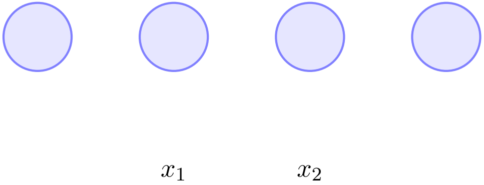
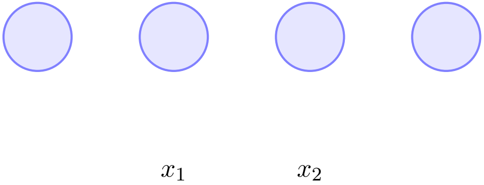
Module 2.8: Representation Power of a Network of Perceptrons

- We will now see how to implement **any** boolean function using a network of perceptrons ...

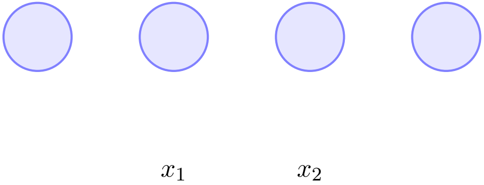
- For this discussion, we will assume True = +1 and False = -1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons





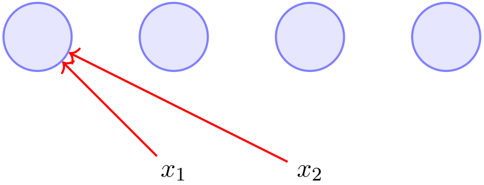
- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights



red edge indicates w = -1
blue edge indicates w = +1

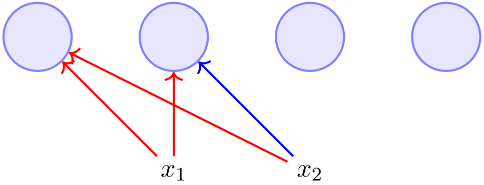
- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights



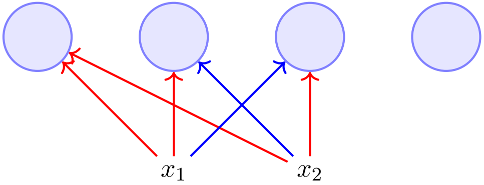
red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights



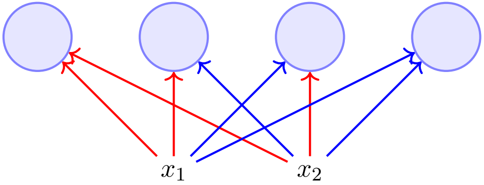
red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights

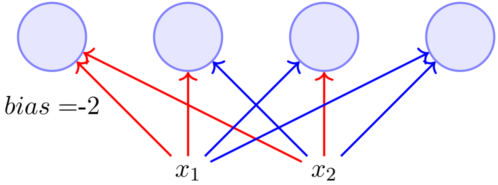


red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights

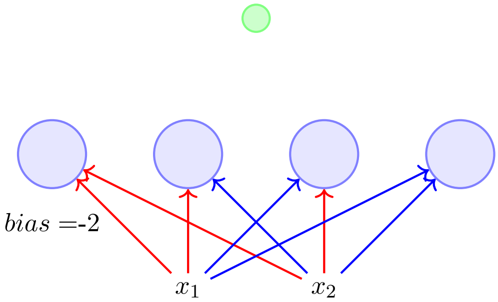


red edge indicates w = -1
blue edge indicates w = +1



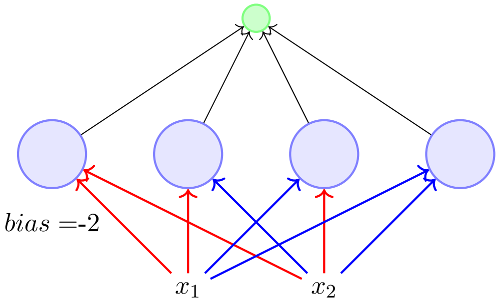
red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is $\geq$ 2)



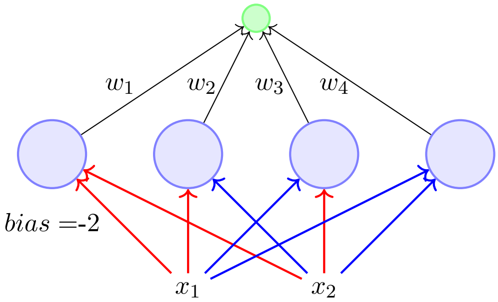
red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)



