

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are both tilted at an angle.

Computing the backpropagation in Feed-Forward neural networks

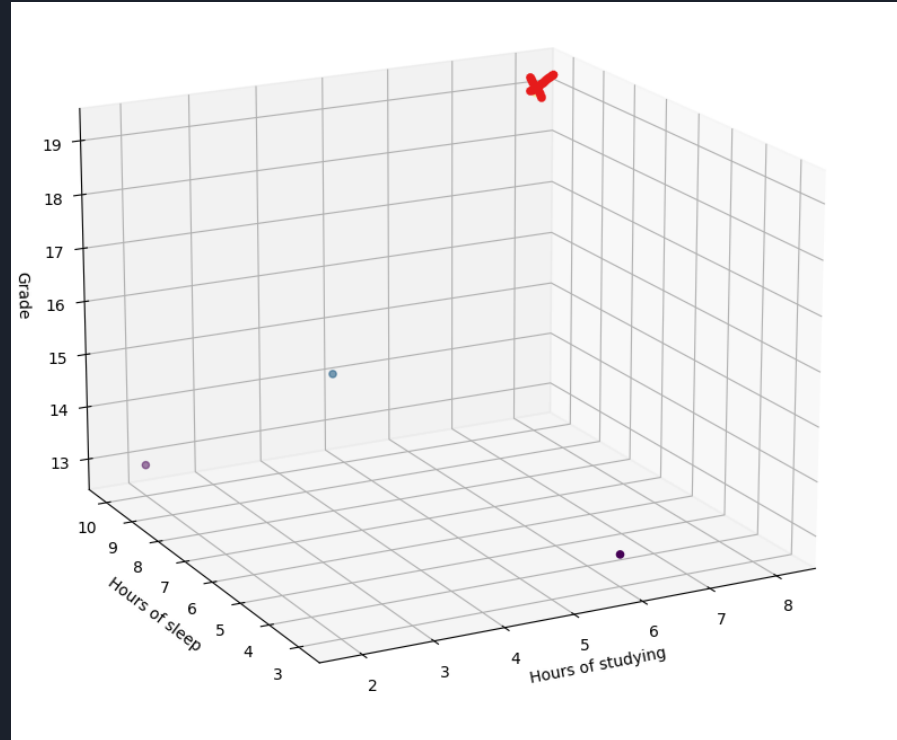
ESIREM - Kévin Naudin



Our simple dataset

- For this example, we'll use a short dataset representing how long a student has studied for a test, how much sleep they got the day before the test, and the grade they received.
- Our input data will be the following (hours of studying, hours of sleep):
 - 4, 8
 - 6, 3
 - 2, 10
 - 8, 10
- And here are the corresponding teacher values (the grade they received):
 - 15
 - 13
 - 13
 - 19

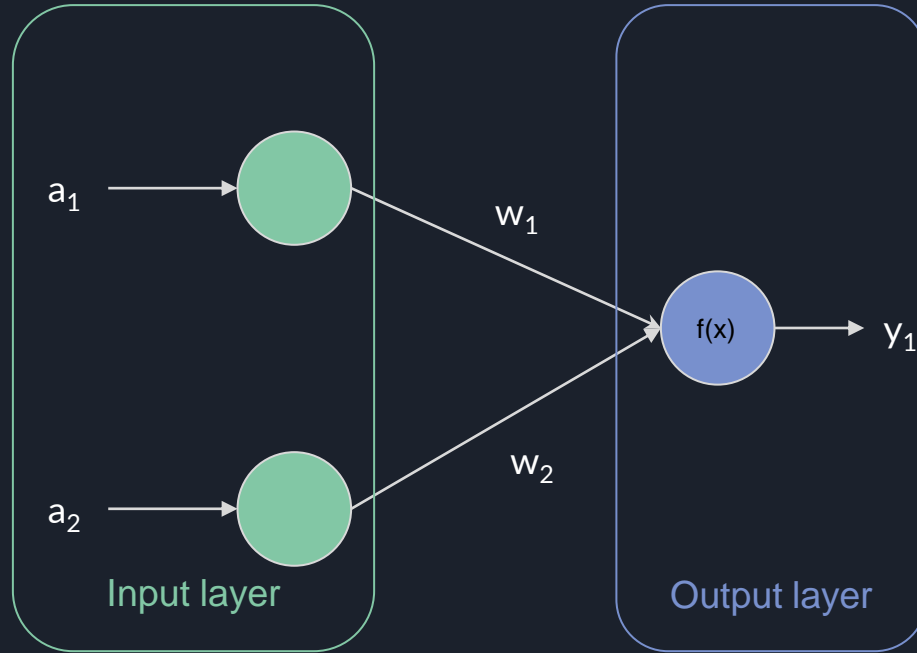
This dataset could be represented as follows



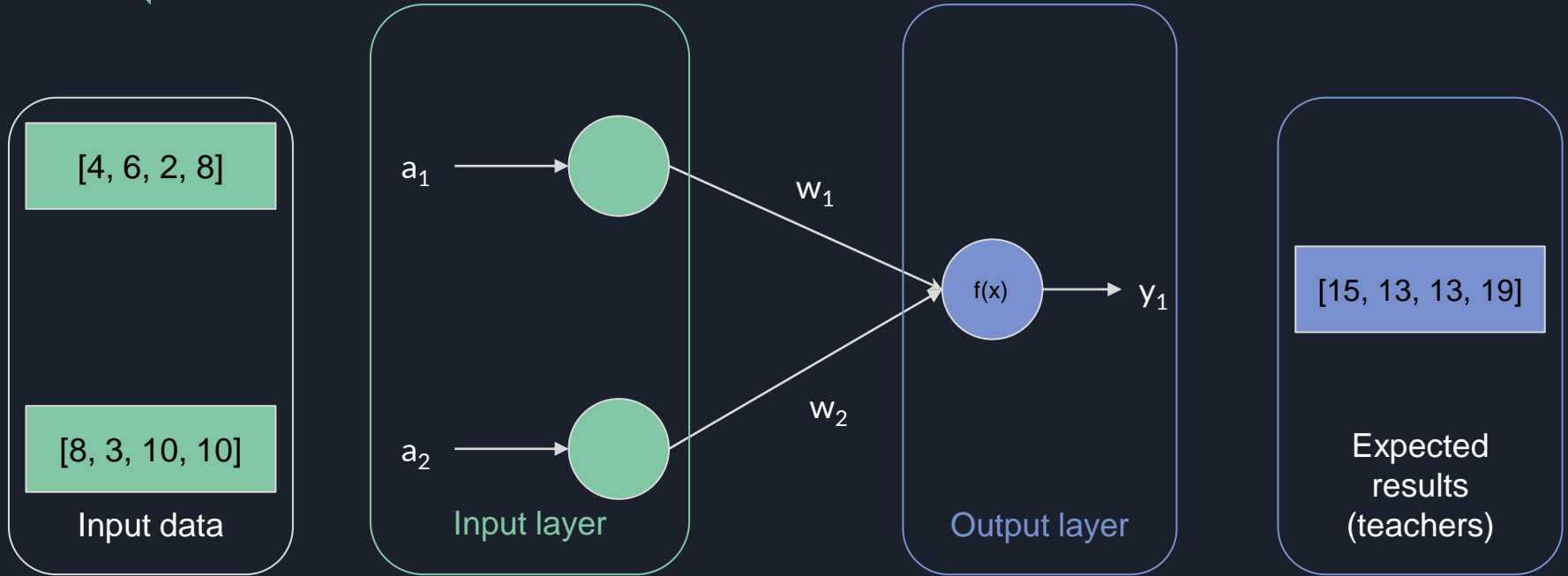
The red mark indicates a point that cannot be clearly seen on this representation



