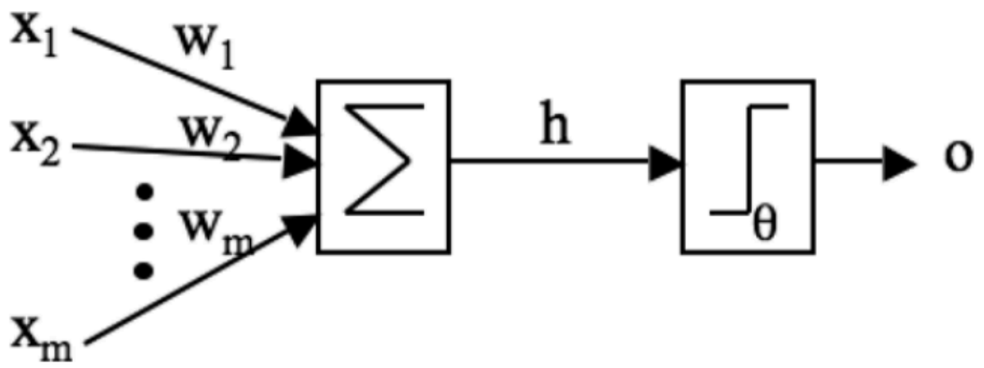
Feed-forward neural networks and backpropagation

Recap on Perceptron

The Perceptron



1. A set of inputs: $x_1, x_2, \ldots x_m$
2. A set of weights: $w_1, w_2, \ldots w_m$
3. An adder:

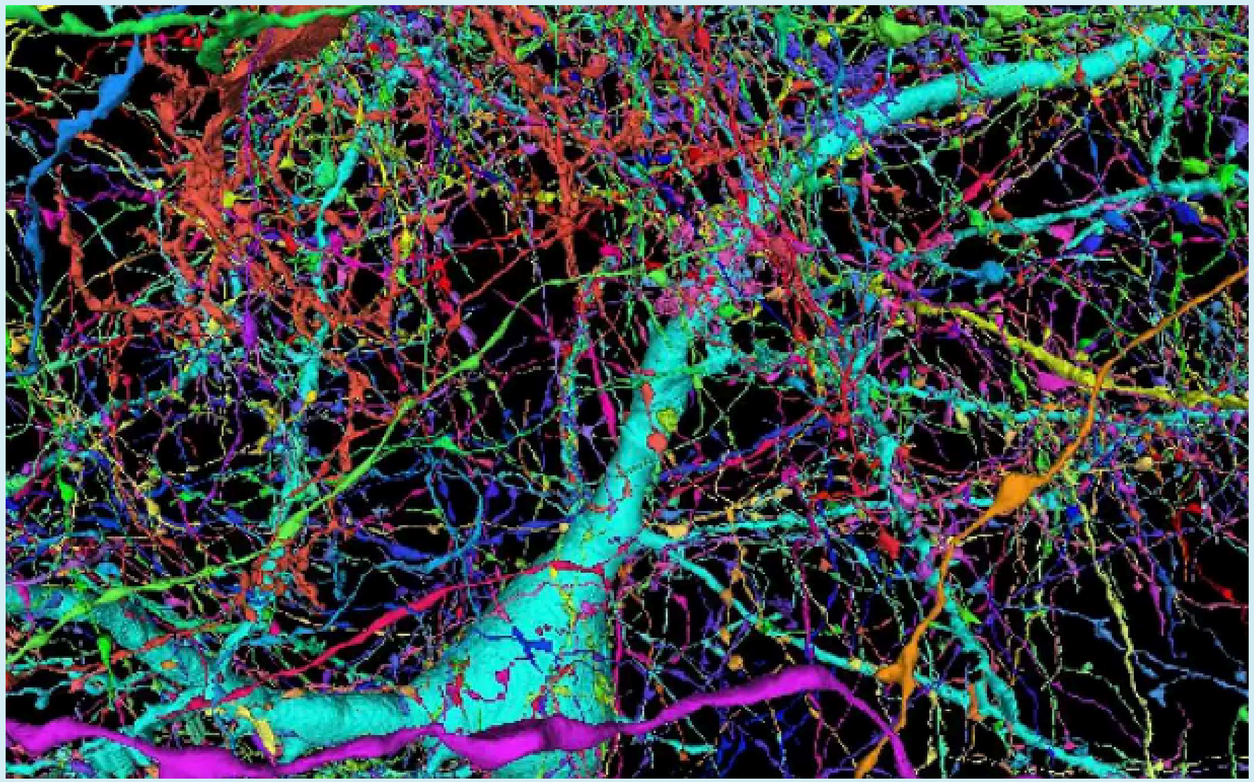
$$h = \sum_{i=1}^{m} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

