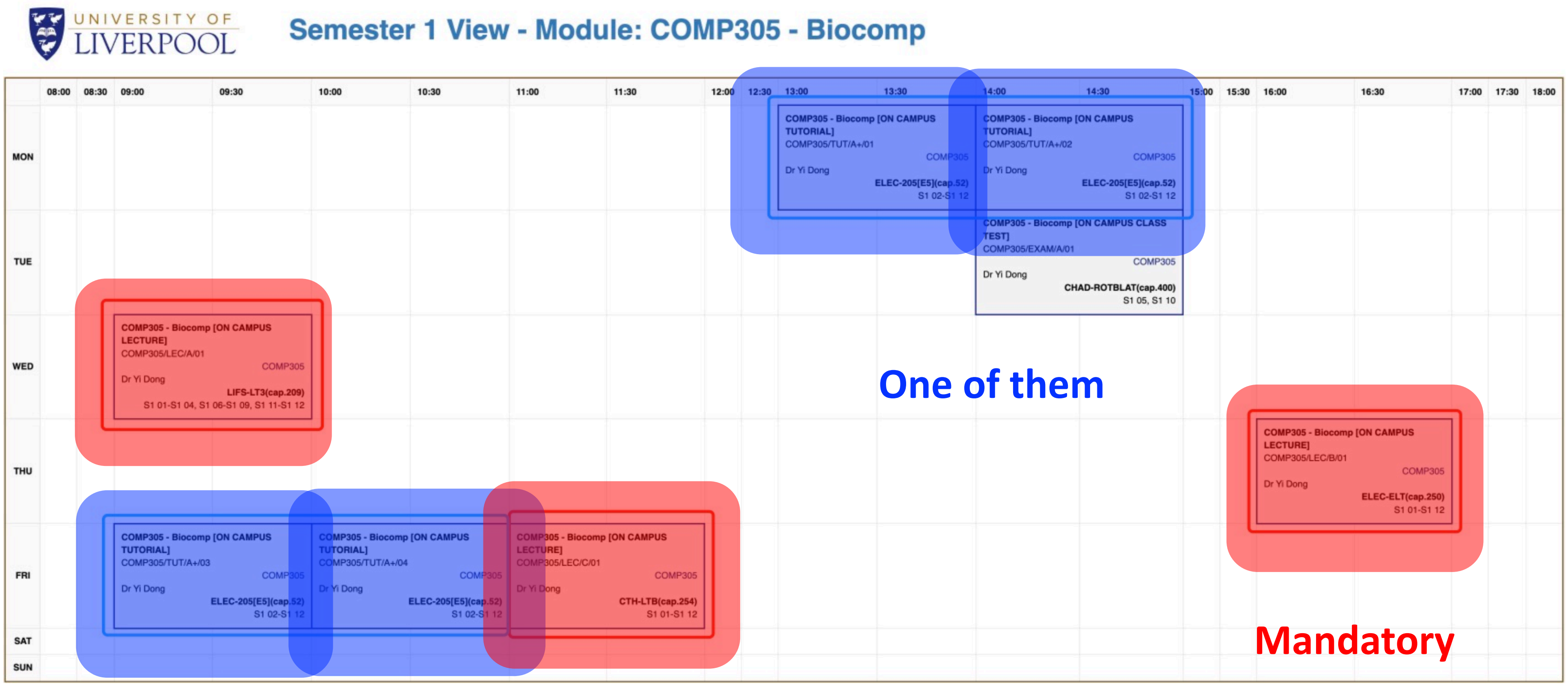
Comp305

Biocomputation

Lecturer: Yi Dong

Comp305 Module Timetable



There will be **26-30** lectures, thee per week. The lecture slides will appear on Canvas. Please use Canvas to access the lecture information. There will be **9** tutorials, one per week.

Lecture/Tutorial Rules

Questions are welcome as soon as they arise, because

1. Questions give feedback to the lecturer;

2. Questions help your understanding;

3. Your questions help your classmates, who might experience difficulties with formulating the same problems/doubts in the form of a question.

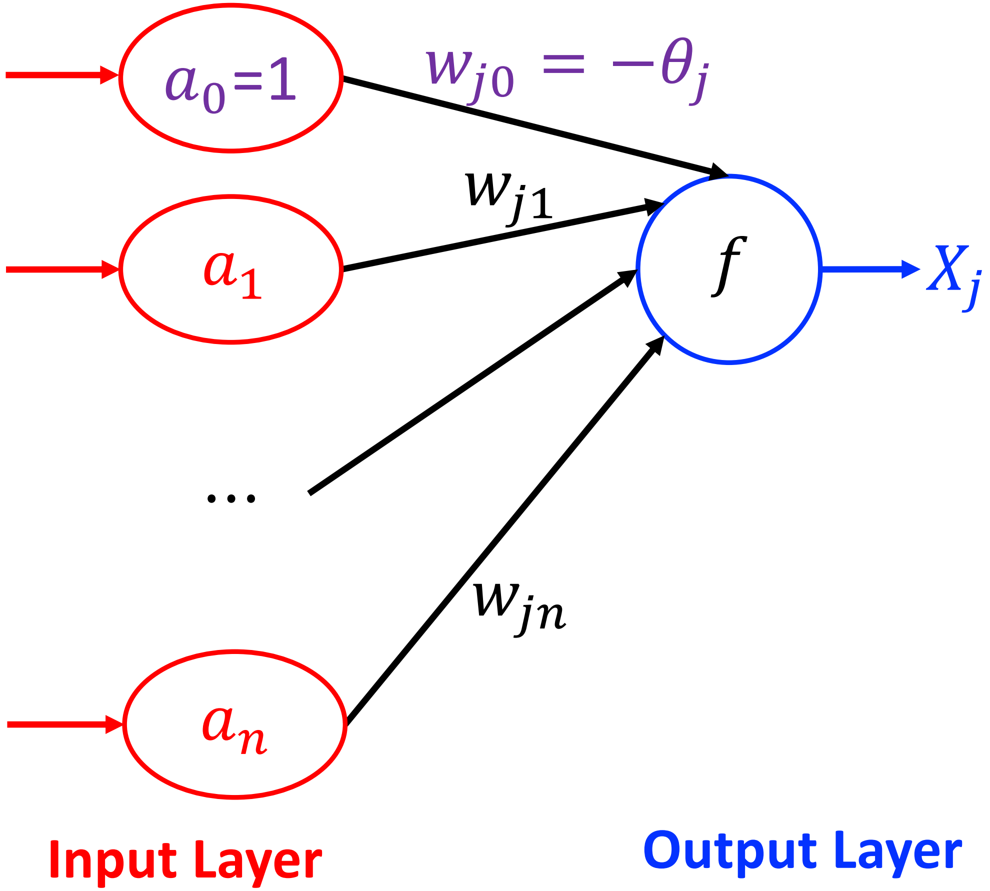
Comp305 Part I.

Artificial Neural Networks

Topic 5.

Multi-Layer Perceptron

Perceptron (1958): Semantics



The weighted input to the j-th output neuron is

$$S_j = \sum_{i=0}^{n} w_{ji} a_i \,,$$

The value X_j of j-th output neuron depends on whether the weighted input is greater than 0.

$$X_j = f(S_j) = \begin{cases} 1, & S_j \geq 0, \\ 0, & S_j < 0. \end{cases}$$

We call f as ***activation function***.

Perceptron Learning Algorithm

Algorithm 1: Perceptron Learning Algorithm

Data: Labelled data set D: r n-dimensional input points, each of which has m labels. Small positive real δ. Learning rate C.