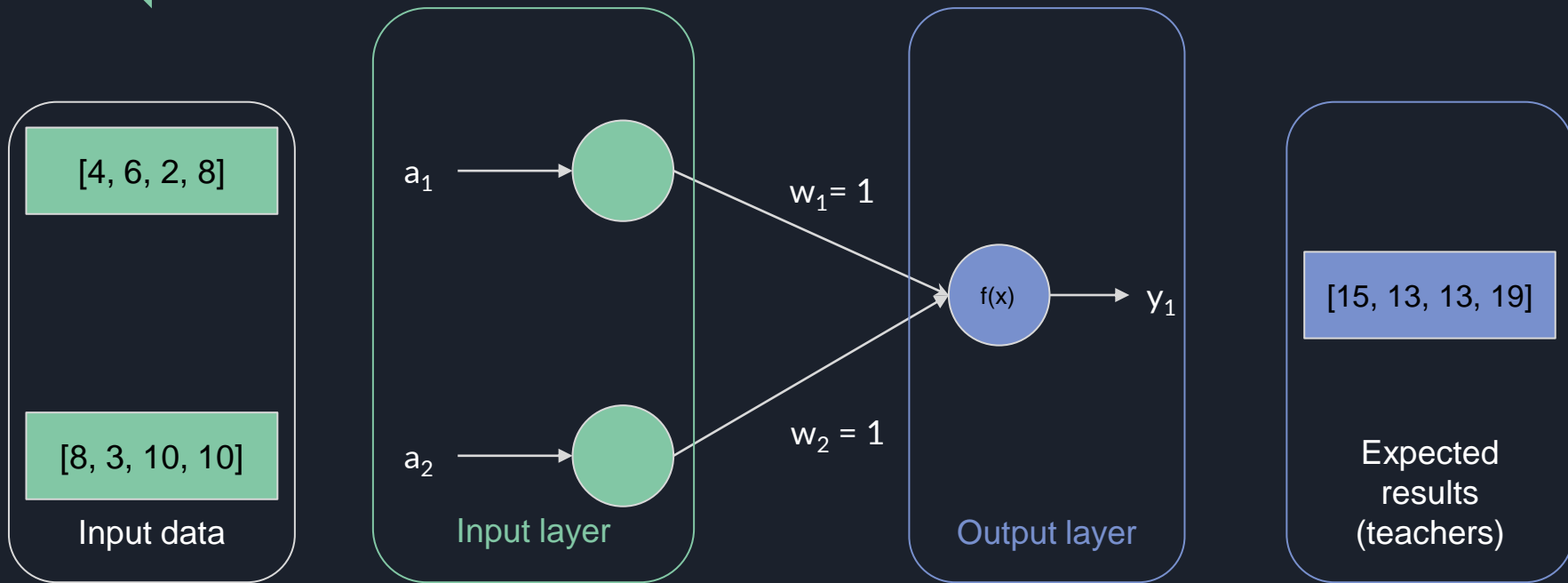
Let's welcome our simple perceptron



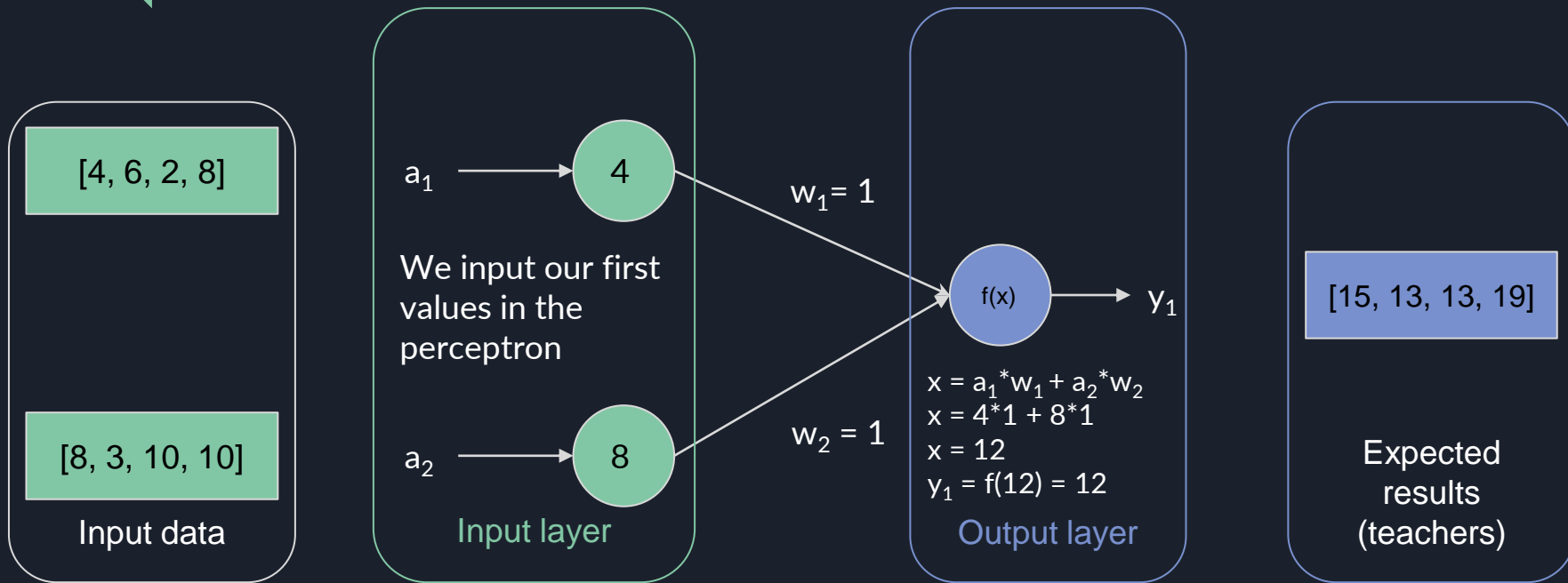
Here are the input values and the expected results added to the graph



For this example we will initialize the weights at 1. Keep in mind that usually they are randomly generated.