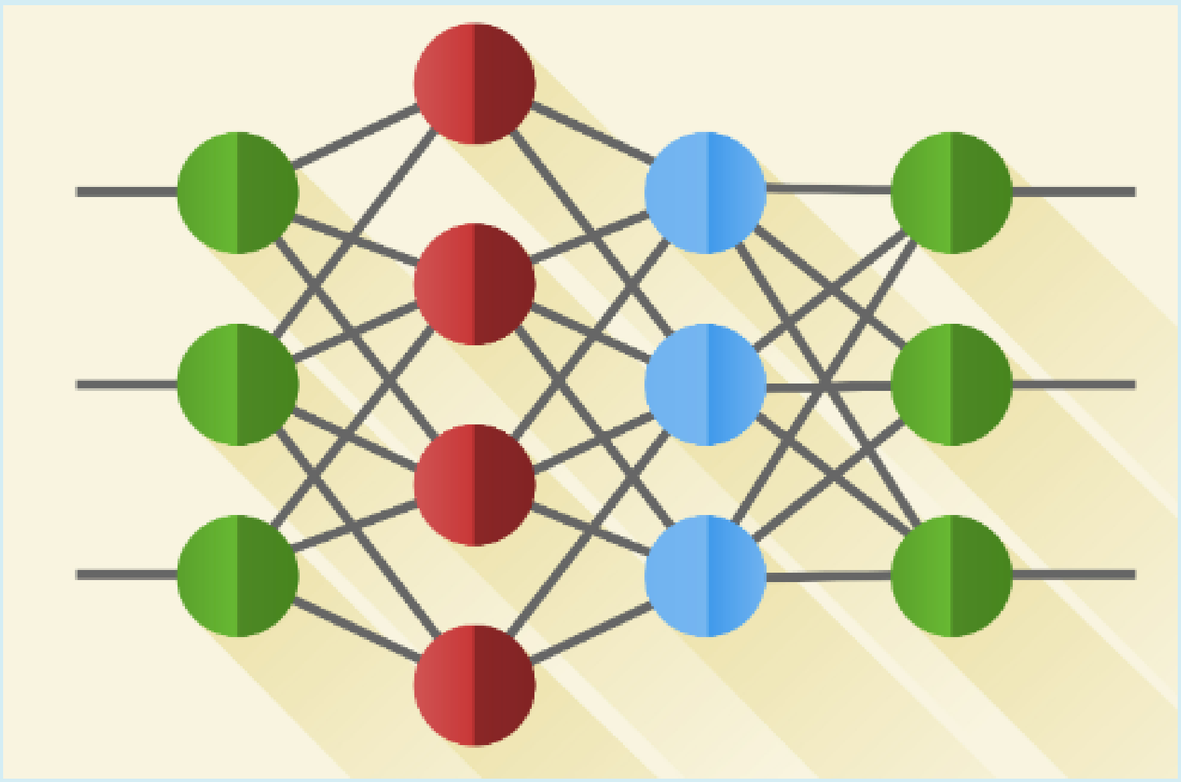
4. An activation function,
   Originally a step function:

$$o = g(h) = \begin{cases} 1 \; if \; h > \theta \\ 0 \; if \; h \leq \theta \end{cases}$$