Result: Weight matrix $w = [w_1, \cdots, w_m]$

Initialize weights w randomly;

while $!convergence\ (RMS \leq \delta)$ **do**

  Pick random $a' \in D$;

  $a \longleftarrow [1, a']$;

  for $j = 1, \cdots, m$ **do**

    /* We represent the learning rule in the vector form */

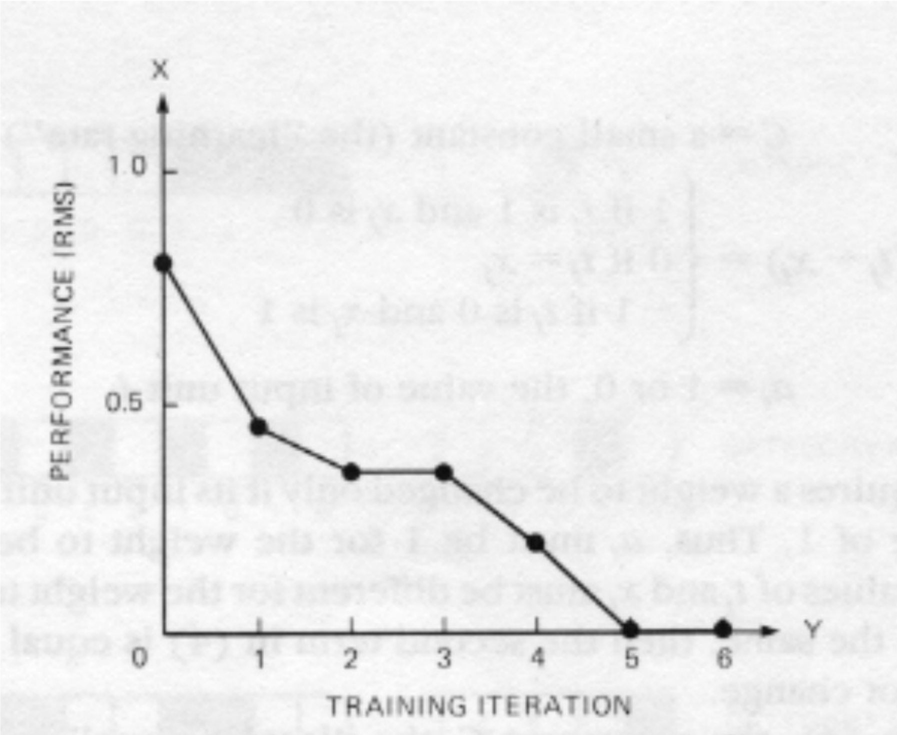
    $w_j = w_j + C(t_j - X_j)a$;

return w;

Then the convergence checking is only done after one epoch.

A common way is to enumerate all the patterns in D sequentially. An epoch means training the neural network with all the training data for one cycle.

Network Performance



Q: Does the learning rule always make network converge?

A: The learning rule will converge for the **absolutely linearly separable** data set.

Understand the Proof

- We care about $S = \sum_{i=0}^{n} w_i a_i = w \cdot a$. Specifically, we care the **sign** of $w \cdot a$!
- The direction of w matters, while the length, i.e., $||w||$ does not. It is because
$$w \cdot a > 0 \Leftrightarrow \lambda w \cdot a > 0, \lambda > 0$$
- Consider the learning rule as the ways to change the direction of w under different situations.
- Then the convergence of perceptron learning rule can be considered as that w finally has the similar direction with the optimal w^* that exists but is unknown.
- During the proof, what we do is to show the angle between w and w^* gets smaller (not necessarily monotonically) along with the number of misclassification and cannot be smaller than 0.

Beyond Linear Separability

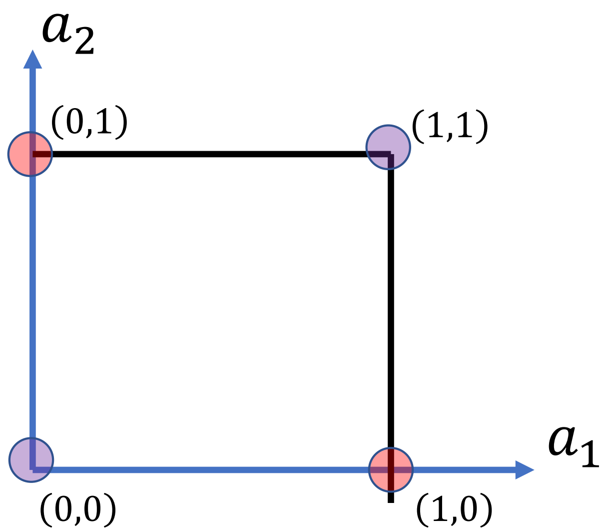
- Now we know the perceptron learning algorithm can finally converge for the data set that is linearly separable.

- How about the one that is not linearly separable?

- The best-known example is an “XOR” (exclusive “OR”) gate, called **the *XOR problem***.

The XOR problem

$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	**"XOR"**
1	1	0
0	1	1
1	0	1
0	0	0

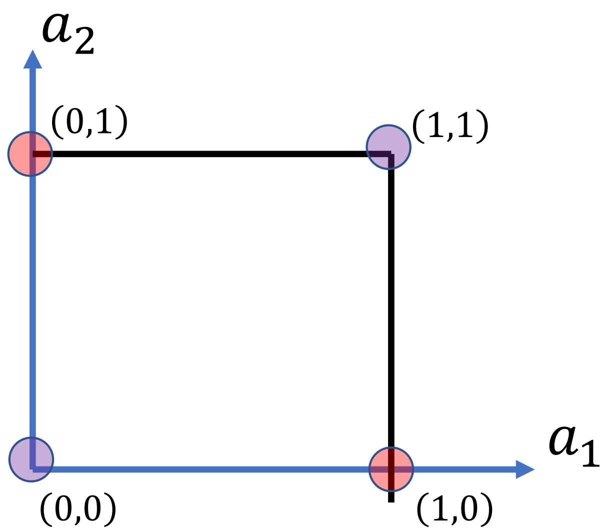


"XOR" – the output is true if only one input, a_1 **or** a_2 is true.

Apparently, XOR is **NOT** linearly separable. (**Why?**)

The XOR problem

a_1	a_2	"XOR"
1	1	0
0	1	1
1	0	1
0	0	0



Apparently, XOR is **NOT** linearly separable.

Proof. Assume there is a line $w_0 + w_1 a_1 + w_2 a_2 = 0$, such that

$$\begin{cases} w_0 + w_1 \times 0 + w_2 \times 1 > 0 \\ w_0 + w_1 \times 1 + w_2 \times 0 > 0 \\ w_0 + w_1 \times 0 + w_2 \times 0 < 0 \\ w_0 + w_1 \times 1 + w_2 \times 1 < 0 \end{cases}$$

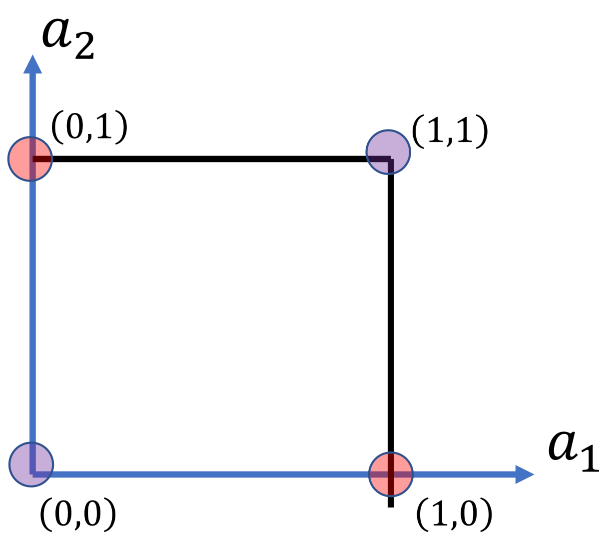
Two red points

Two purple points

$$\Longrightarrow \begin{cases} \left.\begin{array}{r} w_2 > -w_0 \\ w_1 > -w_0 \end{array}\right\} w_1 + w_2 > -2w_0 \\ w_0 < 0 \\ w_1 + w_2 < -w_0 \end{cases}$$

The XOR problem

a_1	a_2	"XOR"
1	1	0
0	1	1
1	0	1
0	0	0



Apparently, XOR is **NOT** linearly separable.

Proof. Assume there is a line $w_0 + w_1 a_1 + w_2 a_2 = 0$, such that

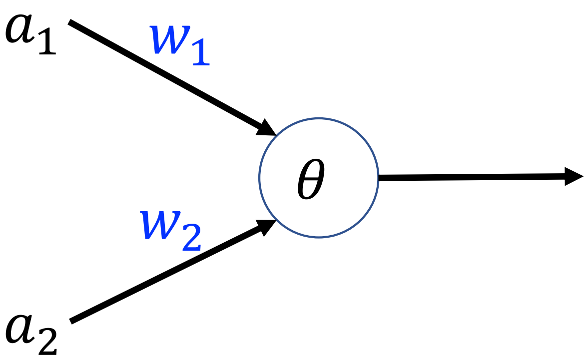
$$\begin{cases} w_0 + w_1 \times 0 + w_2 \times 1 > 0 \\ w_0 + w_1 \times 1 + w_2 \times 0 > 0 \\ w_0 + w_1 \times 0 + w_2 \times 0 < 0 \\ w_0 + w_1 \times 1 + w_2 \times 1 < 0 \end{cases}$$