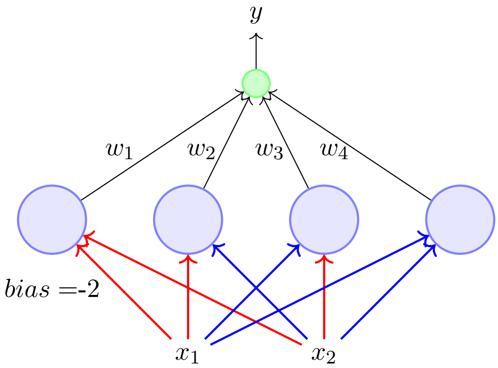
red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)



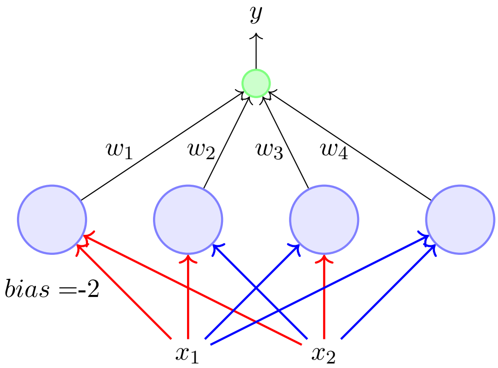
red edge indicates w = -1
blue edge indicates w = +1

- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)



red edge indicates w = -1
blue edge indicates w = +1

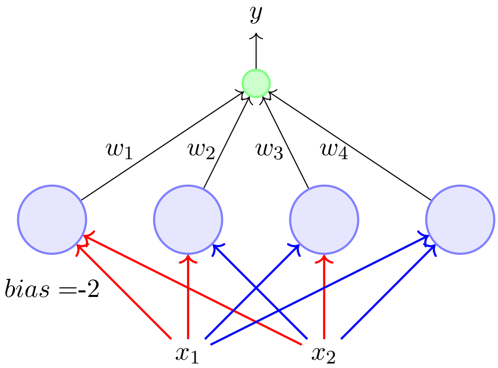
- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)
- The output of this perceptron (y) is the output of this network



red edge indicates w = -1
blue edge indicates w = +1

Terminology:

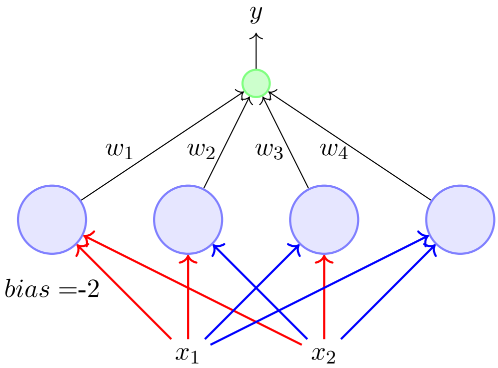
- This network contains 3 layers



red edge indicates w = -1
blue edge indicates w = +1

Terminology:

- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**

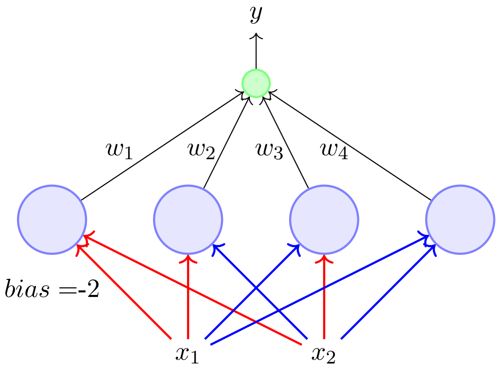


red edge indicates w = -1

blue edge indicates $w = +1$

Terminology:

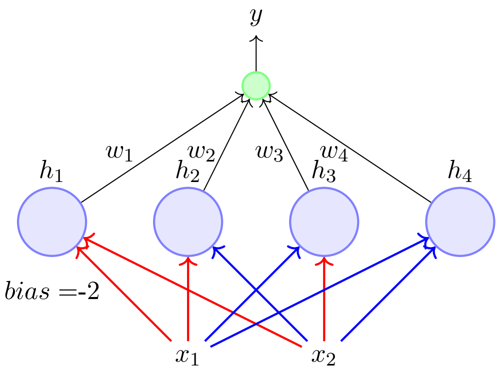
- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**



red edge indicates w = -1
blue edge indicates w = +1

Terminology:

