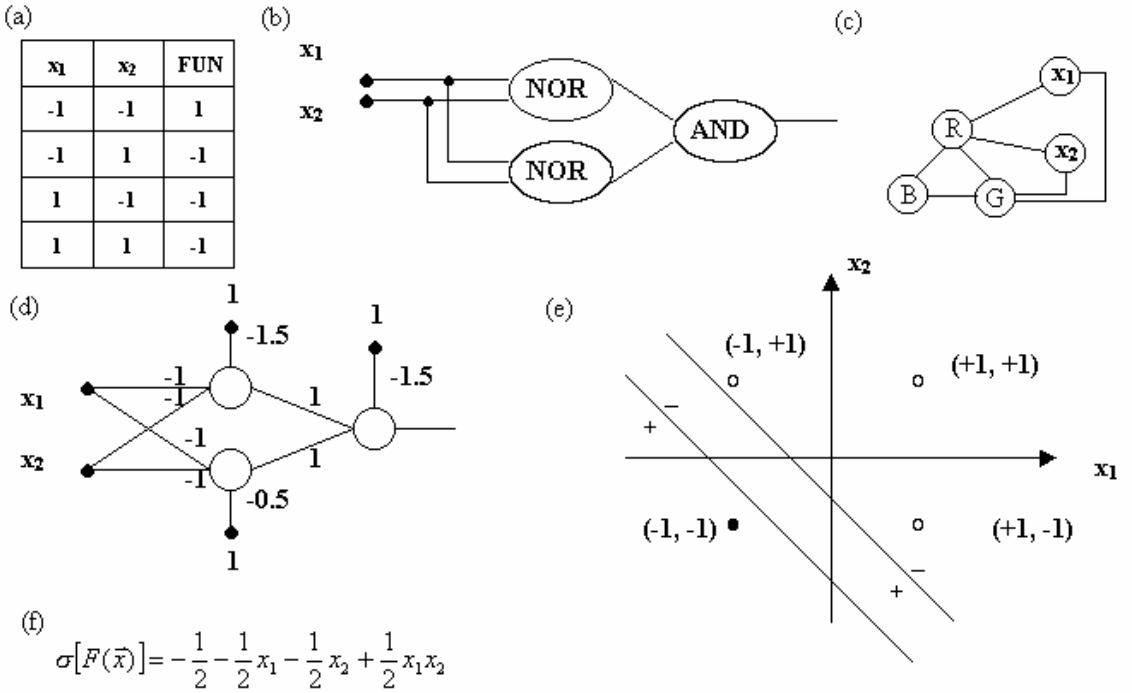
(f)

$$\sigma[F(\vec{x})] = -\frac{1}{2} + \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_1x_2$$

3.

(a)

x_1	x_2	FUN
-1	-1	1
-1	1	-1
1	-1	-1
1	1	-1



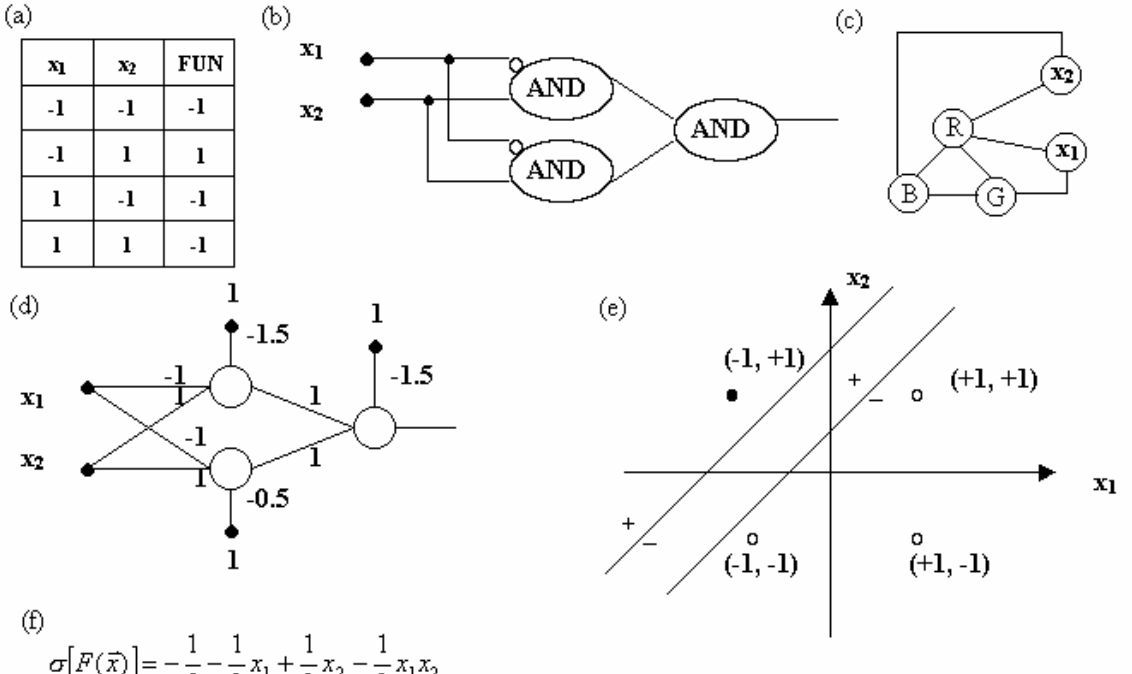
(f)

$$\sigma[F(\vec{x})] = -\frac{1}{2} - \frac{1}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_1x_2$$

4.