Two red points

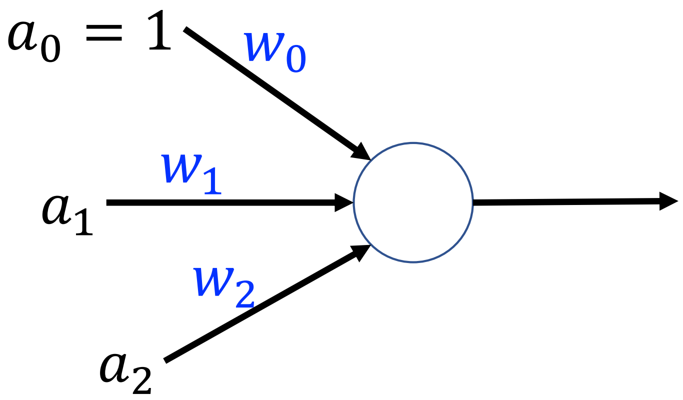
Two purple points

$$\Longrightarrow \begin{cases} \left.\begin{aligned} w_2 > -w_0 \\ w_1 > -w_0 \end{aligned}\right\} w_1 + w_2 > -2w_0 \\ w_0 < 0 \\ w_1 + w_2 < -w_0 \end{cases}$$

The XOR problem



MP neuron

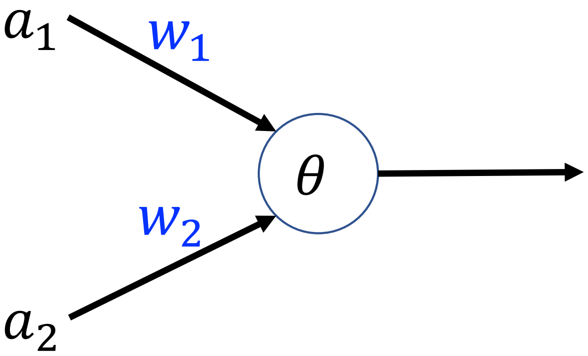


Perceptron

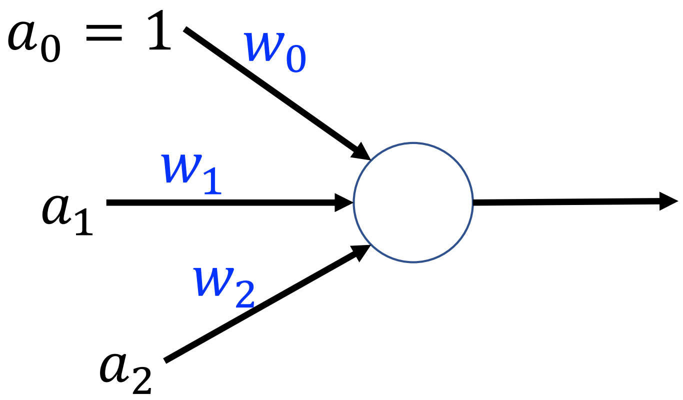
Can we describe "XOR" by a single MP neuron? **No**.

Can we learn "XOR" by a perceptron?

The XOR problem



MP neuron



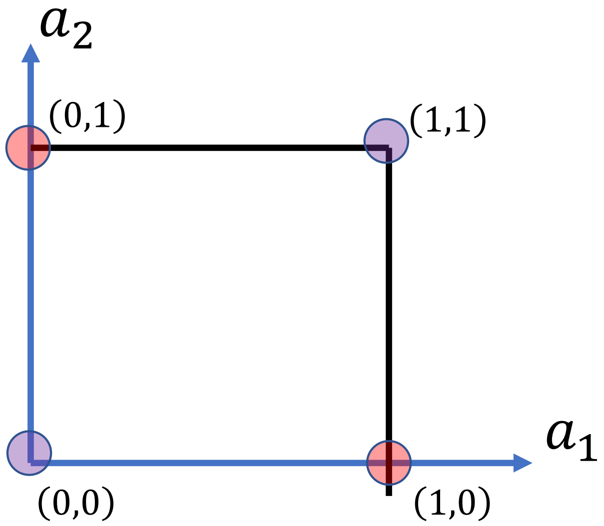
Perceptron

Can we describe "XOR" by a single MP neuron? **No.**

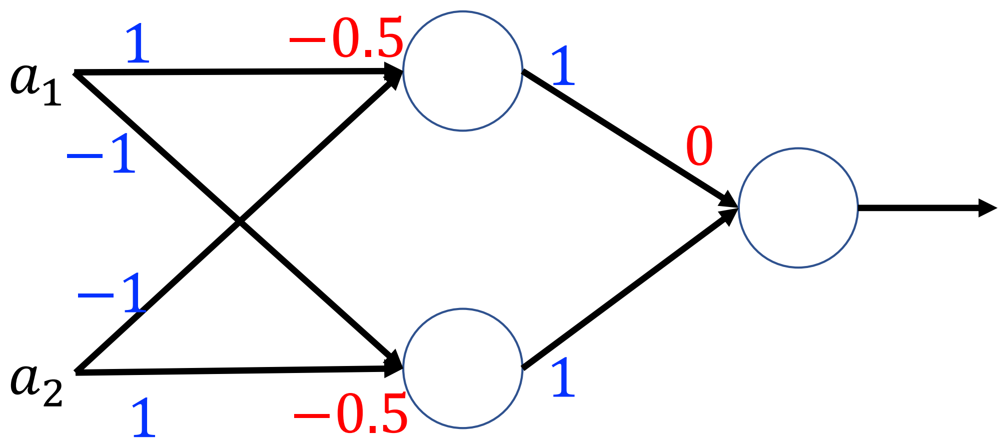
Can we learn "XOR" by a perceptron? **No.**

Hidden Neurons

a_1	a_2	"XOR"
1	1	0
0	1	1
1	0	1
0	0	0

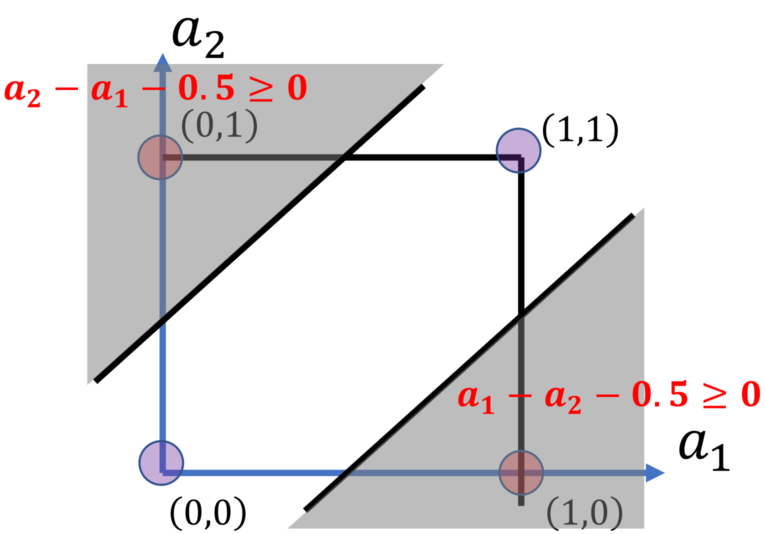


Minsky and Papert showed that **in the case of any non-linearly separable problem, such as XOR, in the network architecture *there must be "hidden neurons"*,** *i.e*. the neurons with output not available to the outside world, in order to help turn the problem into a linearly separable one for the outputs.