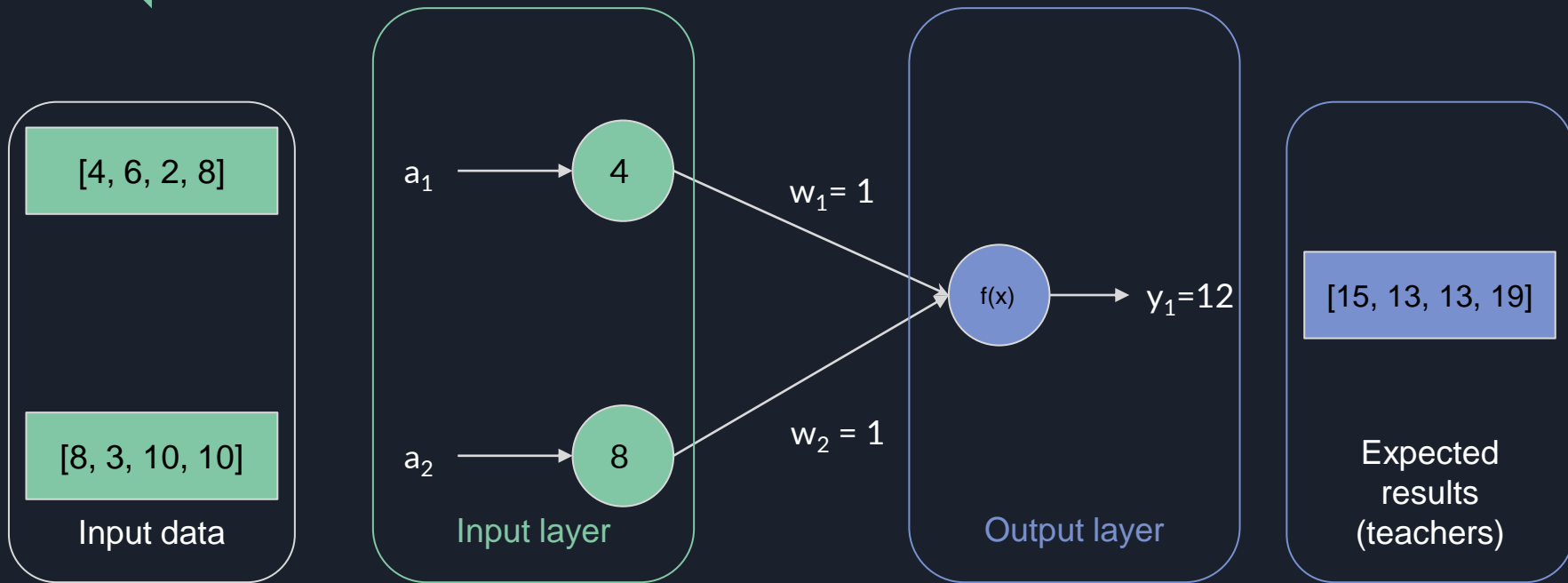


We can now input our first subset of the data in order to compute y_1




In our example, we'll use $f(x) = x$

We can now input our first subset of the data in order to compute y_1



In our example, we'll use $f(x) = x$



We now compare the resulted y value to the expected value (teacher)

We got the predicted value $y=12$ while the expected value (teacher) is $t=15$. That means our neural network got a wrong result and needs its weights to be fixed.

In that case, what is the error value ? Usually, we compare the distance between the two numbers. So in our case, the answer is $e = (y-t) \Rightarrow e = -3$.

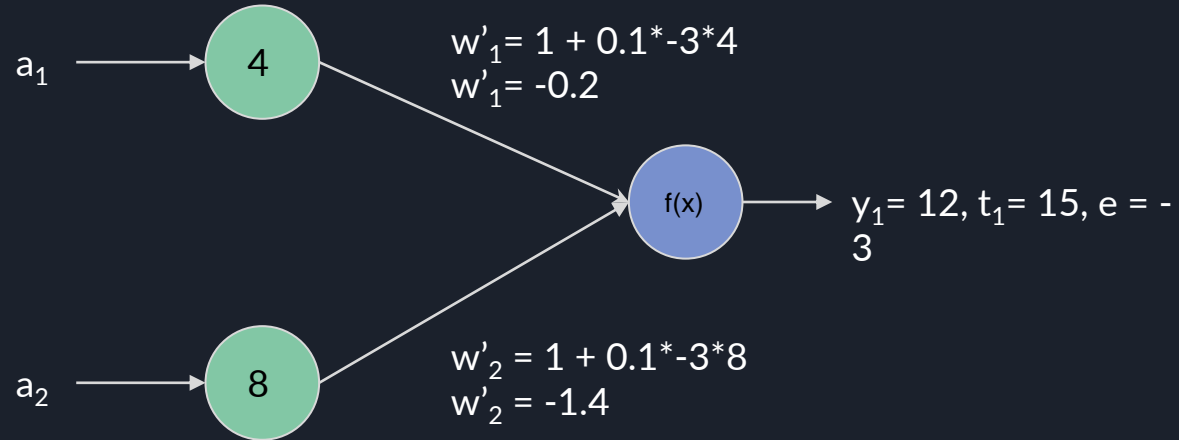
To correct our weights, we propagate that error back to the front of the neural network using Widrow-Hoff's law $w'_i = w_i + \alpha e a_i$

Where:

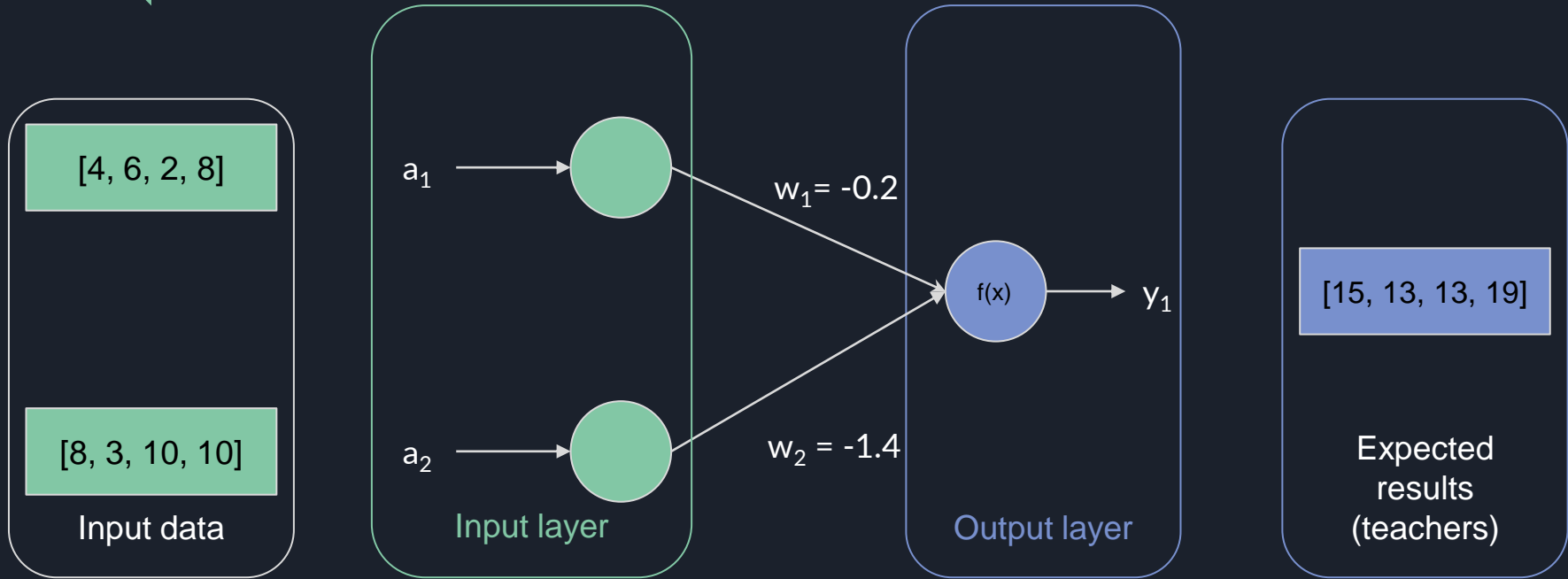
- w'_i is the future value of our weight w_i
- w_i is the weight linking the input unit a_i to the output unit which gave the error e
- α is the learning rate