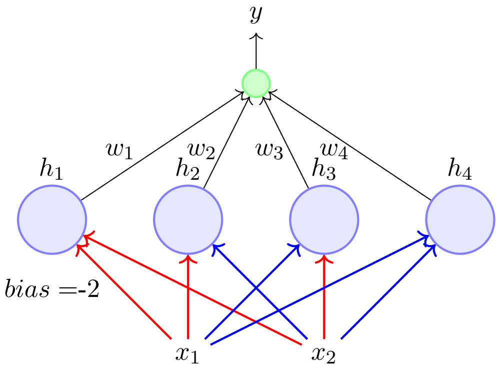
- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**



red edge indicates w = -1
blue edge indicates $w = +1$

Terminology:

- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4

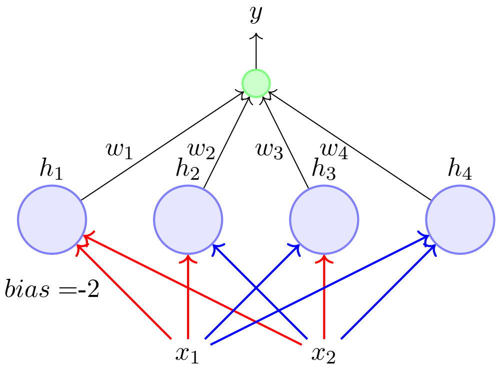


red edge indicates w = -1

blue edge indicates w = +1

Terminology:

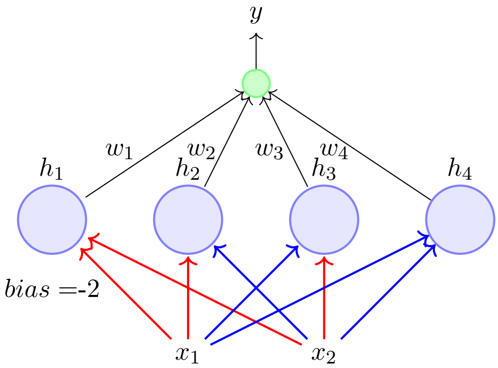
- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4
- The red and blue edges are called layer 1 weights



red edge indicates w = -1
blue edge indicates w = +1

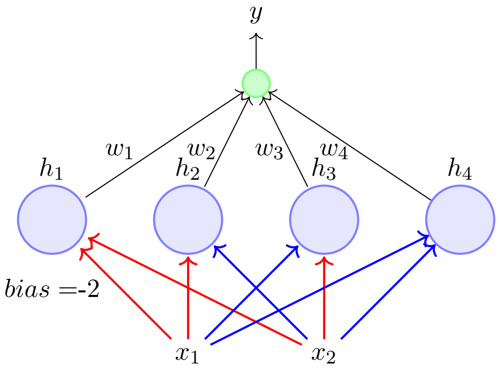
Terminology:

- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4
- The red and blue edges are called layer 1 weights
- w_1, w_2, w_3, w_4 are called layer 2 weights



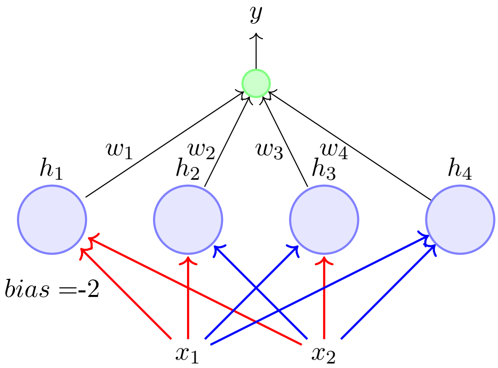
red edge indicates w = -1
blue edge indicates w = +1

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !



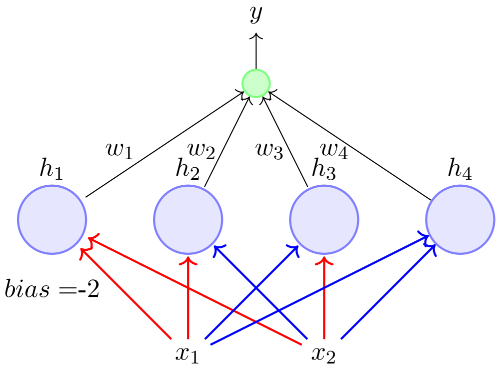
red edge indicates w = -1
blue edge indicates w = +1

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network



red edge indicates w = -1
blue edge indicates w = +1

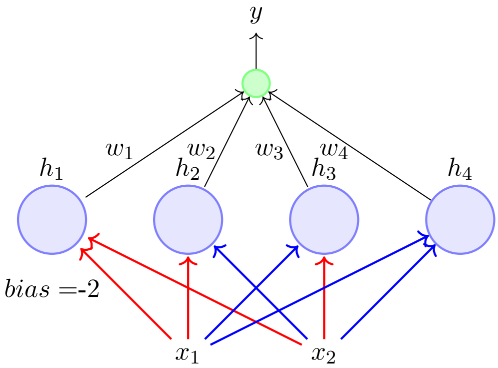
- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim!



red edge indicates w = -1

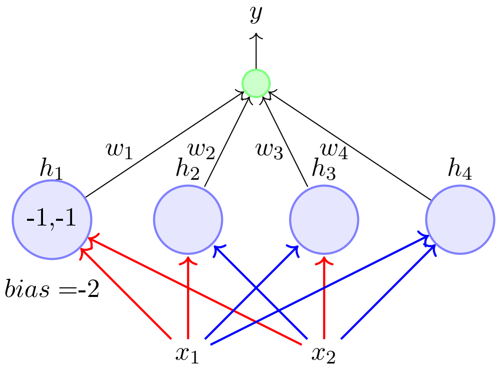
blue edge indicates $w = +1$

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on



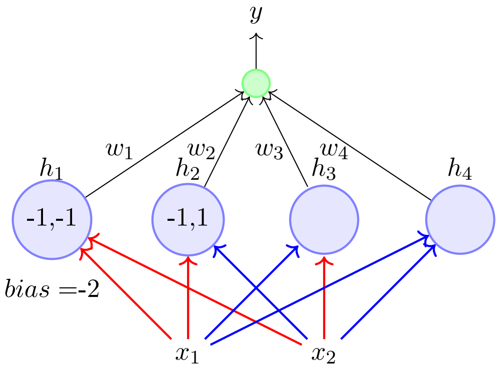
red edge indicates w = -1
blue edge indicates w = +1

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)



red edge indicates w = -1
blue edge indicates $w = +1$

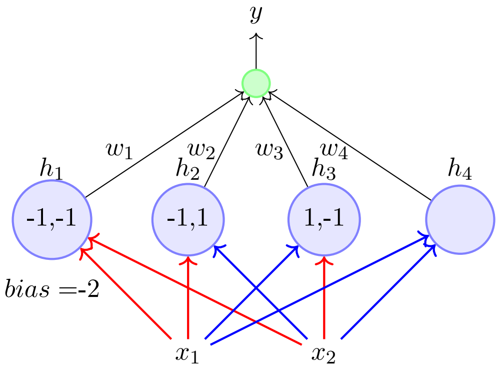
- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- the first perceptron fires for {-1,-1}



red edge indicates w = -1

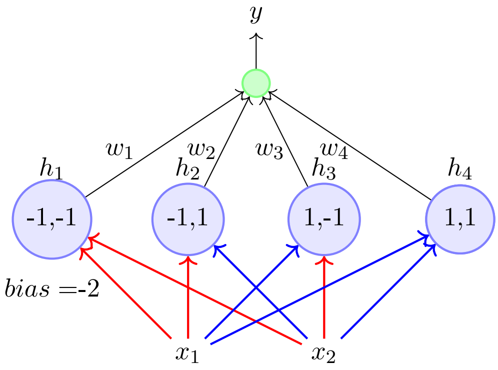
blue edge indicates w = +1

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- the second perceptron fires for {-1,1}



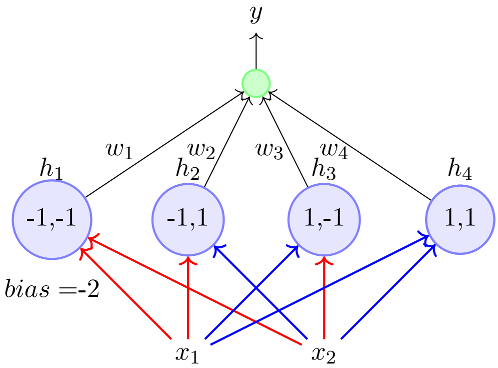
red edge indicates w = -1
blue edge indicates $w = +1$