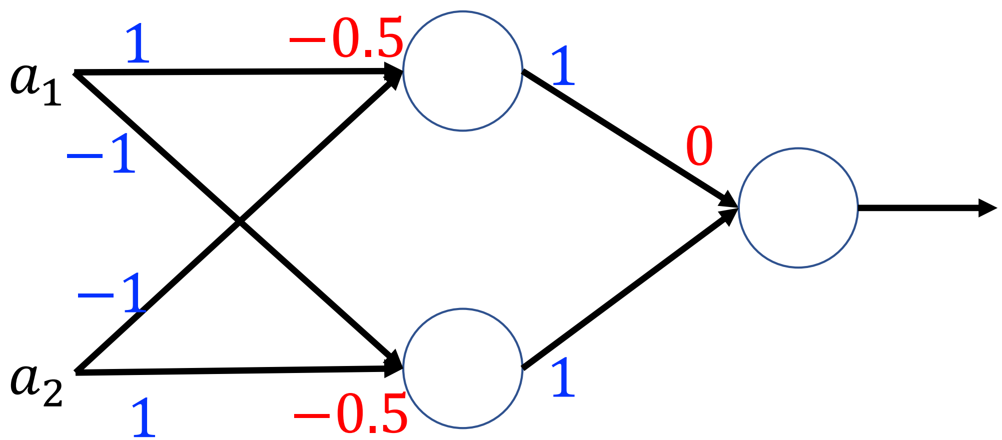


Hidden Neurons

a_1	a_2	"XOR"
1	1	0
0	1	1
1	0	1
0	0	0

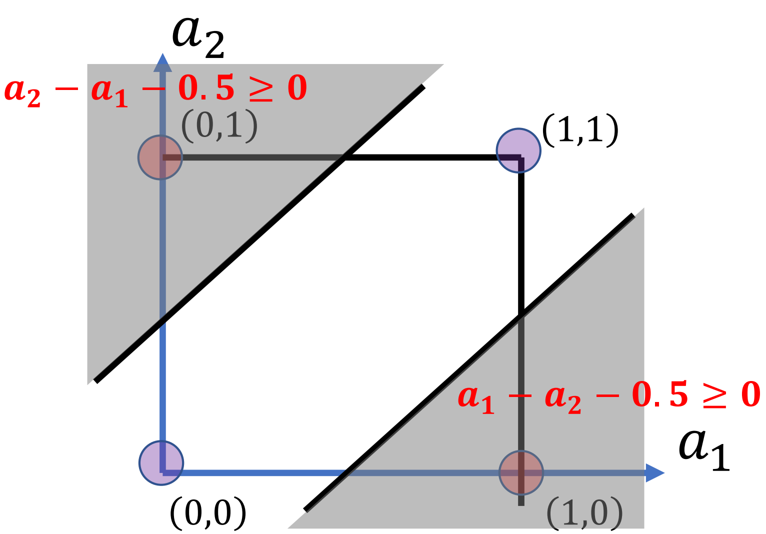


- The three-layer perceptron below is capable to represent XOR.
- Each hidden neuron separates the input space into a closed positive and open negative half-space.
- The left bottom figure shows the linear separations defined by each hidden unit, while the positive half-spaces are shaded.