(a)

x_1	x_2	FUN
-1	-1	-1
-1	1	1
1	-1	-1
1	1	-1



(f)

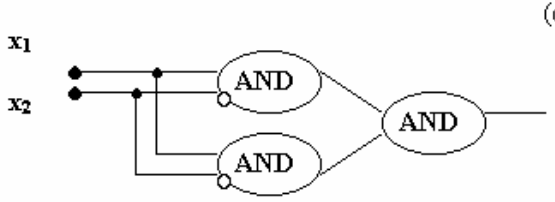
$$\sigma[F(\vec{x})] = -\frac{1}{2} - \frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_1x_2$$

5.

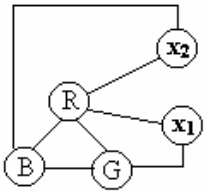
(a)

x_1	x_2	FUN
-1	-1	-1
-1	1	-1
1	-1	1
1	1	-1

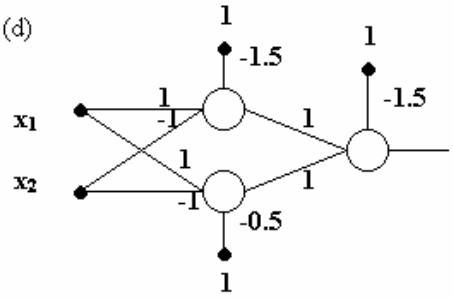
(b)



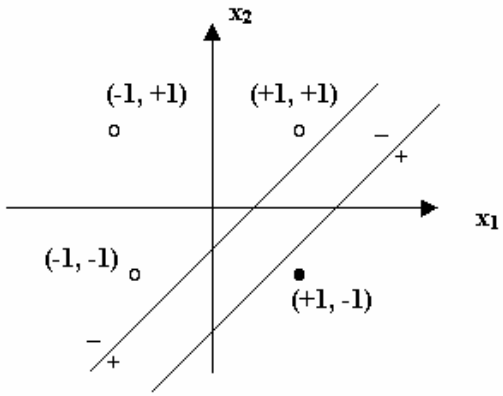
(c)



(d)



(e)



(f)

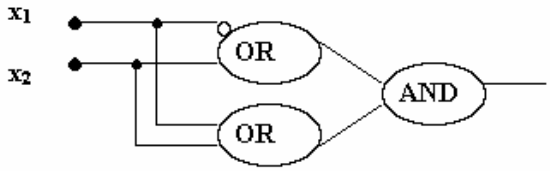
$$\sigma[F(\vec{x})] = -\frac{1}{2} + \frac{1}{2}x_1 - \frac{1}{2}x_2 - \frac{1}{2}x_1x_2$$

6.

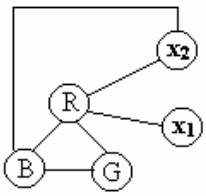
(a)

x_1	x_2	FUN
-1	-1	-1
-1	1	1
1	-1	-1
1	1	1

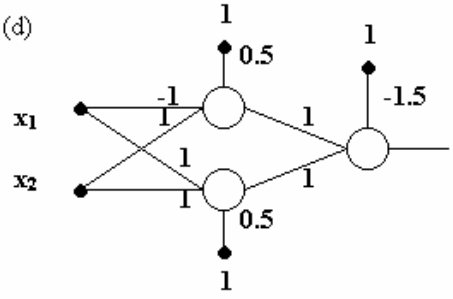
(b)



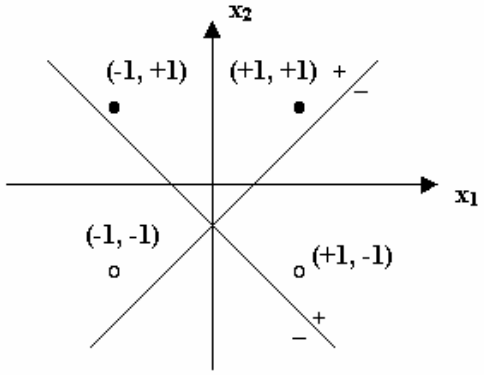
(c)



(d)



(e)



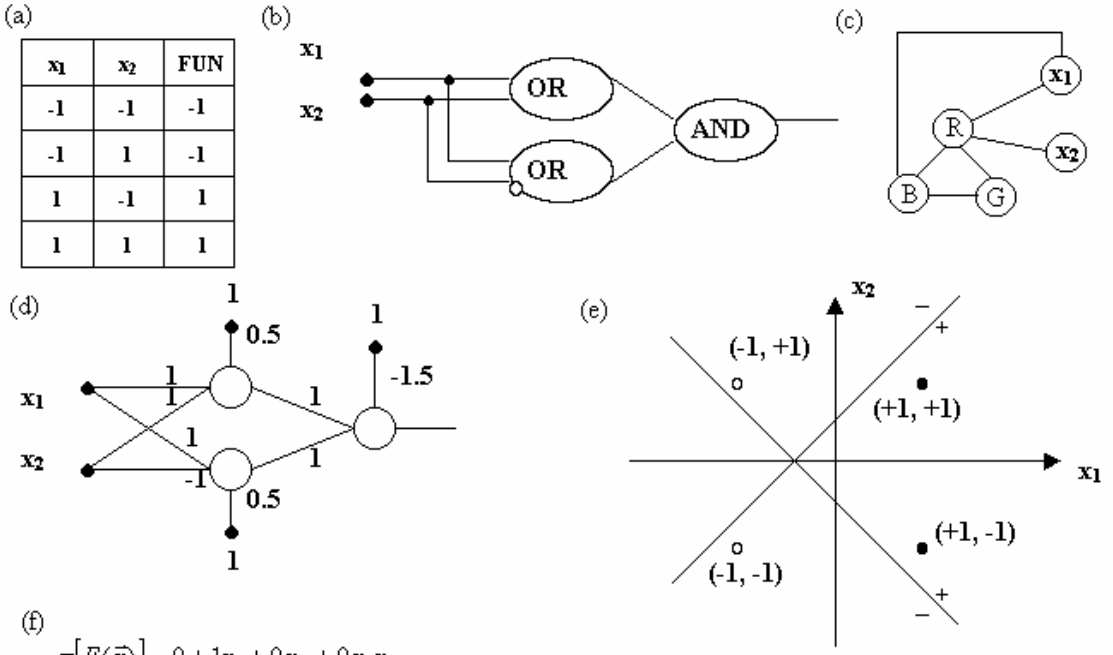
(f)

$$\sigma[F(\vec{x})] = 0 + 0x_1 + 1x_2 + 0x_1x_2$$

7.