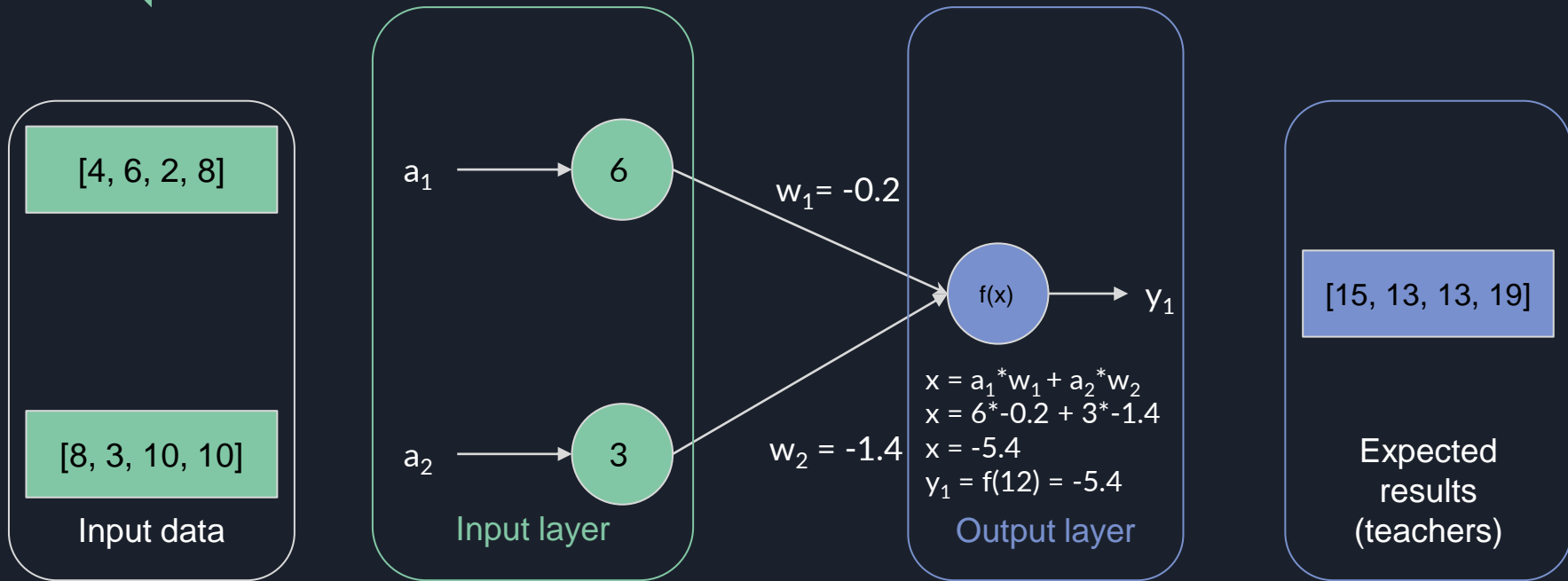
Let's say our learning rate α is equal to 0.1 and then compute the future weights' values



Good ! Now after our first backpropagation, we now have this perceptron

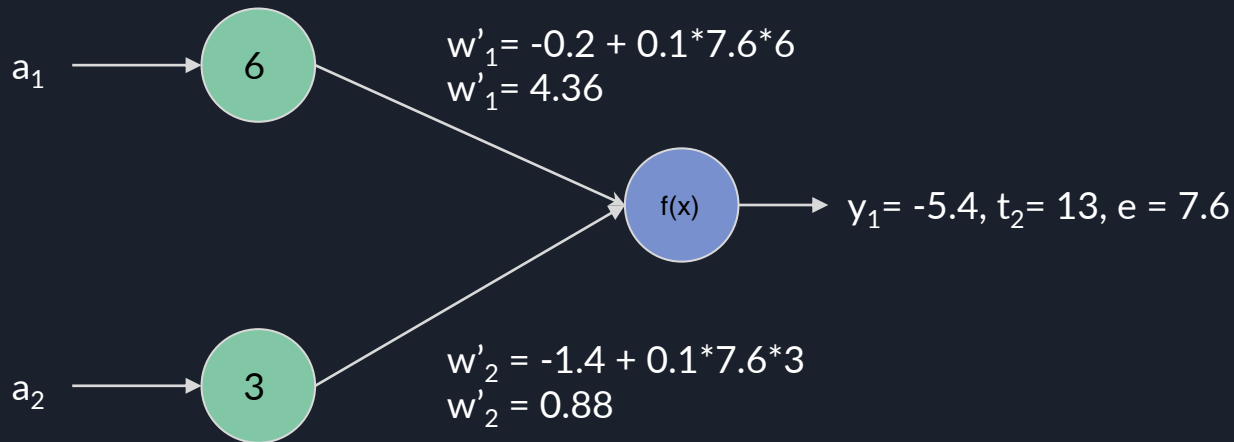


We can now continue the process with the next subset of our data



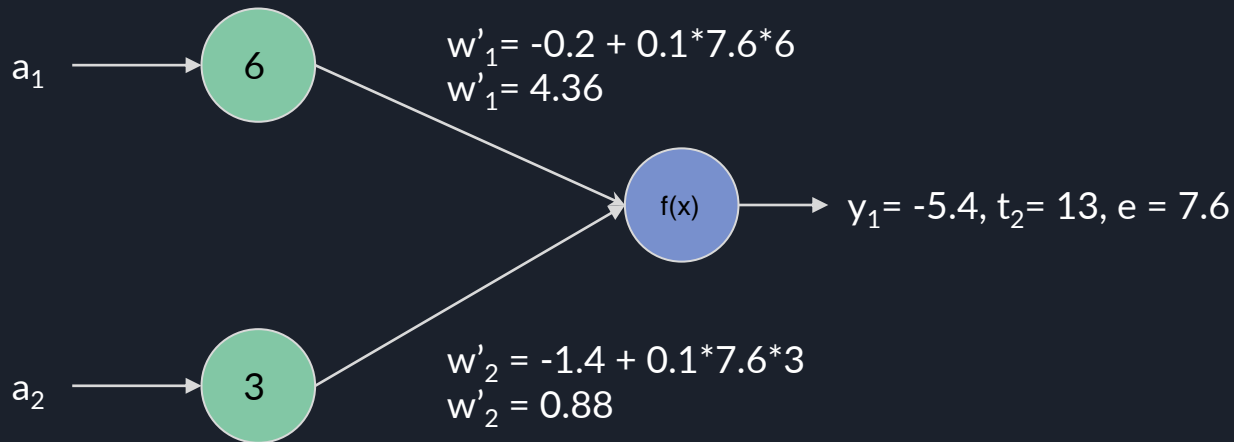


Let's say our learning rate α is equal to 0.1 and then compute the future weights' values



