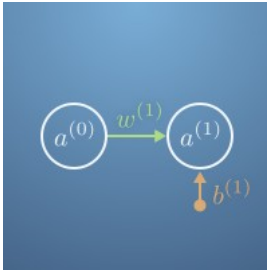


1. Recall from the video the structure of one of the simplest neural networks,

1 / 1 punto



Here there are only two neurons (or nodes), and they are linked by a single edge.

The *activation* of neurons in the final layer, (1), is determined by the activation of neurons in the previous layer, (0),

$$a^{(1)} = \sigma(w^{(1)}a^{(0)} + b^{(1)}),$$

where $w^{(1)}$ is the *weight* of the connection between Neuron (0) and Neuron (1), and $b^{(1)}$ is the *bias* of the Neuron (1). These are then subject to the *activation function*, σ to give the activation of Neuron (1)

Our small neural network won't be able to do a lot - it's far too simple. It is however worth plugging a few numbers into it to get a feel for the parts.

Let's assume we want to train the network to give a *NOT function*, that is if you input 1 it returns 0, and if you input 0 it returns 1.

For simplicity, let's use, $\sigma(z) = \tanh(z)$, for our activation function, and *randomly* initialise our weight and bias to $w^{(1)} = 1.3$ and $b^{(1)} = -0.1$.

Use the code block below to see what output values the neural network initially returns for training data.

```
1 # First we set the state of the network
2 sigma = np.tanh
3 w1 = -5
4 b1 = 5
5
6 # Then we define the neuron activation.
7 def a1(a0):
8     return sigma(w1 * a0 + b1)
9
10 # Finally let's try the network out!
11 # Replace x with 0 or 1 below,
12 a1(0)
13
```

Ejecutar

Restablecer

It's not very good! But it's not trained yet; experiment by changing the weight and bias and see what happens.

Choose the weight and bias that gives the best result for a *NOT function* out of all the options presented.

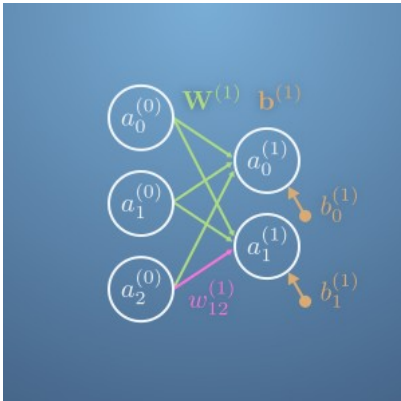
- ☐ $w^{(1)} = 0, b^{(1)} = 5$
- ☐ $w^{(1)} = -3, b^{(1)} = 0$
- ☐ $w^{(1)} = 10, b^{(1)} = 0$
- ☒ $w^{(1)} = -5, b^{(1)} = 5$
- ☐ $w^{(1)} = 3, b^{(1)} = 1$

✓ Correcto

This is the best of all the options presented. We can do better with (for example) $w^{(1)} = -10, b^{(1)} = 10$.

2. Let's extend our simple network to include more neurons.

1 / 1 punto



We now have a slightly changed notation. The neurons which are labelled by their layer with a superscript in brackets, are now also labelled with their number in that layer as a subscript, and form vectors $\mathbf{a}^{(0)}$ and $\mathbf{a}^{(1)}$.

The weights now form a matrix $\mathbf{W}^{(1)}$, where each element, $w_{ij}^{(1)}$, is the link between the neuron j in the previous layer and neuron i in the current layer. For example $w_{12}^{(1)}$ is highlighted linking $a_2^{(0)}$ to $a_1^{(1)}$.

The biases similarly form a vector $\mathbf{b}^{(1)}$.

We can update our activation function to give,

$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}^{(1)} \mathbf{a}^{(0)} + \mathbf{b}^{(1)}),$$

where all the quantities of interest have been *upgraded* to their vector and matrix form and σ acts upon each element of the resulting weighted sum vector separately.