(a)

x_1	x_2	FUN
-1	-1	-1
-1	1	-1
1	-1	1
1	1	1



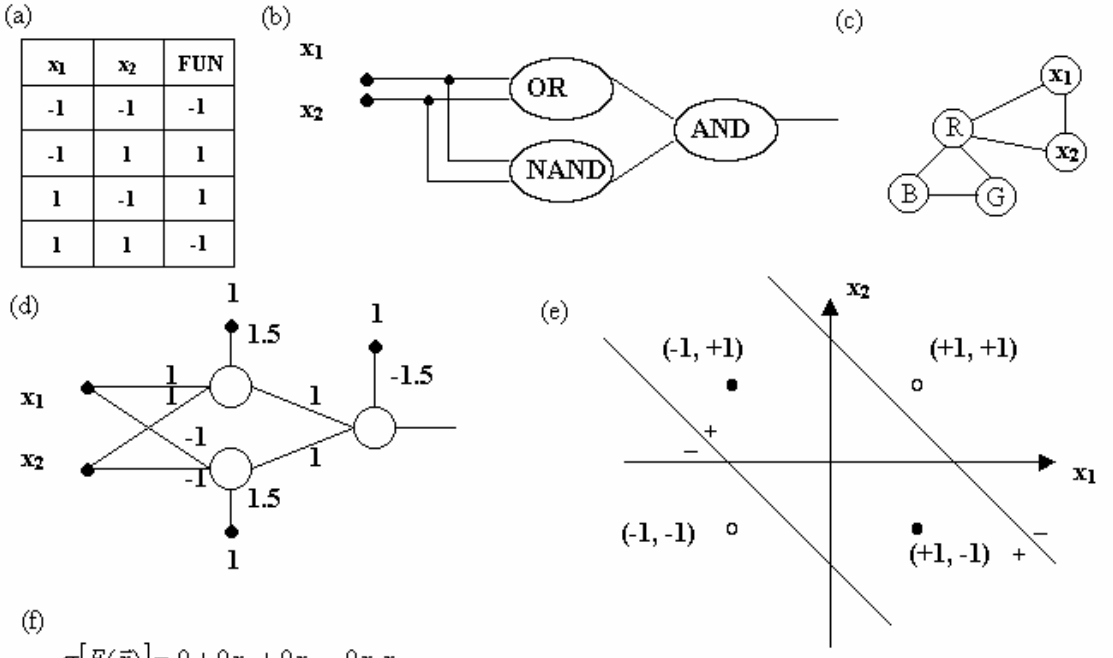
(f)

$$\sigma[F(\vec{x})] = 0 + 1x_1 + 0x_2 + 0x_1x_2$$

8.

(a)

x_1	x_2	FUN
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



(f)

$$\sigma[F(\vec{x})] = 0 + 0x_1 + 0x_2 - 0x_1x_2$$

In above figures:

(a) Truth Table.

(b) Logic Gates.

(c) Graph Boolean function.

(d) Neural Networks.

(e) Geometrical Perspective on Neural Networks.

(f) η-expansion.

The graph Boolean function is using graph to represent Boolean function. In this graph, if no two adjacent nodes have the same color the output of this function is true, otherwise is false. If x_i takes the color Blue it means that the variable is false, color Green is true, and color Red is don't care. The η-expansion will be explained in chapter 7 and chapter 8.