General idea on neural netowrks

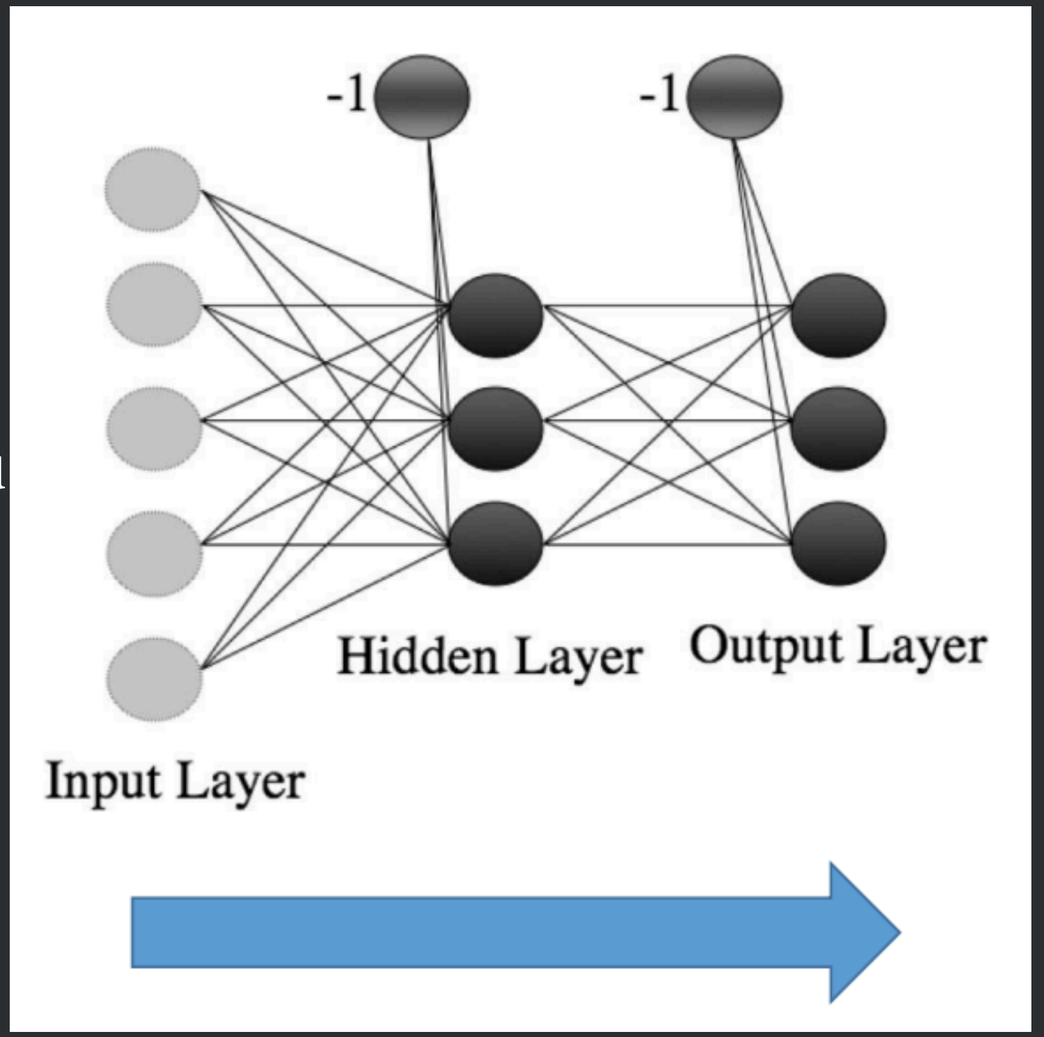
A neural network is a machine learning program, or model, that makes decisions in a manner similar to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions.