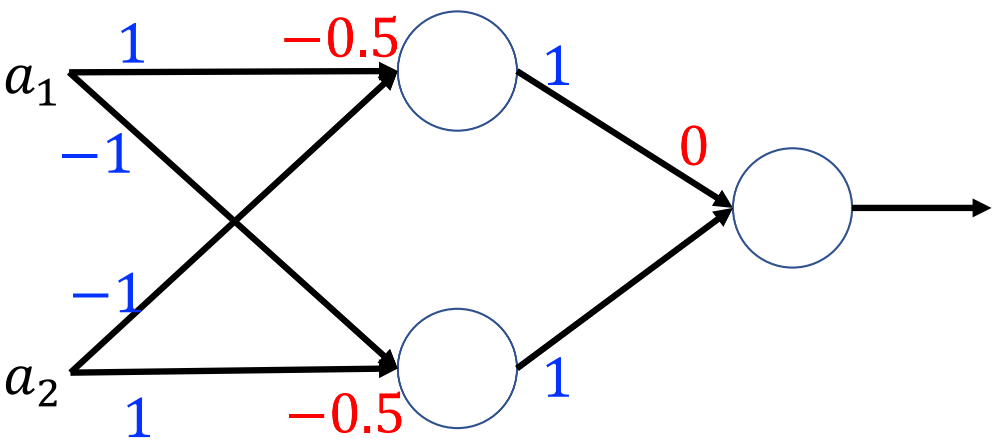


Hidden Neurons

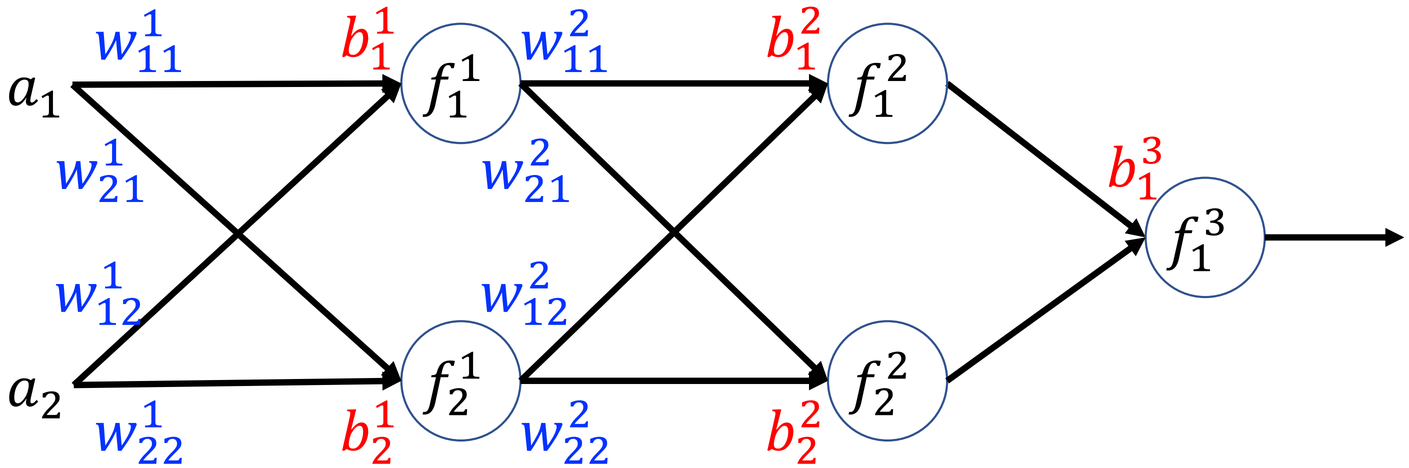
a_1	a_2	"XOR"
1	1	0
0	1	1
1	0	1
0	0	0



This example introduces the idea of ***Multilayer Perceptron.***



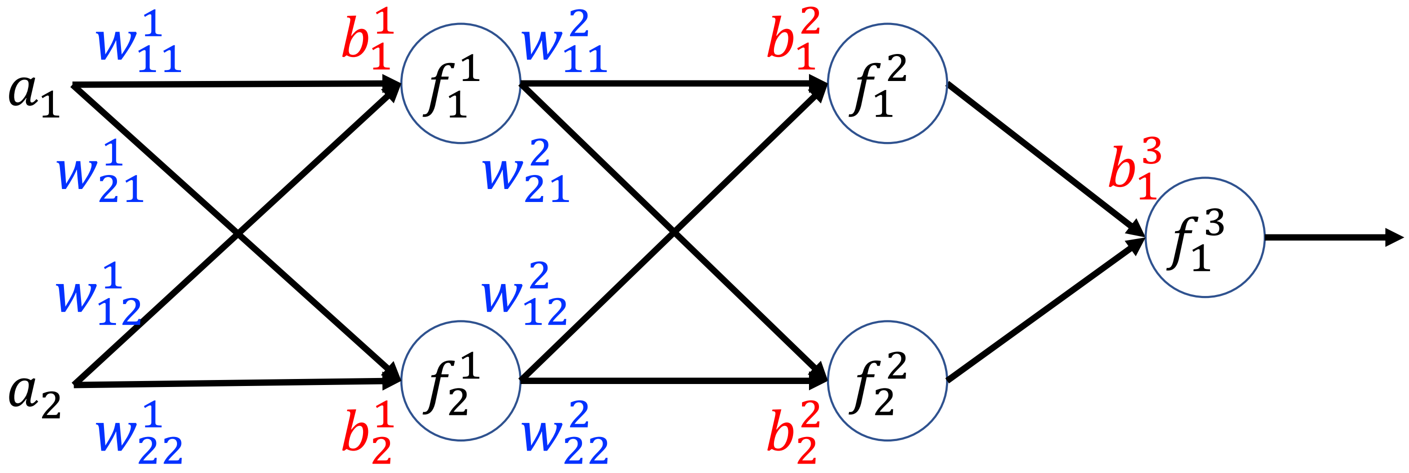
Multilayer Perceptron



A multilayer perceptron (**MLP**) is a layered architecture of neurons, where

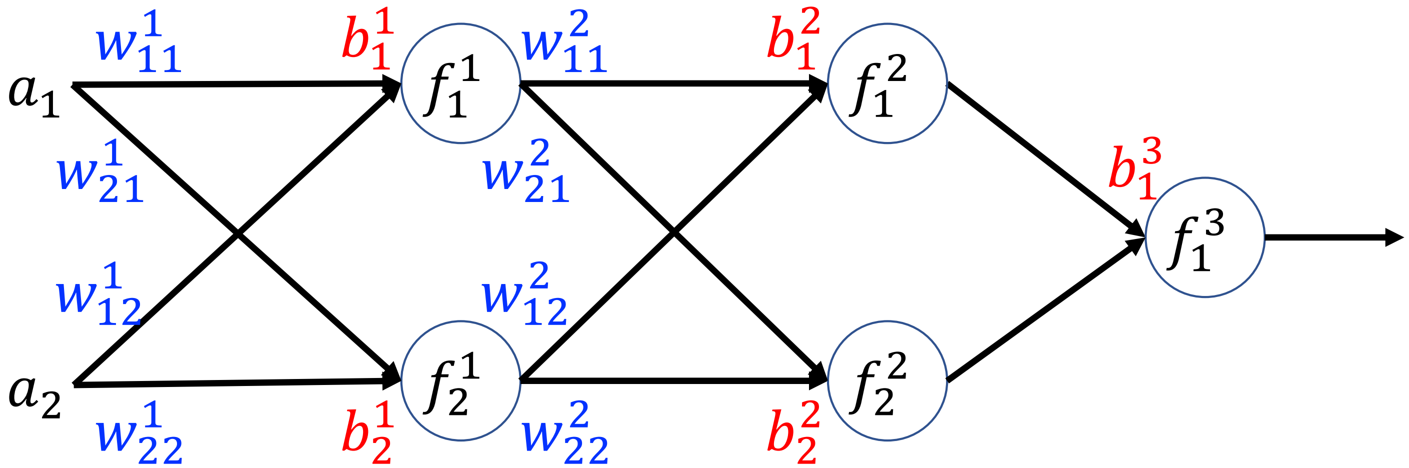
- all the neurons are divided into l subsets, each set is called a layer;
- There are only connections between two adjacent layers. Usually, ***the neurons within a layer are not connected to each other***, though some neural models make use of this kind of architecture.

Multilayer Perceptron



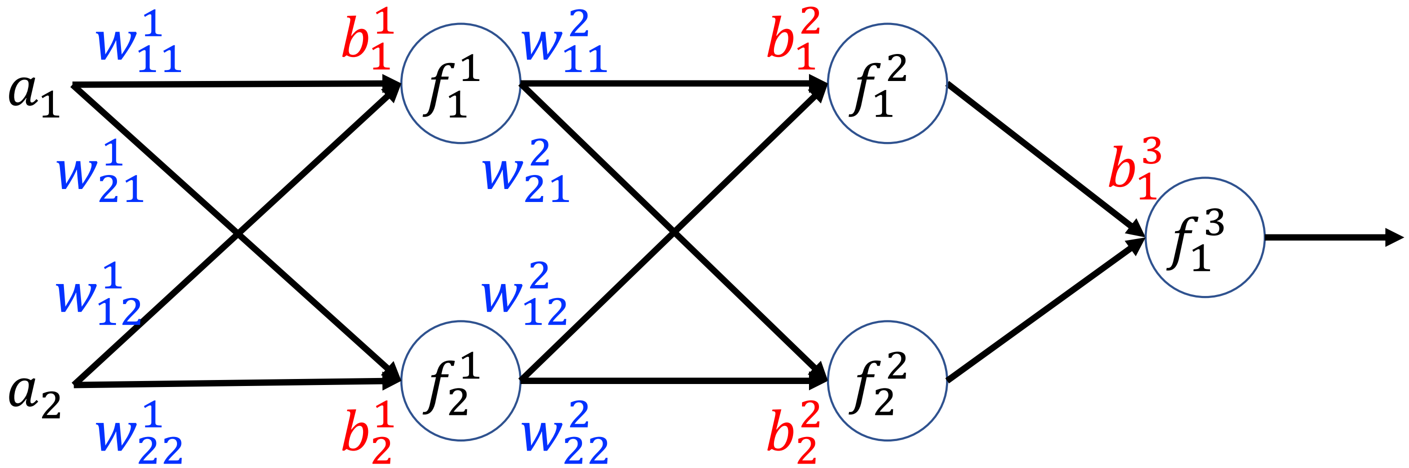
- The first layer is ***the input layer*** ***(We don't usually count it)***;
- The last layer is ***the output layer***.
- All other layers with no direct connections from or to the outside are called ***hidden layers***.

Multilayer Perceptron



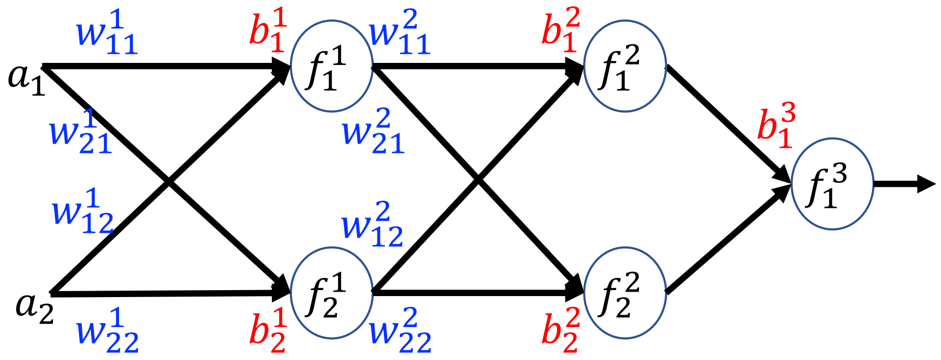
- We consider the fully-connected architectures in this module, that is, every neuron from one layer is connected to all neurons in the following layer.
- Each connection is associated with a real weight and a real bias.
- Inputs are real. Outputs are real.

Multilayer Perceptron



- The input is processed and propagated from one layer to the next, until the final result is computed.
- This process represents the ***forward propagation***.
- For simplicity, from now we assume **the biases in the network are all zero**.

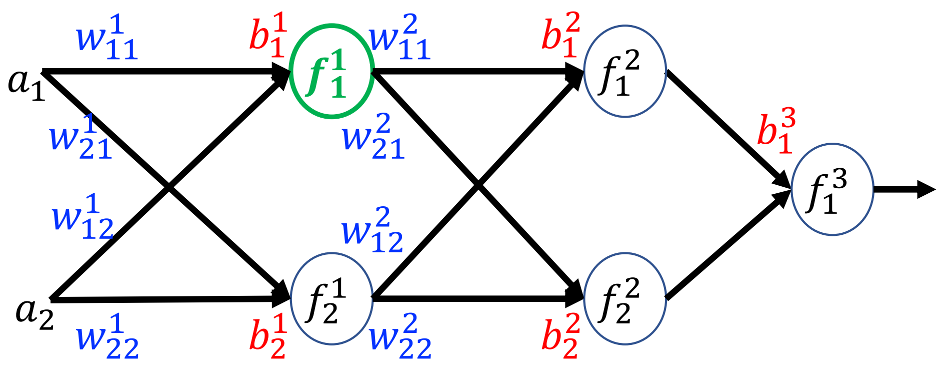
Forward Propagation



l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h: \mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

- **The difference of the multilayer perceptron**, compared to the single layer one, is that **the output value X^1 of the first layer is not the output value of the multilayer perceptron any more.**
- **The output value X^1 of the first layer** is the *input* to the next layer.

Forward Propagation



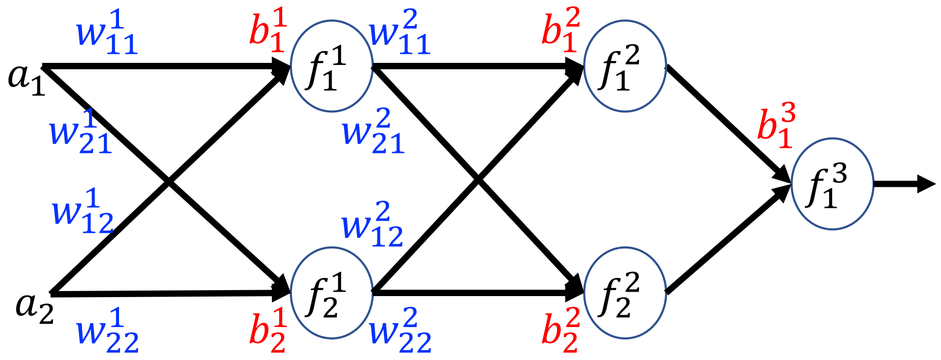
l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h: \mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

First of all, we introduce the computation for a single neuron. For instance, consider the first neuron in the first hidden layer.