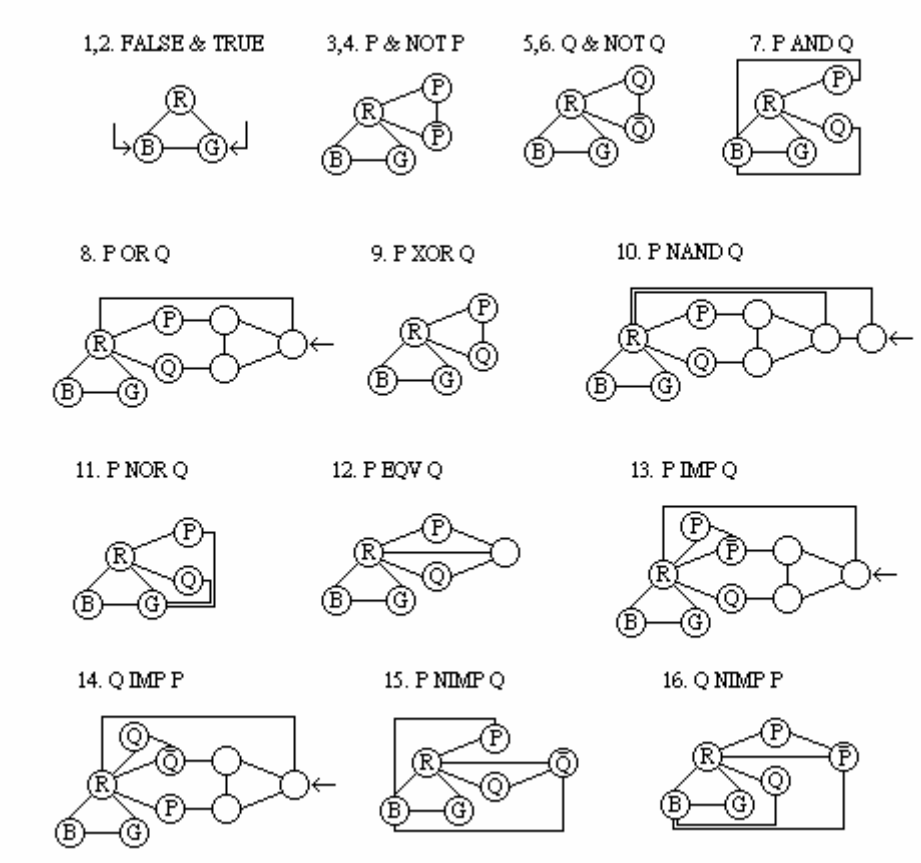


Fig. 7. The Graph Boolean functions

One neuron can simulate 14 Boolean functions, and we use three neurons to handle **XOR** and **NXOR** problem. The three neurons have $14^3 = 2744$ combinations. However, there are only 16 Boolean functions. Most combinations are duplications.

X_1	X_2	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Logic symbol			·	⁄		⁄		⊕	+	↓	⊙	´	⊂	´	⊃	↑	

Table 1. The inputs, outputs, and logic symbols of all 16 Boolean functions.

F_0	**Null**	F_8	**NOR**
F_1	**AND**	F_9	**Equivalence**
F_2	**Inhibition**	F_{10}	**Complement**
F_3	**Transfer**	F_{11}	**Implication**
F_4	**Inhibition**	F_{12}	**Complement**
F_5	**Transfer**	F_{13}	**Implication**
F_6	**Exclusive-OR**	F_{14}	**NAND**
F_7	**OR**	F_{15}	**Identity**

Table 2. The names of all 16 Boolean functions.

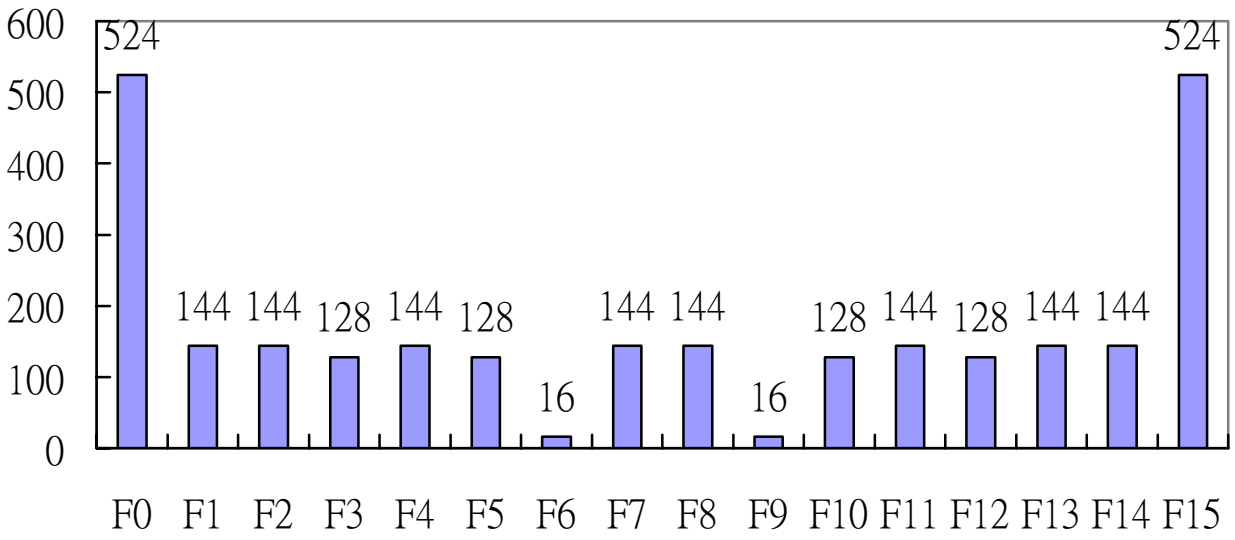


Table 3. The numbers of combinations which make the same Boolean functions.

In three dimensions, binary input patterns occupy the corners of 3-cube. We have 104 methods by using one hyperplane to cut those 8 patterns into two classes.

(1) 0 : 8, 1*2 = 2 methods.

(2) 1 : 7, 8*2 = 16 methods.

(3) 2 : 6, 12*2 = 24 methods.

(4) 3 : 5, 8*3*2 = 48 methods.

(5) 4 : 4, 3*2 + 8 = 14 methods.

So, one neuron can simulate 104 Boolean functions with three dimensions inputs. We try to use the same method to generate all 256 Boolean functions. But we find that using only three neurons (104*104*14) can not handle functions $x_1 \oplus x_2 \oplus x_3$ and $\sim(x_1 \oplus x_2 \oplus x_3)$.

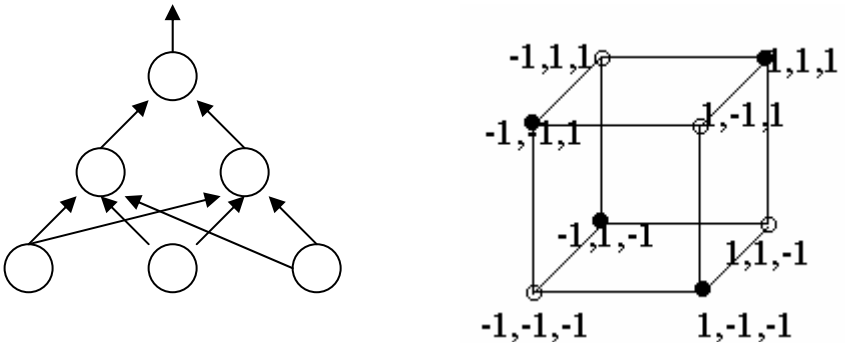


Fig. 2. The 3-2-1 neural networks and the Boolean function this network can't handle.