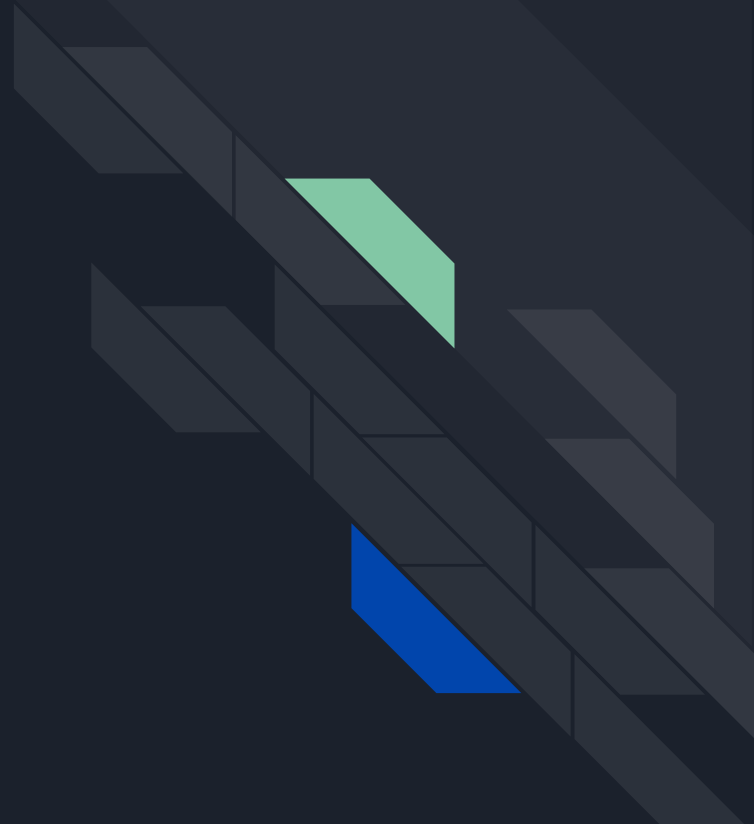


Let's say our learning rate α is equal to 0.1 and then compute the future weights' values

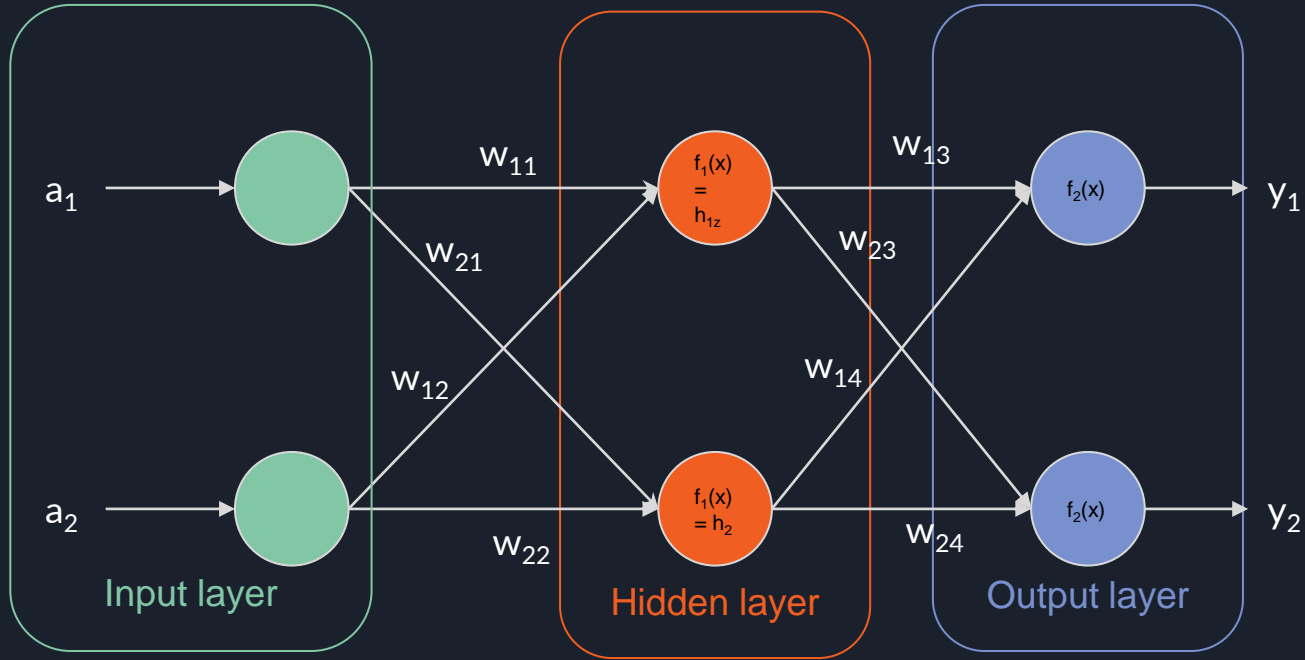


Notice how the more the couple (a, w) impacts the predicted value, how higher it seems to get corrected by the backpropagation

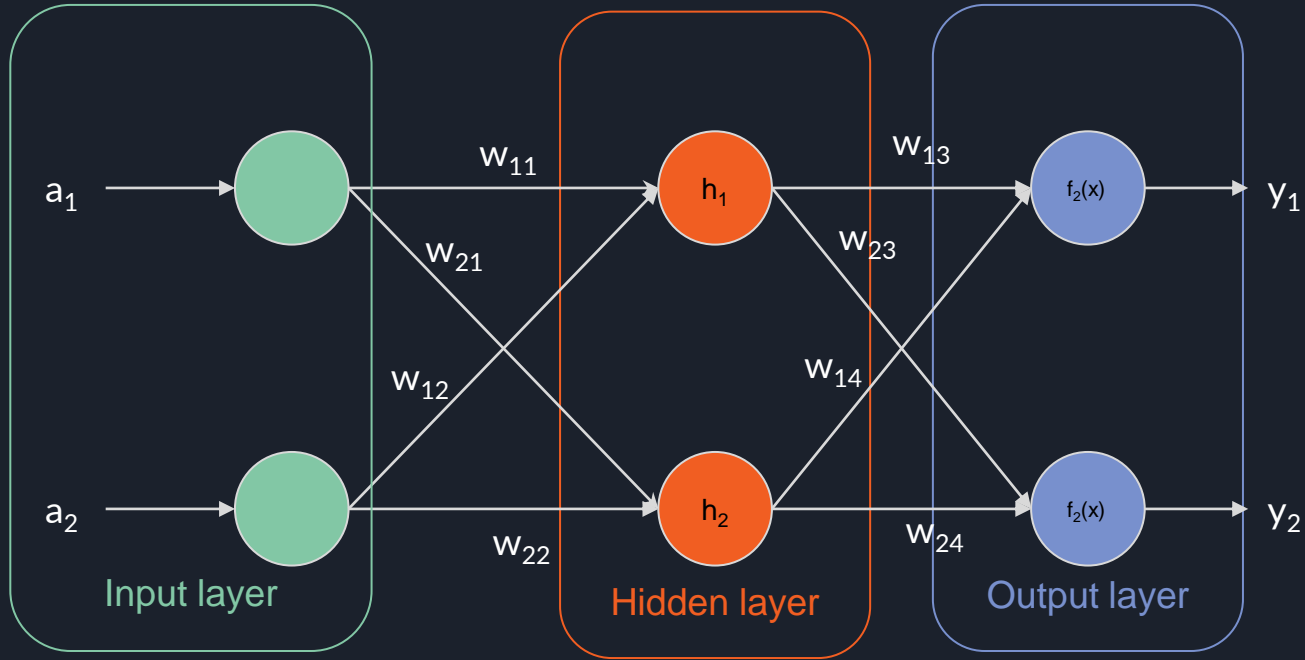
Now what if we add
hidden layers?