$$S_1^1 = \sum_{i=1}^{n^0} w_{1i}^1 X_i^0 + \underbrace{\boldsymbol{b_1^1}}_{=0} = \sum_{i=1}^{n^0} w_{1i}^1 X_i^0$$

$$X_1^1 = f_1^1(S_1^1) = f_1^1\left(\sum_{i=1}^{n^0} w_{1i}^1 X_i^0\right)$$

Forward Propagation

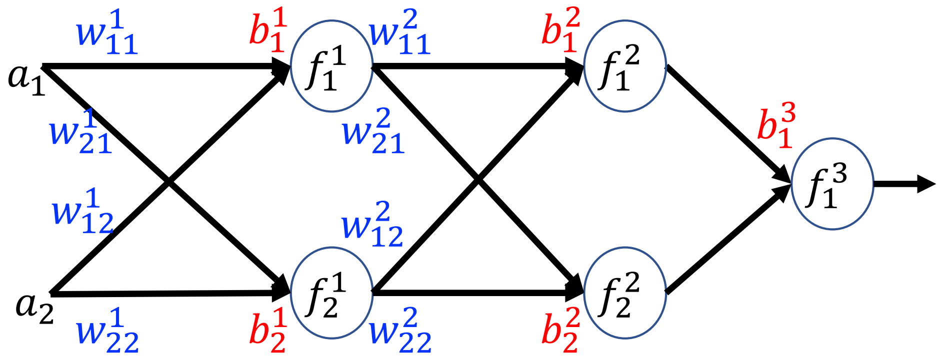


l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h: \mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

We start from the first hidden layer. The output of the j-th neuron in the first layer is

$$X_j^1 = f_j^1(S_j^1) = f_j^1\left(\sum_{i=1}^{n^0} w_{ji}^1 X_i^0\right) \triangleq F_j^1(w_j^1, X^0), \qquad j = 1, \cdots, n^1$$

Forward Propagation



l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h:\mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

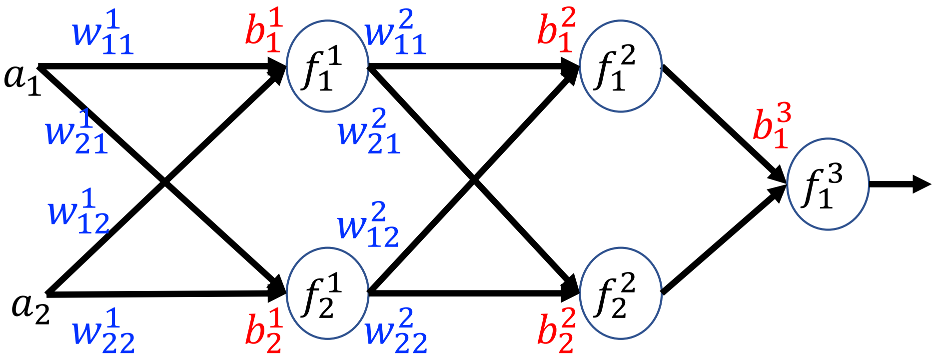
We start from the first hidden layer. The output of the j-th neuron in the first layer is

$$X_j^1 = f_j^1(S_j^1) = f_j^1\left(\sum_{i=0}^{n^0} w_{ji}^1 X_i^0\right) \triangleq F_j^1(w_j^1, X_i^0), \qquad j = 1, \cdots, n^1$$

We can then describe the above relation in a vector form:

$$X^1 = F^1(w^1, X^0)$$

Forward Propagation



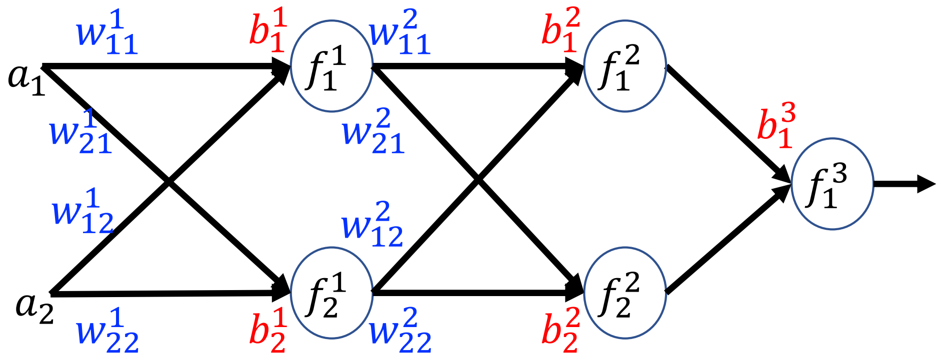
l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
f^h:$\mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

Similarly, we can derive the relation for the following layers:

$$X^1 = F^1(w^1, X^0)$$

What we got in the last slide.

Forward Propagation



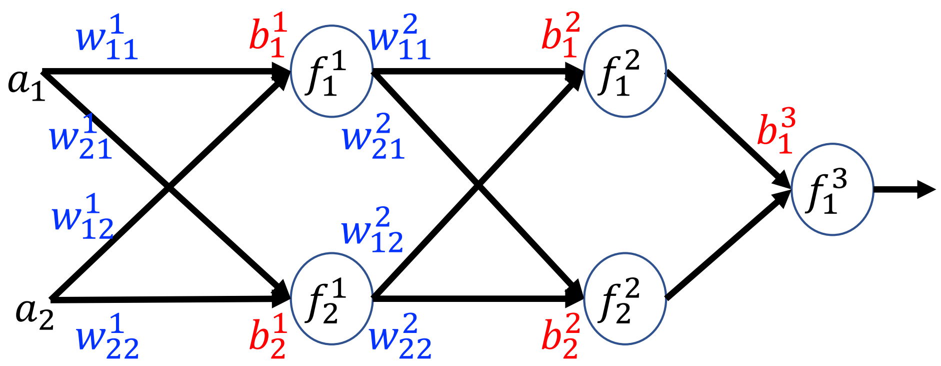
l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h : \mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

Similarly, we can derive the relation for the following layers:

$$X^1 = F^1(w^1, X^0)$$
$$X^2 = F^2(w^2, X^1)$$
$$X^3 = F^3(w^3, X^2)$$
$$\cdots$$
$$X^l = F^l(w^l, X^{l-1})$$

What we got in the last slide.

Forward Propagation



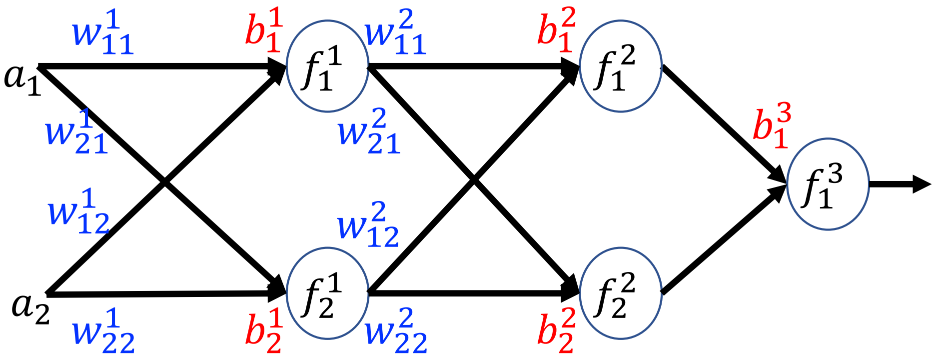
l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h: \mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

Similarly, we can derive the relation for the following layers:

What we got in the last slide.

$$X^1 = F^1(w^1, X^0)$$
$$X^2 = F^2(w^2, X^1)$$
$$X^3 = F^3(w^3, X^2)$$
$$\cdots$$
$$X^l = F^l\left(w^l, X^{l-1}\right)$$

Forward Propagation



l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
f^h:$\mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

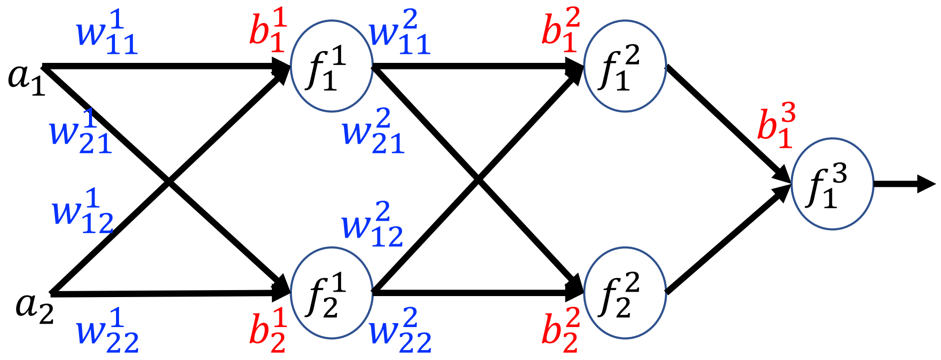
Finally, we get:

$$X^l = F^l\left(w^l, F^{l-1}\left(w^{l-1}, \cdots F^1(w^1, X^0)\right)\right)$$

We may represent it in another form:

$$X = X^l = F\left(w^l, w^{l-1}, \cdots, w^1, X^0\right) = F\left(w^l, w^{l-1}, \cdots, w^1, a\right)$$