We reuse 3-3-1 architecture (104^4 combinations) to generate 256 Boolean functions and construct the following table.

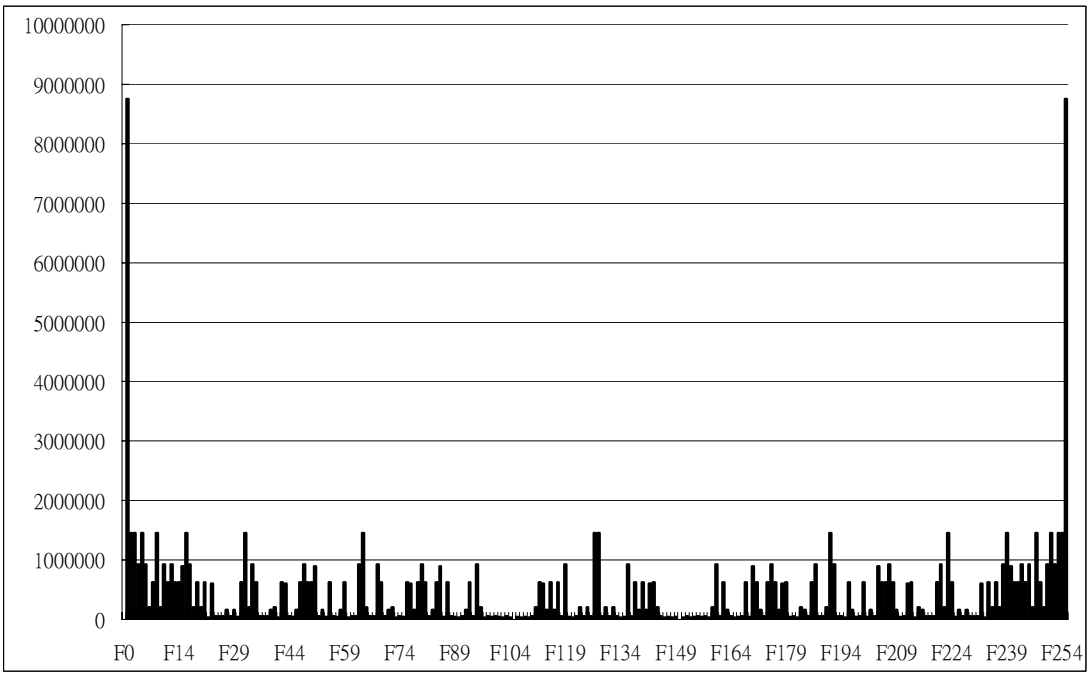


Table 4. The numbers of combinations which make the same Boolean functions with three dimension inputs.

F0	8751200
F1	1457568
F2	1457568
F4	1457568
F8	1457568
F16	1457568
F32	1457568
F64	1457568
F127	1457568
F3	926736
F5	926736
F10	926736
F12	926736
F17	926736
F34	926736
F48	926736
F63	926736
F68	926736
F80	926736
F95	926736
F119	926736
F15	888864
F51	888864
F85	888864
F7	628368
F11	628368
F13	628368
F14	628368
F19	628368
F21	628368
F31	628368
F35	628368
F42	628368

F47	628368
F49	628368
F50	628368
F55	628368
F59	628368
F69	628368
F76	628368
F79	628368
F81	628368
F84	628368
F87	628368
F93	628368
F112	628368
F115	628368
F117	628368
F23	601872
F43	601872
F77	601872
F113	601872
F6	205920
F9	205920
F18	205920
F20	205920
F33	205920
F40	205920
F65	205920
F72	205920
F96	205920
F111	205920
F123	205920
F125	205920
F27	157344
F29	157344

F39	157344
F46	157344
F53	157344
F58	157344
F71	157344
F78	157344
F83	157344
F92	157344
F114	157344
F116	157344
F24	58368
F36	58368
F66	58368
F126	58368
F25	56112
F26	56112
F28	56112
F37	56112
F38	56112
F44	56112
F52	56112
F56	56112
F61	56112
F62	56112
F67	56112
F70	56112
F74	56112
F82	56112
F88	56112
F91	56112
F94	56112
F98	56112
F100	56112

F103	56112
F110	56112
F118	56112
F122	56112
F124	56112
F30	43968
F45	43968
F54	43968
F57	43968
F75	43968
F86	43968
F89	43968
F99	43968
F101	43968
F106	43968
F108	43968
F120	43968
F22	31200
F41	31200
F73	31200
F97	31200
F104	31200
F107	31200
F109	31200
F121	31200
F60	28512
F90	28512
F102	28512
F105	3360

Table 5. The detailed data of Table 4. F128~F255 are the same as F127~F0.

So far, we observe how algebra, geometry, linear algebra, Boolean algebra, and neural networks combined together. Neural networks utilize geometry and linear algebra to solve the problem of Boolean algebra. In Boolean algebra, we tend to use + • ~ → ≡ … those operator to solve problems. In neural networks, we use hyperplane to cut space to find the solution. For example, if we want to implement this truth table:

A	B	C	D	F
0	0	0	0	0
		0	1	0
		1	0	0
		1	1	0
0	1	0	0	0
		0	1	0
		1	0	0
		1	1	1
1	0	0	0	0
		0	1	0
		1	0	1
		1	1	1
1	1	0	0	0
		0	1	1
		1	0	1
		1	1	1

We may write down

$$F = \bar{A}BCD + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}D + ABC\bar{D} + ABCD,$$

and use Boolean theorems to simplify this expression or use K-map method directly to get

F = AC + BCD + ABD.