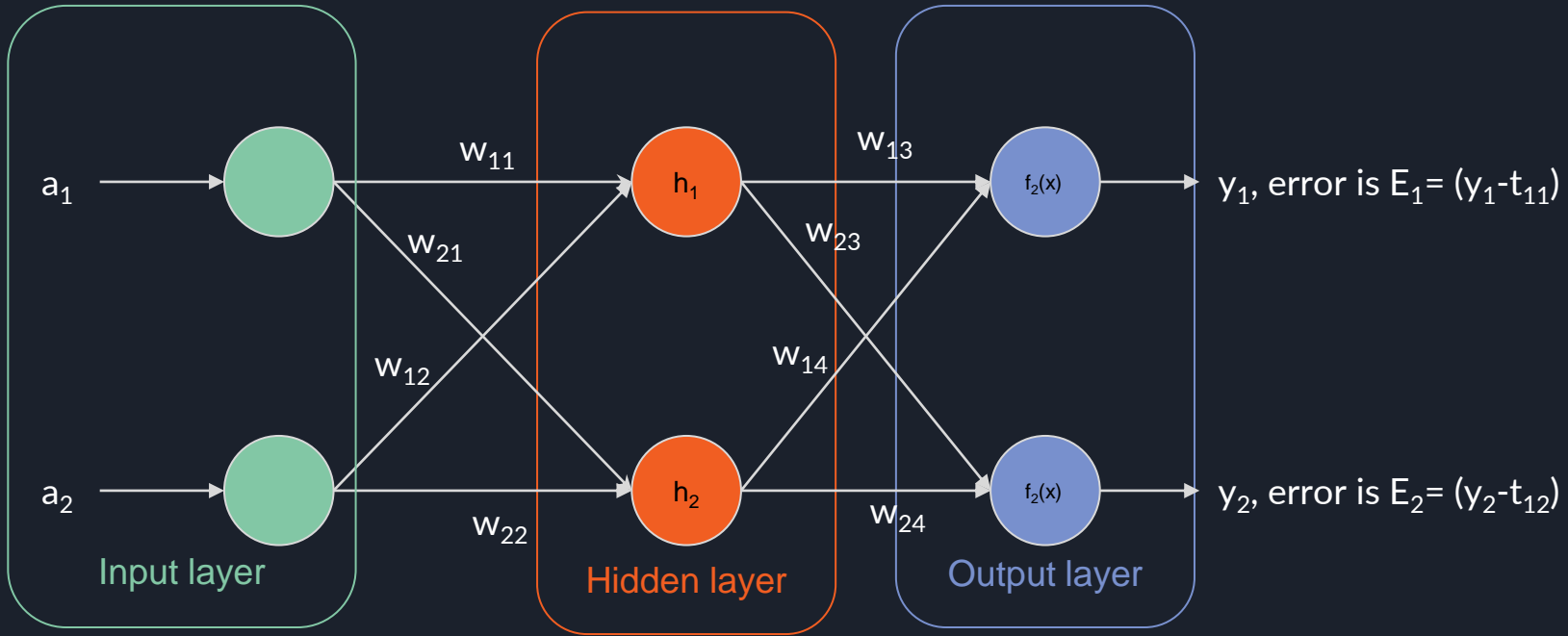
Here is a multi-layer perceptron



Here is a multi-layer perceptron



We know the error value at the end of the network

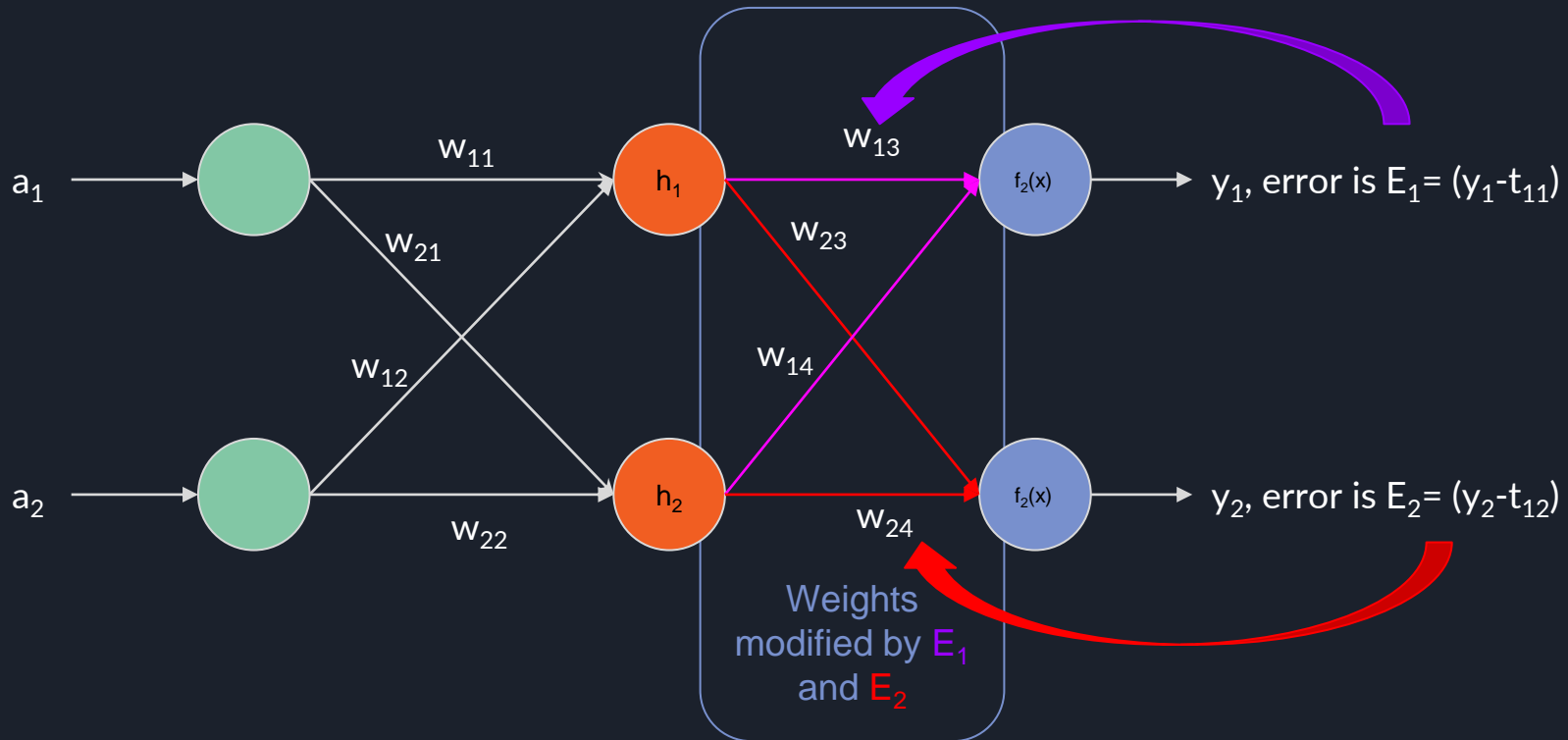




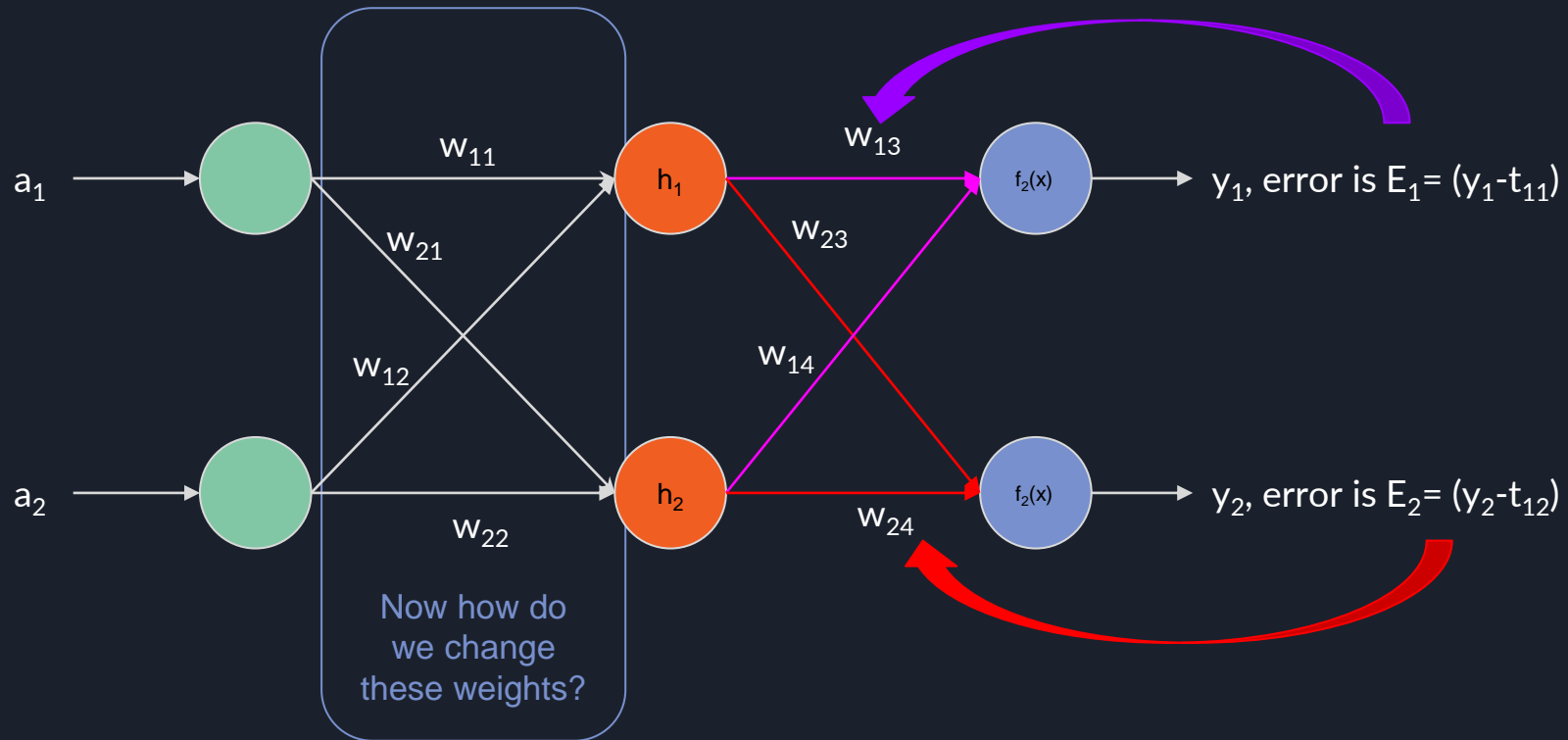
How do we apply backpropagation with E_1 and E_2 ?


When we propagate the error back in the network, it can only affect the weight directly connected to the next unit

The error E is propagated to the weights responsible for its computation