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- the third perceptron fires for {1,-1}



red edge indicates w = -1
blue edge indicates w = +1

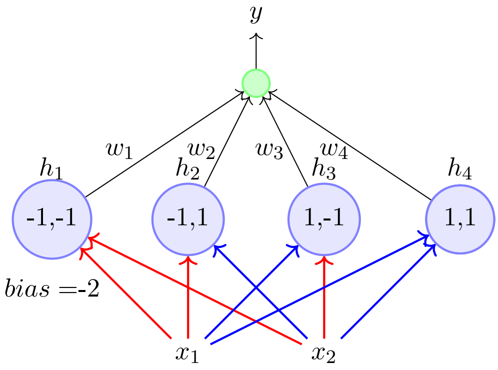
- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- the fourth perceptron fires for {1,1}



red edge indicates w = -1
blue edge indicates w = +1

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- Let us see why this network works by taking an example of the XOR function

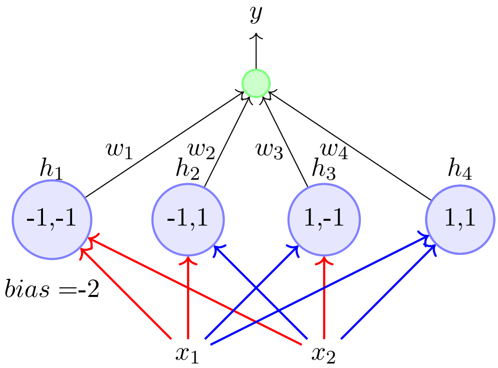
- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)



red edge indicates w = -1

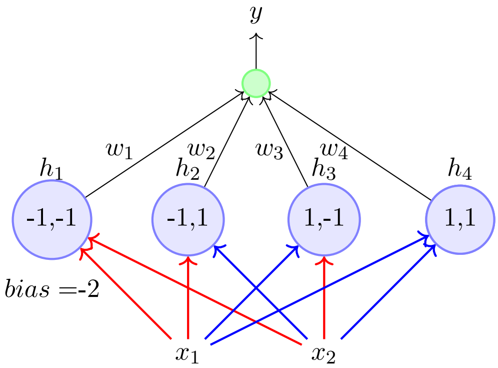
blue edge indicates w = +1

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)



red edge indicates w = -1
blue edge indicates w = +1

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1



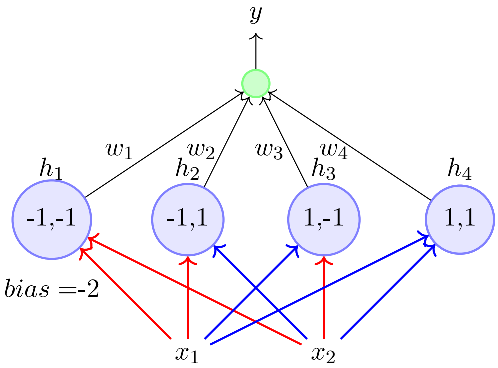
red edge indicates w = -1

blue edge indicates w = +1

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

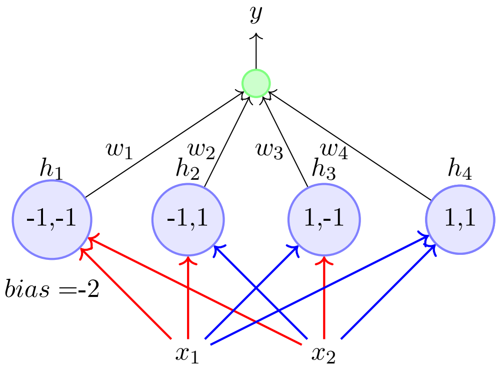


red edge indicates w = -1

blue edge indicates w = +1

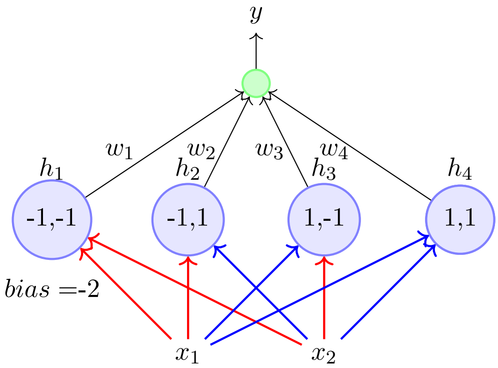
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)