Then use logic gate to implement this Boolean function.

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	1

The K-map.

(You can find more information about Gate Logic or K-map Method from “Contemporary Logic Design”, written by Randy H. Katz, or other logic design books.)

On the other hand, we use all the information from truth table (as training patterns) to find a hyperplane that separates patterns correctly. The hyperplane we find is 7.1651*A + 3.6203*B + 7.1653*C + 3.6203*D - 3.6196 = 0 (we use –1 in place of 0 to train the networks). Note that if patterns are not specified in truth table, their value can be either +1 or 0 (-1) for us to obtain the applicable Boolean expression.

We may say the neural network have the global sense, because it uses information from all patterns to find the decision hyperplane. However, Boolean algebra must parse the truth table one by one and use + • ~ → ≡ … these operators we are familiar with to construct a Boolean expression. The Boolean function corresponding to the hyperplane in neural networks may be strange, and hard for us to understand. On the other hand, it has the freedom to find the solution without the limit

of + • ~ → ≡.

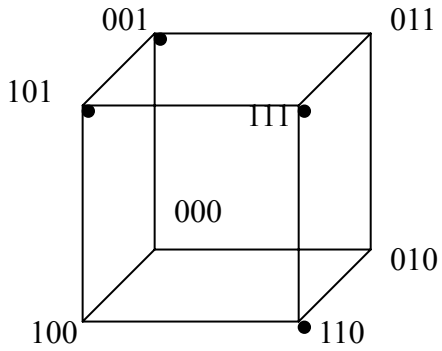
The variables x1 xi and operators + ~ can express all Boolean functions. For one neuron, if it has two inputs, it can't implement two functions - XOR and XNOR; if it has three inputs, it can't implement 152 functions in all 256 Boolean functions. As input dimensions increase the capability of one neuron is decreasing (2D: $\frac{2}{16} = \frac{1}{8}$, 3D: $\frac{152}{256} > \frac{1}{2}$). We may consider that Boolean algebra is more powerful than neural networks. Compare neuronal model equation $y = \sigma(\sum w_i x_i + w_0)$ with Boolean expressions only using operators + and ~:

$Y = a_1X_1 + a_2X_2 + a_3X_3 + \ldots. + a_nXn$, $a_i = \sim$ or empty string, $X_i = 1$ or 0.

Notice that X_i only appears once in above equation. The neuronal model equation can replace Boolean expression by $w_0 = 0$, $w_i = +1$ or -1, $\sigma(x) = x$ and $x_i = 1$ or 0. Furthermore, w_i is real number and function $\sigma(\bullet)$ can be any proper function. So, $y = \sigma(\sum w_i x_i + w_0)$ is more powerful than $Y = a_1X_1 + a_2X_2 + a_3X_3 + \ldots. + a_nXn$, $a_i = \sim$ or empty string, $X_i = 1$ or 0. On the other hand, the above Boolean expression can't construct all Boolean functions without adding the nest form. Comparatively, neural networks need multilayer constructions to implement all Boolean functions. From above, neural networks are more powerful than Boolean algebra.

Exercises

1.1 (a) Design a feedforward network to divide the black dots from other corners with fewest neurons and layers. Please specify the values of weights and thresholds.



(b) Is it possible to do (a) with a single neuron? Why?

1.2 Consider the neural network in Fig. P1. The value of each neuron can be 0 or 1 and the activation function used is f (net) = {1, net>0; 0, net<0}. Each neuron decides its own value according to the values of neighbors. We can adjust the weights of neurons so that every neuron will have distinct set of action rules (e.g. {000|0, 001|1, …, 111|0} is a set of action rules). If two neurons that have different sets of action rules are considered to be different kinds, then, how many kinds of neurons can we have in the network?

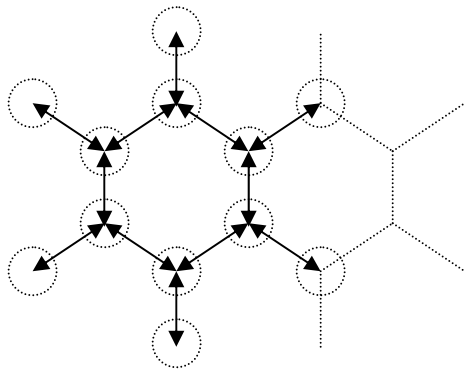


Fig. P1.

1.3 Design a feedforward network which provides the classification of the following pattern vectors:

Class 1:

$X = [0\ 0\ 0\ 0]^t, [0\ 0\ 1\ 0]^t, [0\ 0\ 1\ 1]^t, [0\ 1\ 0\ 0]^t, [0\ 1\ 0\ 1]^t, [0\ 1\ 1\ 0]^t, [0\ 1\ 1\ 1]^t, [1\ 0\ 1\ 0]^t, [1\ 1\ 0\ 0]^t, [1\ 1\ 1\ 0]^t, [1\ 1\ 1\ 1]^t$

Class 2:

$X = [0\ 0\ 0\ 1]^t, [1\ 0\ 0\ 0]^t, [1\ 0\ 0\ 1]^t, [1\ 0\ 1\ 1]^t, [1\ 1\ 0\ 1]^t$

Please specify the value of weights and thresholds and use as few neurons and layers as possible.

1.4 Please refer to p.217 in "Introduction to Artificial Neural Systems" by Jacek M. Zurada.

$$M(J, n) = 2^J,\ n \geqq J.$$

Is it enough to use J neurons as hidden layer to represent all 2^{2^n} Boolean functions when n = J?