The error E is propagated to the weights responsible for its computation





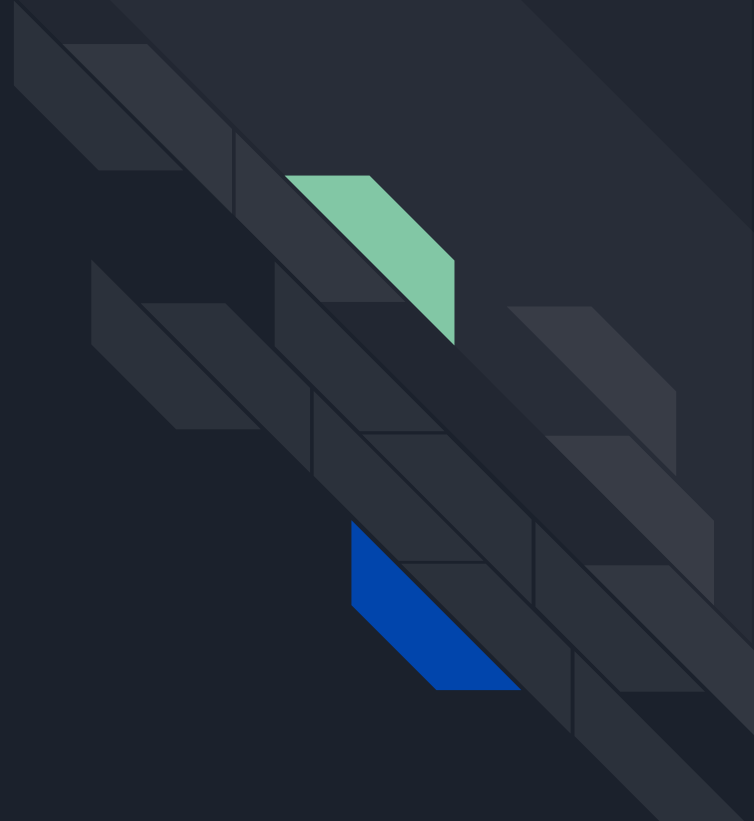
Expected values (teachers) give us what the neural network must output, but...

We do not know the expected values for the hidden layers of our neural network.

We can not use E to modify the weight at the front of the NN, because the weights at the front are not directly responsible for the computation of E .

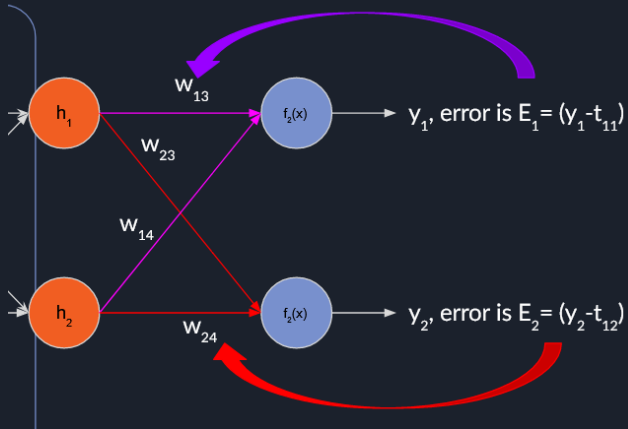
We need to remember something we stated before...