Forward Propagation



l: the number of layers,
n^h: the number of neurons in the h-th layer
$n = n^0$: the number of input neurons (0-th layer).
$m = n^l$: the number of output neurons (l-th layer).
X^h: the output value of the h-th layer.
$a = X^0$: the input value of the MLP.
$X = X^l$: the output value of the MLP.
$f^h:\mathbb{R}^{n_h} \to \mathbb{R}^{n_h}$: activation function of the h-th layer

Finally, we get:

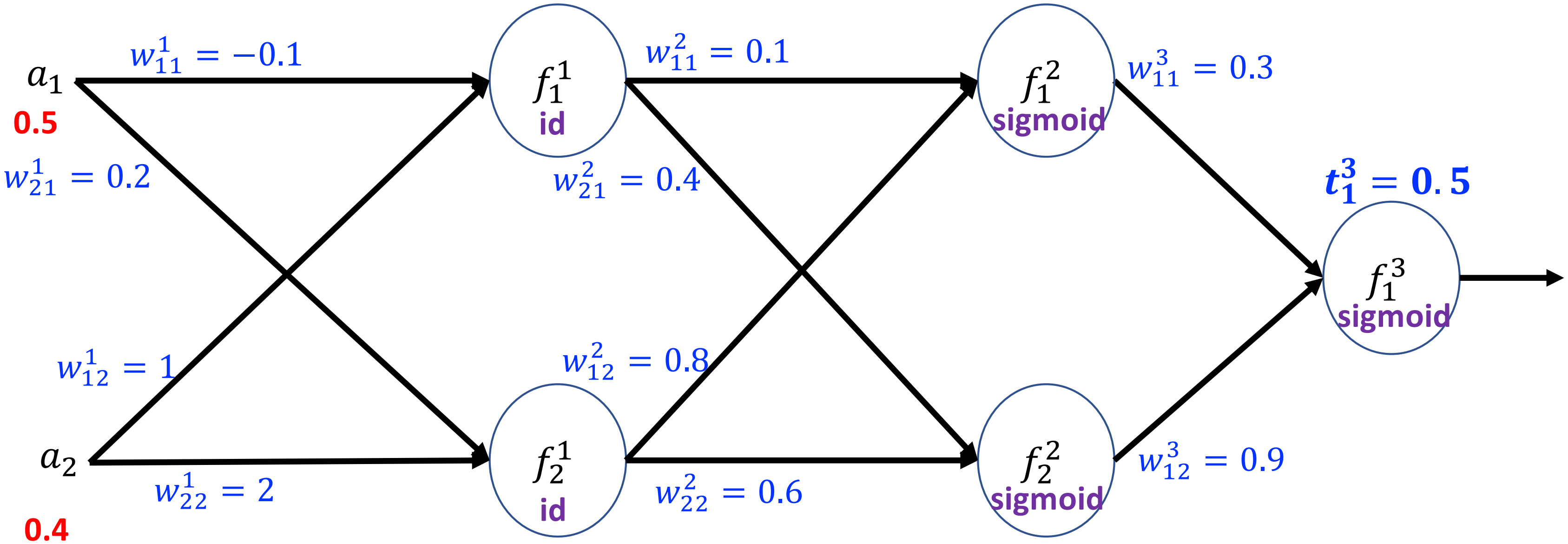
$$X^l = F^l\left(w^l, F^{l-1}\left(w^{l-1}, \cdots F^1(w^1, X^0)\right)\right)$$

We may represent it in another form:

$$X = X^l = F\left(w^l, w^{l-1}, \cdots, w^1, X^0\right) = F\left(w^l, w^{l-1}, \cdots, w^1, a\right)$$

The process of such layer-by-layer calculation to obtain the output of a multilayer perceptron is thus called forward propagation.

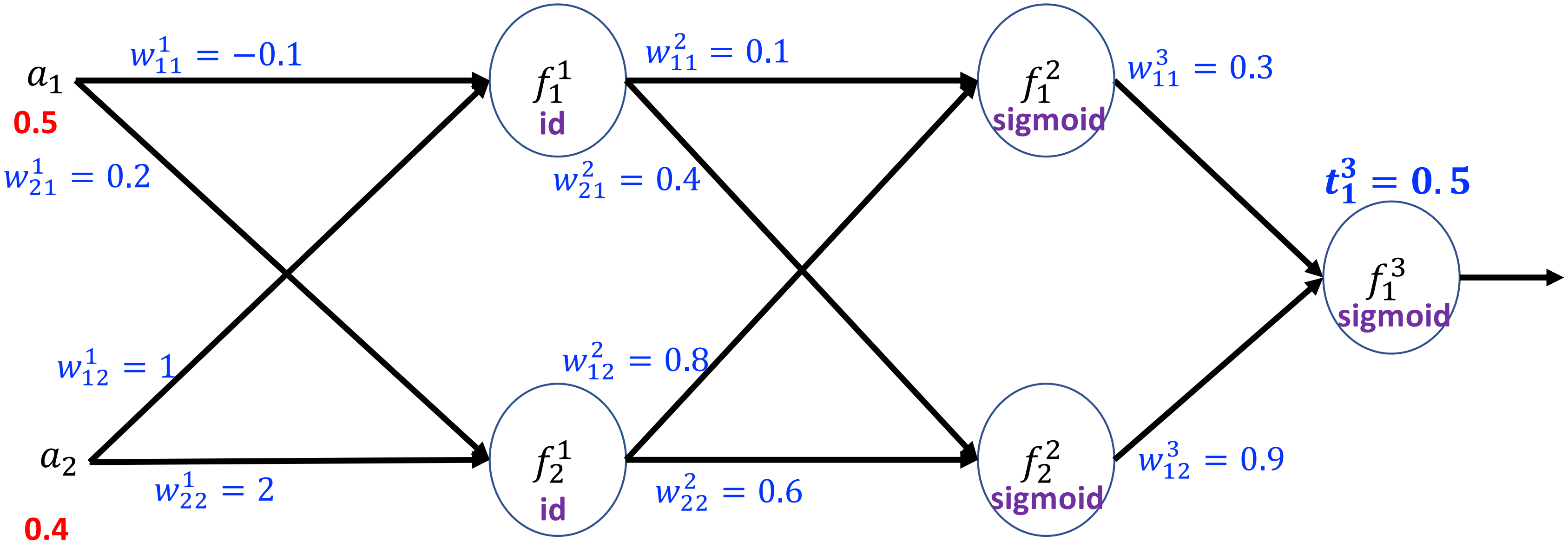
A Running Example



a_1	a_2	t_1
0.5	0.4	0.5

As we mentioned, we assume bias are all equal to 0 for simplicity.