1.5 To compare neural model equation $y = \sigma(\Sigma\ w_i x_i + w_0)$ with Boolean equations which use only operators $\cap$, $\cup$ and ~.

(1) List all different Boolean equations in the forms as follows.

(a) $Y = a_1X_1 \cup a_2X_2 \cup a_3X_3$.

(b) $Y = a_1X_1 \cap a_2X_2 \cap a_3X_3$.

(c) $Y = a_1X_1 \cup a_2X_2 \cap a_3X_3$.

(d) $Y = a_1X_1 \cap a_2X_2 \cup a_3X_3$.

(a_1, a_2, a_3 can be ~ or empty string.)

And specify which equations appear in neural model's 104 functions, which didn't. (Hint: You can use the 3-cube to represent Boolean equations)

(2) We have five methods to cut the cube to get the 104 functions. For each cutting method, write a corresponding Boolean equation in the forms given above.

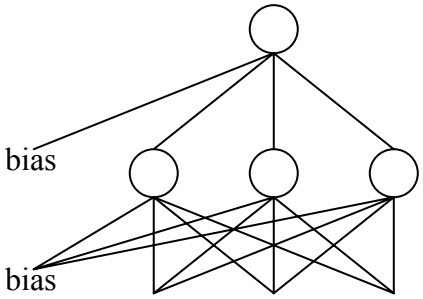
1.6 We have these training patterns:

0: (0, 0, 0), (0, 1, 1), (1, 1, 0)

1: (0, 1, 0), (1, 0, 0), (1, 1, 1).

The training result:

$$W_1 = \begin{bmatrix} -0.9 & 1.5 & -1.0 \\ 2.5 & -3.7 & 0.37 \\ 2.5 & -2.0 & 4.0 \end{bmatrix}$$



$W_2 = \begin{bmatrix} 5 & 6 & 6 \end{bmatrix}$

$Bias_1 = \begin{bmatrix} -0.95 & -2.0 & -3.1 \end{bmatrix}$

$Bias_2 = -3$

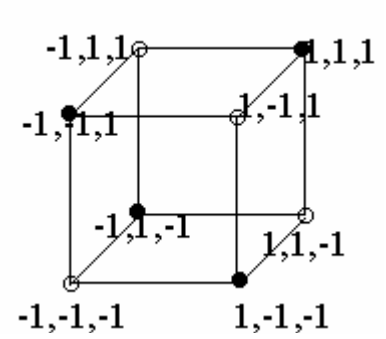
(1) Write the Boolean function of training patterns. (Make simplification)

(2) Write the nested Boolean function of the network architecture for each neuron. Notice that the Boolean function you wrote must be in the forms given in problem 1.

(3) Prove that function (1) and (2) are equivalent.

(4) (0, 0, 1) and (1, 0, 1) didn't appear in training patterns. So, we can get four different Boolean functions. Please choose the simplest Boolean function and compare it with the output of the above neural network of these two patterns.

1.7 Training patterns:

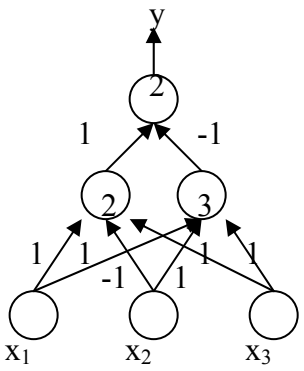


(1) Write the Boolean function. (You must make simplification)

(2) Design a 3-2-1 neural network to classify these patterns. If it cannot classify these patterns correctly, construct your own neural network to classify these patterns.

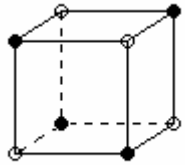
1.8 Boolean functions and neural network have a one to one map in architecture. Please find the map and make a description of it.

Hint: $y = (x_1 \wedge \neg x_2 \wedge x_3) \vee \neg(x_1 \wedge x_2 \wedge x_3)$ can map to this neural network



1.9 Find the coloring graph solution for $x_1 \oplus x_2 \oplus x_3 = Y$.

Remember $x_1 \oplus x_2 \oplus x_3 = Y$ is,



coloring graph is similar to that in page 79 "Turing Machines" by J. E. Hopcroft, 70-80.

1.10 Proposition: The Boolean function is defined as: there are six Boolean variables as input (x1, x2, …, x6) of this function, the output of this function is 1 only when any two variables are 1, (more than two variables or less than two variables are 1, the output of this function is 0.)

(a) Use Boolean algebra or Boolean function to express this proposition.

(b) Write a program (Turing machine, Lisp, C, or other programs) to simulate this expression, the input of the program is these six Boolean variables, the output of the program is according to the proposition.

(c) Discuss the trained weights of the multilayer network in homework for the above proposition. Can you figure out the meaning of those weights?

(d) Construct a multilayer network for this proposition. Use six neurons in input layer and one neuron in output layer.

1.11 In class we discussed a neuron with two Boolean variables as input. This neuron can simulate 14 Boolean functions (14 Boolean states) excluding the XOR. Assume the neuron is in state Si, which is one of the 14 Boolean functions. When we slightly tune the weights w1 w2 w3 of this neuron the current state Si will change to Sj first. Discuss all possible such first Sj when this neuron is in Si for all 14 states, i = 1~14.

1.12 Write an algorithm to train the neural network in problem 2 by its training

patterns. Notice that the algorithm must be constructed by the following subroutine:

train_one_neuron(x, d, w, y, Δw)

input: x, d, w. output: y, Δw.

x: an array of the input value.

d: a desire output.

w: the weights.

y: the output.

Δw: an array of the value of the weights should be added.