“Notice how the more the couple (a,w) impacts the predicted value, how higher it seems to get corrected by the backpropagation”



Back we go!

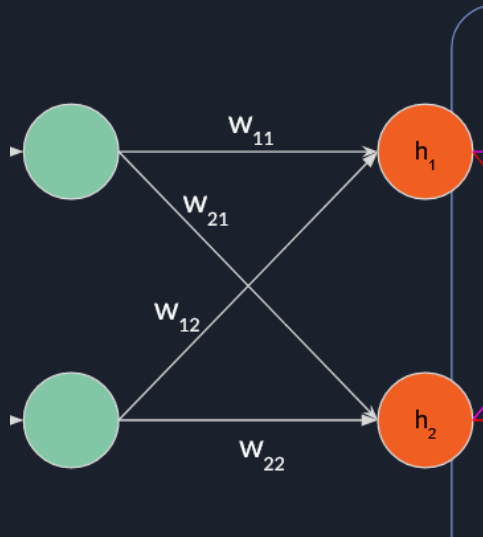
There are multiple ways to propagate E back in our NN, but we'll go for the most simple approach.



If these weights are responsible for the amount of error of E

Back we go!

There are multiple ways to propagate E back in our NN, but we'll go for the most simple approach.

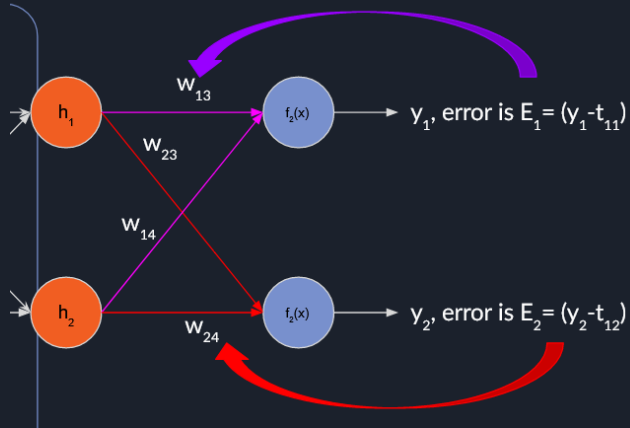


Then these weights must be responsible for the bad prediction of h_1 and h_2 !

Back we go!

We can say that the value of w_{13} and w_{14} are responsible for a portion of the error of E_1

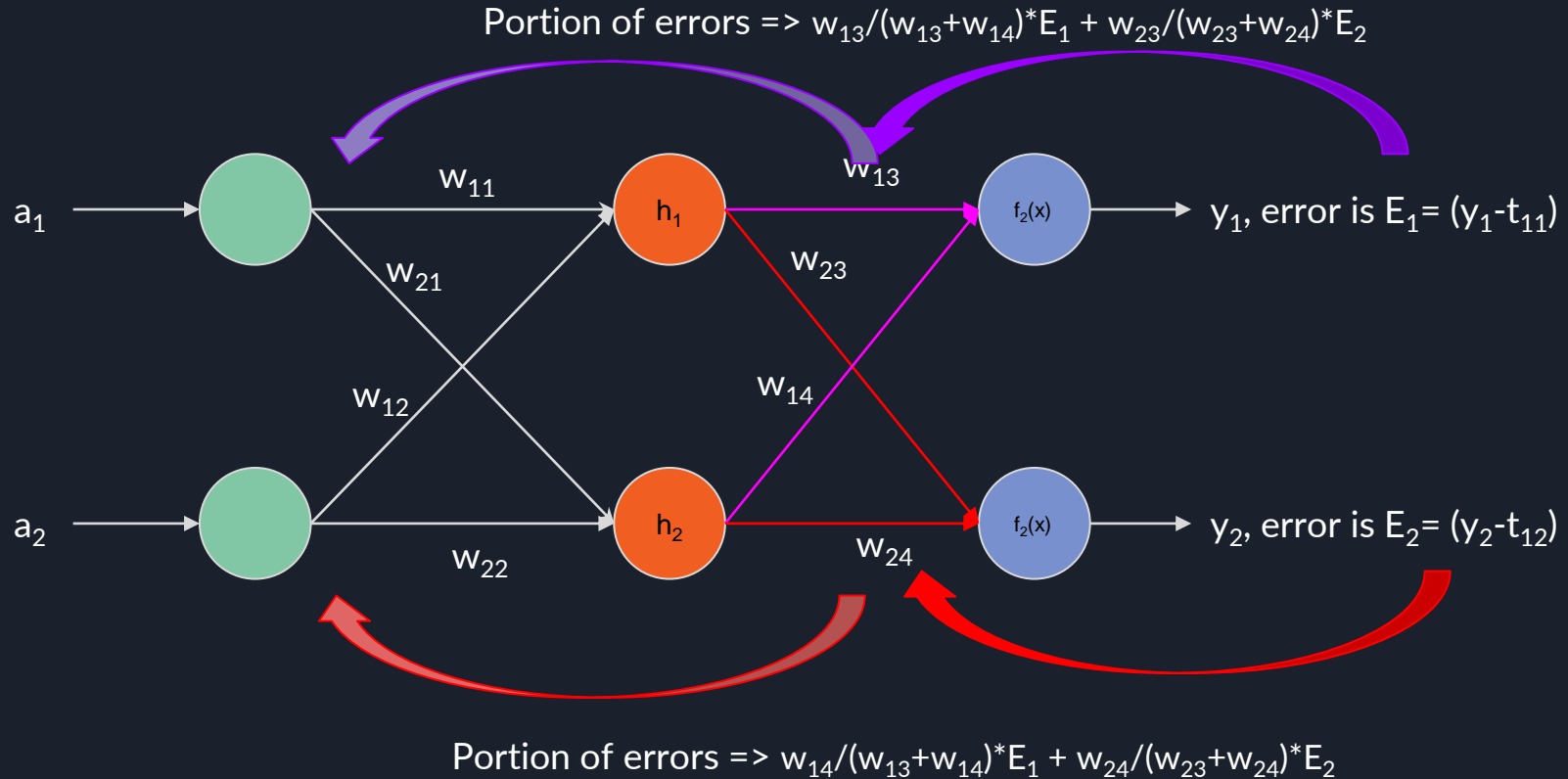
Just like the value of w_{23} and w_{24} are responsible for a portion of the error of E_2



$$e_{h1} = \underbrace{w_{13}/(w_{13}+w_{14})}_{\text{Impact of } h_1 \text{ on } E_1} E_1 + \underbrace{w_{23}/(w_{23}+w_{24})}_{\text{Impact of } h_1 \text{ on } E_1} E_2$$

$$e_{h2} = \underbrace{w_{14}/(w_{13}+w_{14})}_{\text{Impact of } h_2 \text{ on } E_1} E_1 + \underbrace{w_{24}/(w_{23}+w_{24})}_{\text{Impact of } h_2 \text{ on } E_2} E_2$$

As we go back, the error propagated diminishes...





And so the front of the NN is less affected by the error of the back of the NN