x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

red edge indicates w = -1
blue edge indicates w = +1

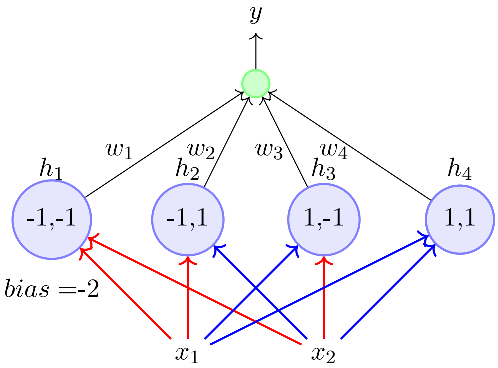


red edge indicates w = -1
blue edge indicates w = +1

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

- This results in the following four conditions to implement XOR: $w_1 < w_0, w_2 \geq w_0, w_3 \geq w_0, w_4 < w_0$

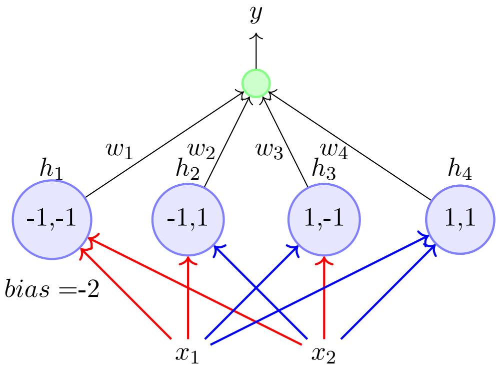


red edge indicates w = -1
blue edge indicates w = +1

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

- This results in the following four conditions to implement XOR: $w_1 < w_0, w_2 \geq w_0, w_3 \geq w_0, w_4 < w_0$
- Unlike before, there are no contradictions now and the system of inequalities can be satisfied

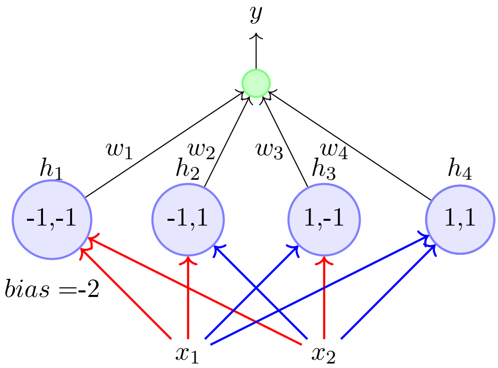


red edge indicates w = -1
blue edge indicates w = +1

- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

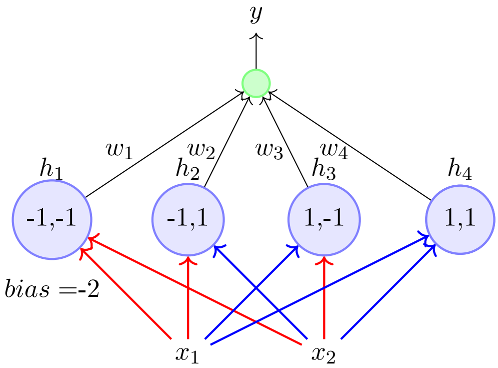
x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

- This results in the following four conditions to implement XOR: $w_1 < w_0, w_2 \geq w_0, w_3 \geq w_0, w_4 < w_0$
- Unlike before, there are no contradictions now and the system of inequalities can be satisfied
- Essentially each w_i is now responsible for one of the 4 possible inputs and can be adjusted to get the desired output for that input



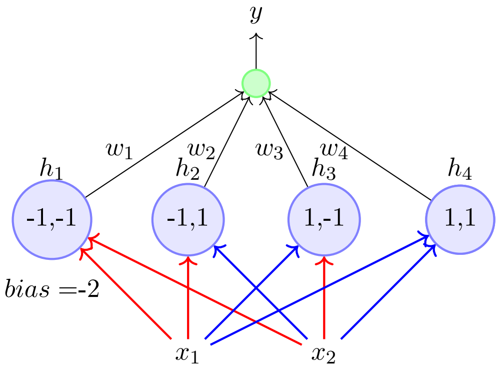
red edge indicates w = -1
blue edge indicates w = +1