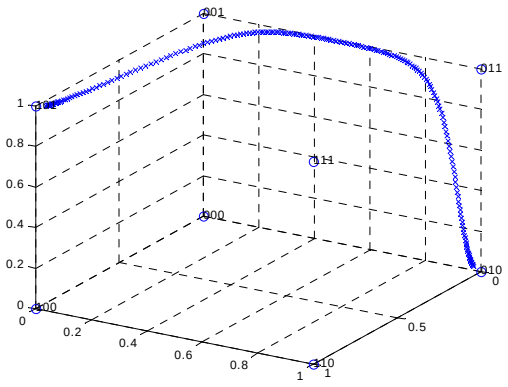
1.13 In appendix A we showed a modular design method for training ANN on data flow machine. Each neuron's desire response

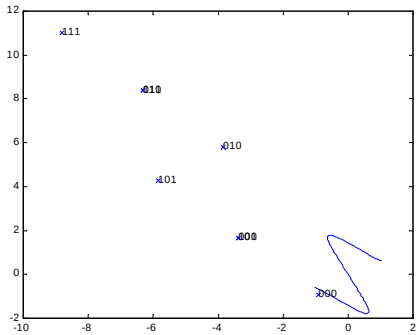
$$d_j^l = y_j^l + \sum_{i=1}^{m_{l+1}} \frac{\Delta w_{ij}^{l+1}}{\eta y_i^l} w_{ij}^{l+1}, \qquad \{l+1 = 2....L\}.$$

is inducted from choosing bipolar sigmoid function $\sigma(u) = \frac{2}{1+e^{-u}} - 1$. Please try to formalize the desire response when we choose unipolar sigmoid function

$$\sigma(u) = \frac{1}{1+e^{-u}}$$

1.14 Discuss and analyze the results obtained in training a 1-3-2 ANN. You can use the following results or your own results. The input is a continues number between 0 to 1 and output is a 'S' curve in the 2D plane.





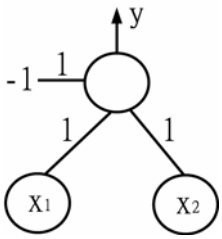
1.15 (a) A dataflow information processing architecture is a MIMD architecture without global or shared memory in which each processing element only operates when all of the necessary information that it needs to function has arrive. Show that neural networks are dataflow architectures.

(b) Invent a concrete, detailed example of a dataflow architecture that is not a neural network.

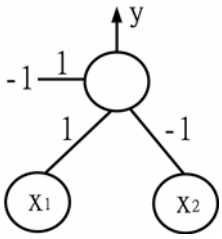
1.16 We have learned basic type of neural network:

$$y = f(\sum_{i=1}^{n} w_i x_i + \theta)$$

We can directly use it to simulate Boolean logic. For example: OR Boolean function: $y=x_1$ OR x_2, where x_1, x_2, y are Boolean variables having values {0,1}.



When above network use 'hardlim' active function, its output $y=x_1$ OR x_2.
Another example is AND Boolean function: $y=x_1$ AND $\neg x_2$



Please check the truth table for it by yourself and answer questions below:
(a) Draw the neural network that perform:

$$y = \left(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4\right)$$

Note: the weights and bias should be {-1,1} for simplicity.

(b) Draw the neural network that peform:

$$y = \left(x_1 \wedge \neg x_2 \wedge x_3 \wedge x_4\right)$$

Note: the weights and bias should be {-1,1} for simplicity.

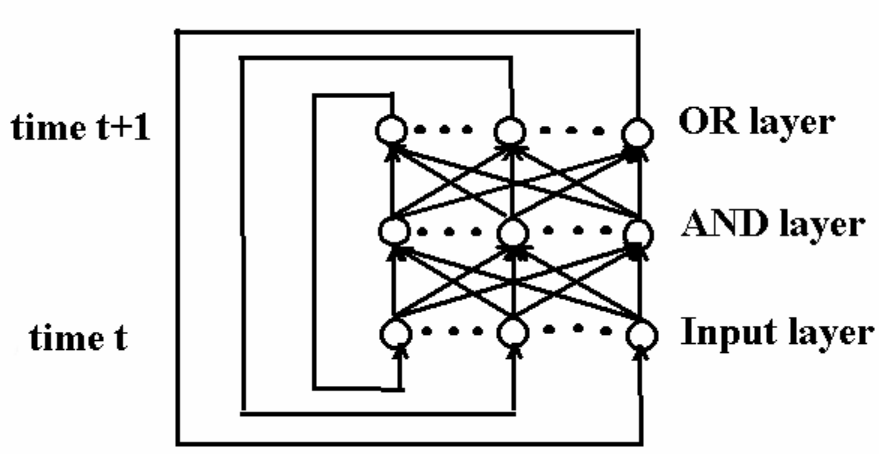
(a) Can you formalize how to set up the weights and biases for a given OR expression?

(b) Can you formalize how to set up the weights and biases for a given AND expression?

(c) Try to use a 3-layers neural network for simulating the DNF equation:

$$y = \left(x_1 \wedge \neg x_2 \wedge x_3\right) \vee \neg\left(x_3 \wedge x_4 \wedge x_5\right)$$

(f) The neural network's outputs can be feedback as inputs like:



It can simulate many kinds of dynamic process like gene regulation and etc. If n Boolean variables $x_1, x_2, \ldots, x_n$ are changed by time and its value are known for a period of time. We can build ANN model like above for these n-variables. Please try to build an ANN model for the given 4-variables.

Time	x_1	x_2	x_3	x_4
1	1	0	1	1
2	0	1	0	0
3	0	0	0	1
4	1	1	1	1
5	0	1	1	0
6	1	1	0	0
7	1	0	0	0
8	0	1	0	1

(Hint: first rewrite $x_1 = F(x_1, x_2, x_3, x_4)$ and get its DNF)