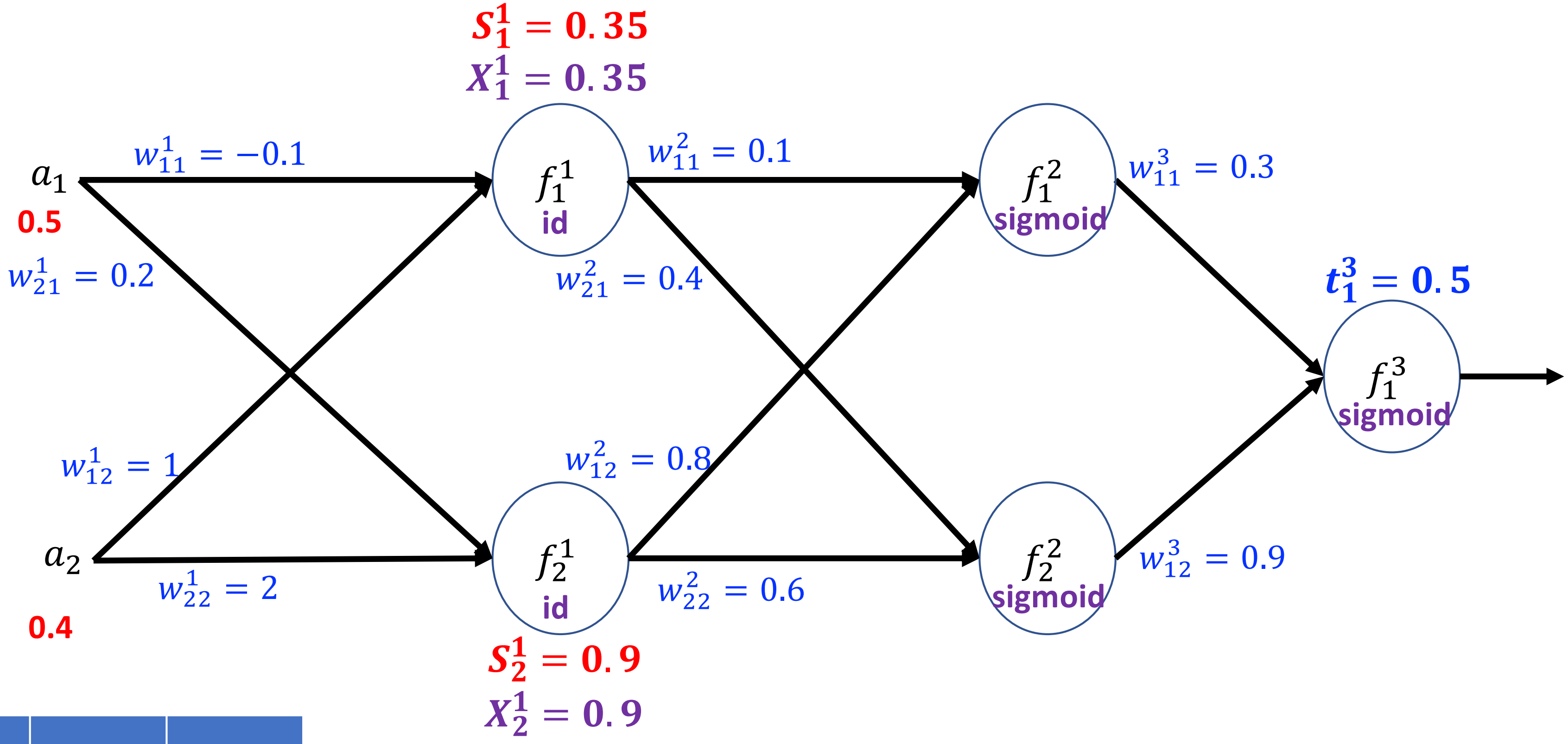
A Running Example



$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	$\boldsymbol{t_1}$
0.5	0.4	0.5

Id: identity function.
You may not know sigmoid function. I will introduce it later.

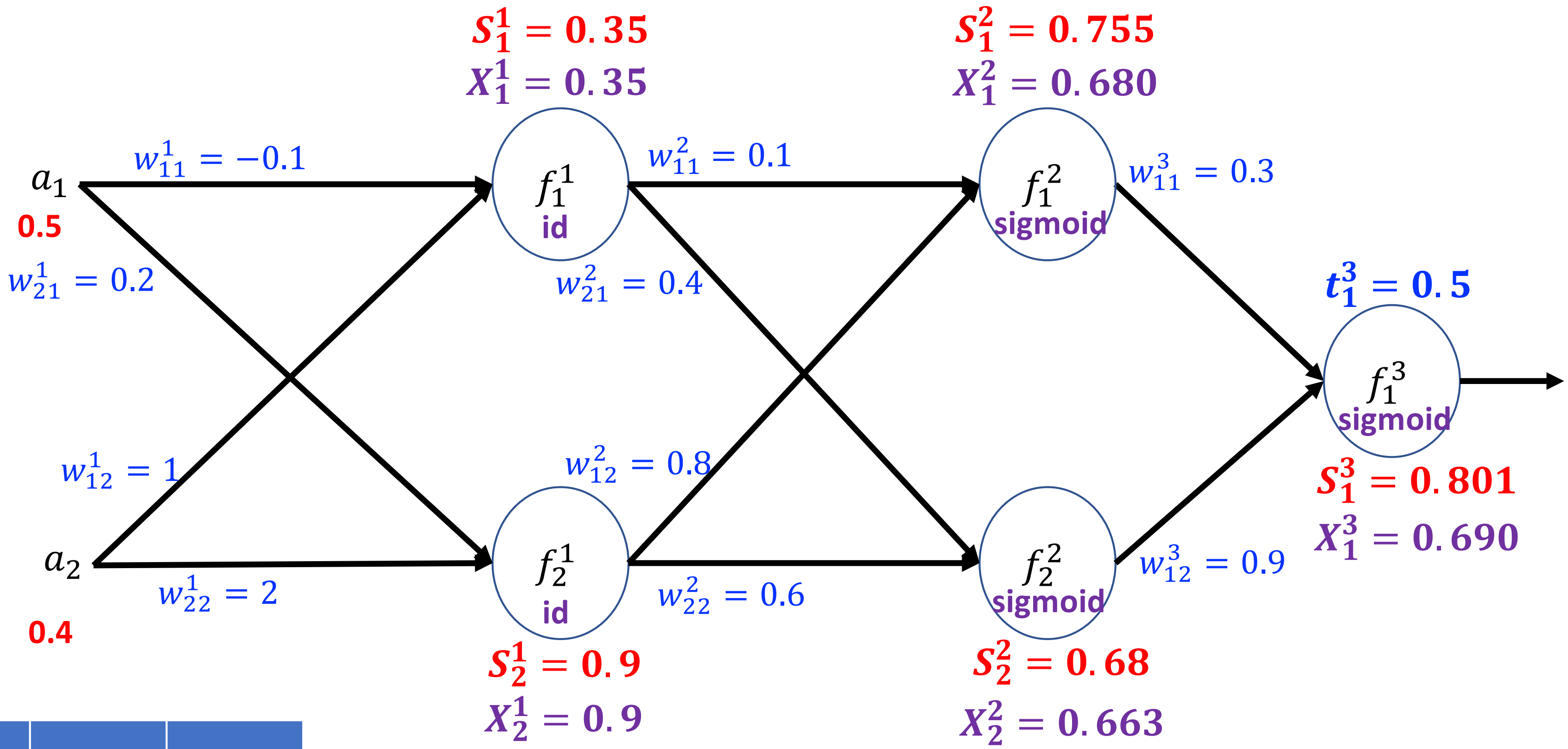
A Running Example



a_1	a_2	t_1
0.5	0.4	0.5

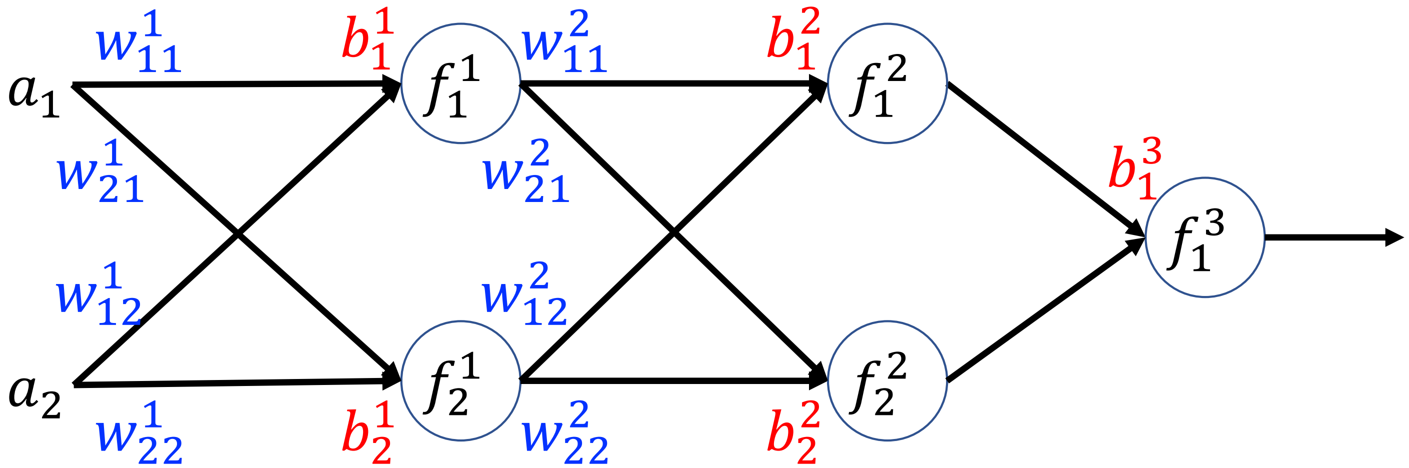
$$X_1^1 = f_1^1(S_1^1) = \mathrm{id}(w_{11}^1 X_1^0 + w_{12}^1 X_2^0) = \mathrm{id}(w_{11}^1 a_1 + w_{12}^1 a_2)$$
$$= \mathrm{id}(-0.1\times0.5 + 1\times0.4) = \mathrm{id}(0.35) = 0.35$$

A Running Example



a_1	a_2	t_1
0.5	0.4	0.5

Multilayer Perceptron



- Issue: The hidden neurons cannot be trained by making their outputs become closer to the desired values given by the training set.