- It should be clear that the same network can be used to represent the remaining 15 boolean functions also



red edge indicates w = -1
blue edge indicates $w = +1$

- It should be clear that the same network can be used to represent the remaining 15 boolean functions also
- Each boolean function will result in a different set of non-contradicting inequalities which can be satisfied by appropriately setting w_1, w_2, w_3, w_4

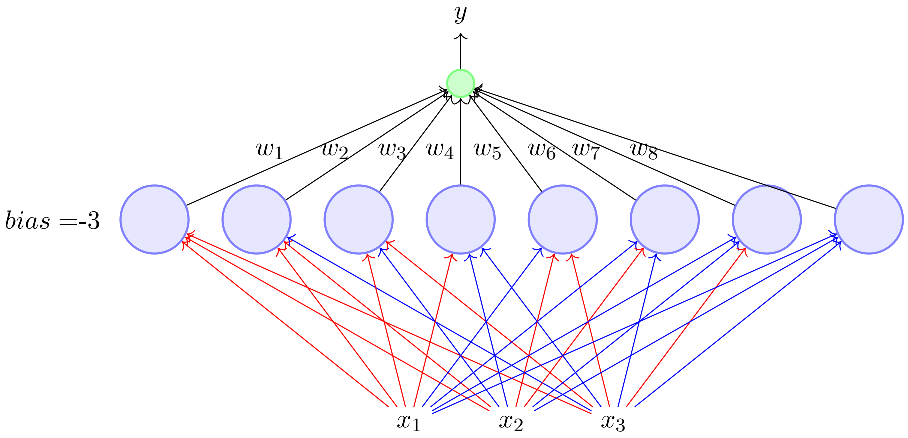


red edge indicates w = -1
blue edge indicates $w = +1$

- It should be clear that the same network can be used to represent the remaining 15 boolean functions also
- Each boolean function will result in a different set of non-contradicting inequalities which can be satisfied by appropriately setting w_1, w_2, w_3, w_4
- Try it!

- What if we have more than 3 inputs ?

- Again each of the 8 perceptorns will fire only for one of the 8 inputs
- Each of the 8 weights in the second layer is responsible for one of the 8 inputs and can be adjusted to produce the desired output for that input



- What if we have n inputs ?

Theorem

Any boolean function of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons and one output layer containing 1 perceptron

Theorem

Any boolean function of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons and one output layer containing 1 perceptron

Proof (informal:) We just saw how to construct such a network

Note: A network of $2^n + 1$ perceptrons is not necessary but sufficient. For example, we already saw how to represent AND function with just 1 perceptron

Theorem

Any boolean function of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons and one output layer containing 1 perceptron

Proof (informal:) We just saw how to construct such a network

Note: A network of $2^n + 1$ perceptrons is not necessary but sufficient. For example, we already saw how to represent AND function with just 1 perceptron

Catch: As n increases the number of perceptrons in the hidden layers obviously increases exponentially

- Again, why do we care about boolean functions ?

- Again, why do we care about boolean functions ?
- How does this help us with our original problem: which was to predict whether we like a movie or not?

- Again, why do we care about boolean functions ?
- How does this help us with our original problem: which was to predict whether we like a movie or not? Let us see!

- We are given this data about our past movie experience

$$
\begin{matrix} p_1 \\ p_2 \\ \vdots \\ n_1 \\ n_2 \\ \vdots \end{matrix}
\begin{bmatrix}
x_{11} & x_{12} & \dots & x_{1n} & y_1 = 1 \\
x_{21} & x_{22} & \dots & x_{2n} & y_2 = 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_{k1} & x_{k2} & \dots & x_{kn} & y_i = 0 \\
x_{j1} & x_{j2} & \dots & x_{jn} & y_j = 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
$$

- We are given this data about our past movie experience
- For each movie, we are given the values of the various factors $(x_1, x_2, \ldots, x_n)$ that we base our decision on and we are also also given the value of y (like/dislike)