For a network with weights, $\mathbf{W}^{(1)} = \begin{bmatrix} -2 & 4 & -1 \\ 6 & 0 & -3 \end{bmatrix}$, and bias $\mathbf{b} = \begin{bmatrix} 0.1 \\ -2.5 \end{bmatrix}$,

calculate the output, $\mathbf{a}^{(1)}$, given an input vector,

$$\mathbf{a}^{(0)} = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.1 \end{bmatrix}$$

You may do this calculation either by hand (to 2 decimal places), or by writing python code. Input your answer into the code block below.

(If you chose to code, remember that you can use the @ operator in Python to perform operate a matrix on a vector.)

```
1 # First set up the network.
2 sigma = np.tanh
3 W = np.array([[ -2,  4, -1], [ 6,  0, -3]])
4 b = np.array([0.1, -2.5])
5
6 # Define our input vector
7 x = np.array([0.3, 0.4, 0.1])
8
9 # Calculate the values by hand,
10 # and replace a1_0 and a1_1 here (to 2 decimal places)
11 # (Or if you feel adventurous, find the values with code!)
12 a1 = np.array([0.76159416, -0.76159416])
13
```

Ejecutar

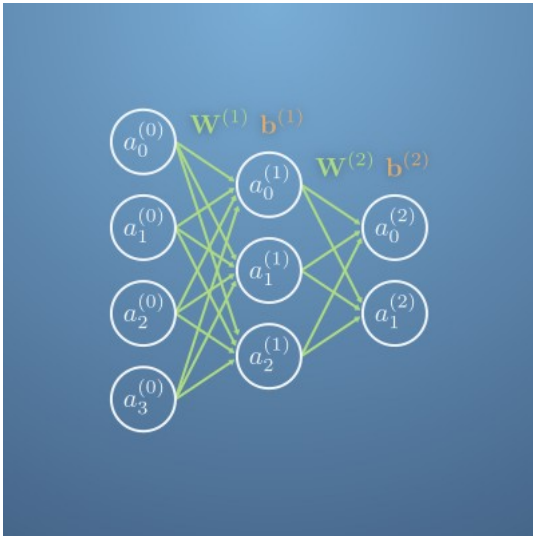
Restablecer

✓ Correcto

Well done. Why not try again using the other method.

3. Now let's look at a network with a *hidden layer*.

1 / 1 punto



Here, data is input at layer (0), this activates neurons in layer (1), which become the inputs for neurons in layer (2).

(We've stopped explicitly drawing the biases here.)

Which of the following statements are true?

- ☐ None of the other statements.
- ☐ This neural network has 9 biases.
- ☐ This network can always be replaced with another one with the same amount of input and output neurons, but no hidden layers.
- ☒ This neural network has 5 biases.

✓ **Correcto**

In general there are as many biases as there are output and hidden neurons.

- ☒ The number of weights in a layer is the product of the input and output neurons to that layer.

✓ **Correcto**

This gives us 12 weights in the first layer, and 6 weights in the second.

- ☐ The number of weights in a layer is the sum of the input and output neurons to that layer plus 1.

4. Which of the following statements about the neural network from the previous question are true?

1 / 1 punto

- ☒ $\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(2)}(\mathbf{W}^{(1)} \mathbf{a}^{(0)} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}),$

✓ **Correcto**

In this form, the entire function of the neural network is shown, with each layer chained together. This is the same as,

$$\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(2)} \mathbf{a}^{(1)} + \mathbf{b}^{(2)})$$

$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}^{(1)} \mathbf{a}^{(0)} + \mathbf{b}^{(1)}).$$

but in a single statement.

☐ $\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(2)} \mathbf{W}^{(1)} \mathbf{a}^{(0)} + \mathbf{W}^{(2)} \mathbf{b}^{(1)} + \mathbf{b}^{(2)})$

☐ None of the other statements.

☐ $\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(2)}),$