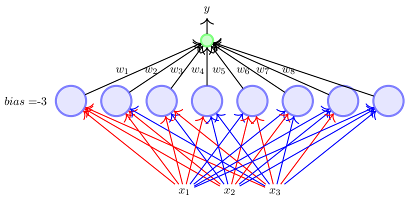
$$
\begin{array}{c}
p_1 \\ p_2 \\ \vdots \\ n_1 \\ n_2 \\ \vdots
\end{array}
\begin{bmatrix}
x_{11} & x_{12} & \ldots & x_{1n} & y_1 = 1 \\
x_{21} & x_{22} & \ldots & x_{2n} & y_2 = 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_{k1} & x_{k2} & \ldots & x_{kn} & y_i = 0 \\
x_{j1} & x_{j2} & \ldots & x_{jn} & y_j = 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
$$

$$
\begin{array}{c}
p_1 \\ p_2 \\ \vdots \\ n_1 \\ n_2 \\ \vdots
\end{array}
\begin{bmatrix}
x_{11} & x_{12} & \dots & x_{1n} & y_1 = 1 \\
x_{21} & x_{22} & \dots & x_{2n} & y_2 = 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_{k1} & x_{k2} & \dots & x_{kn} & y_i = 0 \\
x_{j1} & x_{j2} & \dots & x_{jn} & y_j = 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}
$$

- We are given this data about our past movie experience
- For each movie, we are given the values of the various factors $(x_1, x_2, \dots, x_n)$ that we base our decision on and we are also also given the value of y (like/dislike)
- p_i's are the points for which the output was 1 and n_i's are the points for which it was 0

$$\begin{array}{c} p_1 \\ p_2 \\ \vdots \\ n_1 \\ n_2 \\ \vdots \end{array} \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} & y_1 = 1 \\ x_{21} & x_{22} & \dots & x_{2n} & y_2 = 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{k1} & x_{k2} & \dots & x_{kn} & y_i = 0 \\ x_{j1} & x_{j2} & \dots & x_{jn} & y_j = 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

- We are given this data about our past movie experience
- For each movie, we are given the values of the various factors $(x_1, x_2, \dots, x_n)$ that we base our decision on and we are also also given the value of y (like/dislike)
- p_i's are the points for which the output was 1 and n_i's are the points for which it was 0
- The data may or may not be linearly separable



$$\begin{array}{c} p_1 \\ p_2 \\ \vdots \\ n_1 \\ n_2 \\ \vdots \end{array} \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} & y_1 = 1 \\ x_{21} & x_{22} & \dots & x_{2n} & y_2 = 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{k1} & x_{k2} & \dots & x_{kn} & y_i = 0 \\ x_{j1} & x_{j2} & \dots & x_{jn} & y_j = 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

- We are given this data about our past movie experience
- For each movie, we are given the values of the various factors $(x_1, x_2, \dots, x_n)$ that we base our decision on and we are also also given the value of y (like/dislike)
- p_i's are the points for which the output was 1 and n_i's are the points for which it was 0
- The data may or may not be linearly separable
- The proof that we just saw tells us that it is possible to have a network of perceptrons and learn the weights in this network such that for any given p_i or n_j the output of the network will be the same as y_i or y_j (i.e., we can separate the positive and the negative points)

The story so far ...

- Networks of the form that we just saw (containing, an input, output and one or more hidden layers) are called Multilayer Perceptrons (MLP, in short)

The story so far ...

- Networks of the form that we just saw (containing, an input, output and one or more hidden layers) are called Multilayer Perceptrons (MLP, in short)
- More appropriate terminology would be“Multilayered Network of Perceptrons” but MLP is the more commonly used name

The story so far ...

- Networks of the form that we just saw (containing, an input, output and one or more hidden layers) are called Multilayer Perceptrons (MLP, in short)
- More appropriate terminology would be“Multilayered Network of Perceptrons” but MLP is the more commonly used name
- The theorem that we just saw gives us the representation power of a MLP with a single hidden layer

The story so far ...

- Networks of the form that we just saw (containing, an input, output and one or more hidden layers) are called Multilayer Perceptrons (MLP, in short)
- More appropriate terminology would be "Multilayered Network of Perceptrons" but MLP is the more commonly used name
- The theorem that we just saw gives us the representation power of a MLP with a single hidden layer
- Specifically, it tells us that a MLP with a single hidden layer can represent **any** boolean function