Feed-forward network structure

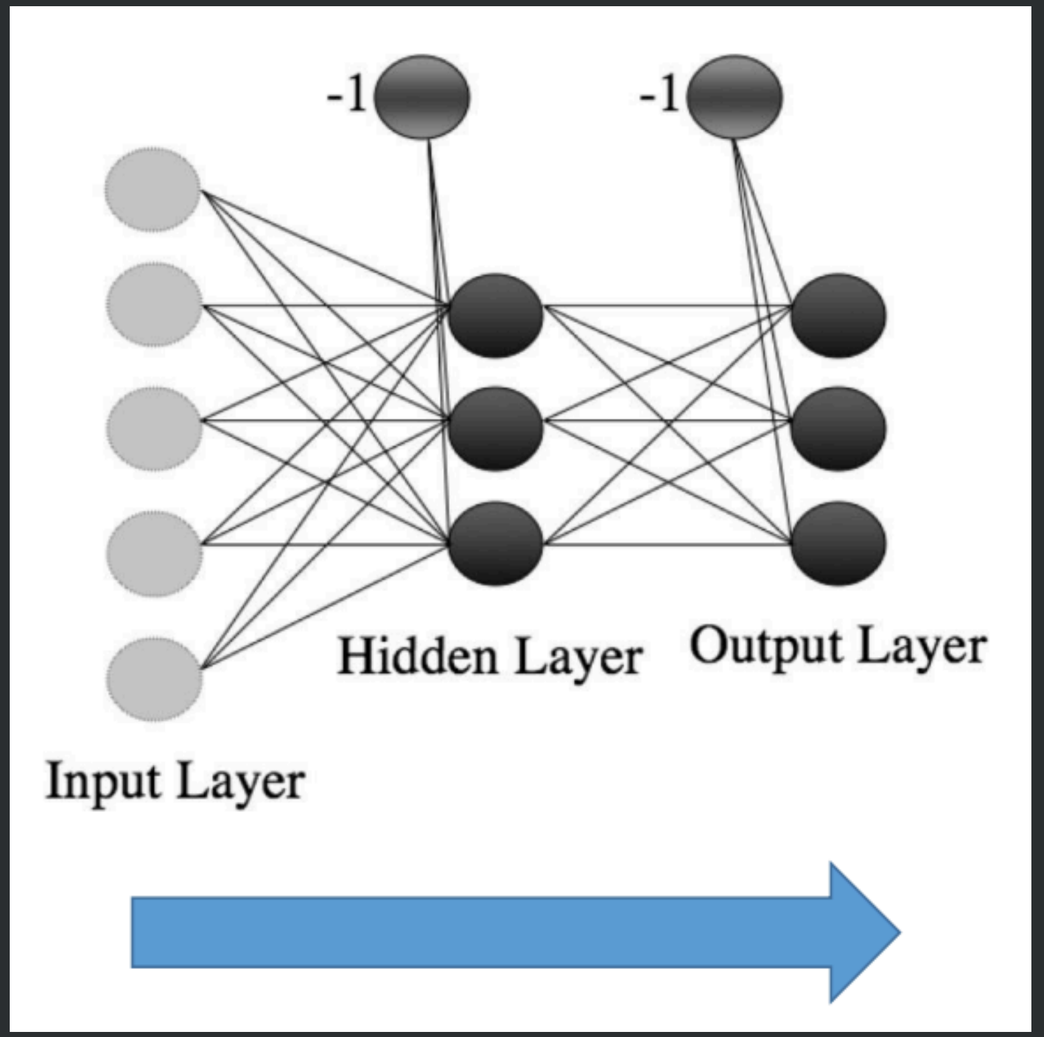
- Feed-forward network, also called Multi-layer Perceptron is the simplest artificial intelligence model
- In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes, and to the output nodes



Feed-forward network structure

- Input Layer: This layer consists of neurons that receive inputs and pass them on to the next layer.
- Hidden Layers: These layers are not exposed to the input or output and can be considered as the computational engine of the neural network. Each hidden layer's neurons take the weighted sum of the outputs from the previous layer, apply an activation function, and pass the result to the next layer.
- Output Layer: The final layer that produces the output for the given inputs.

Each neuron in one layer is connected to every neuron in the next layer. The strength of the connection between neurons is represented by weights, and learning in a neural network involves updating these weights.



Going forward

The process of making predictions

1. The input layer of the neural network receives the input features of the data.

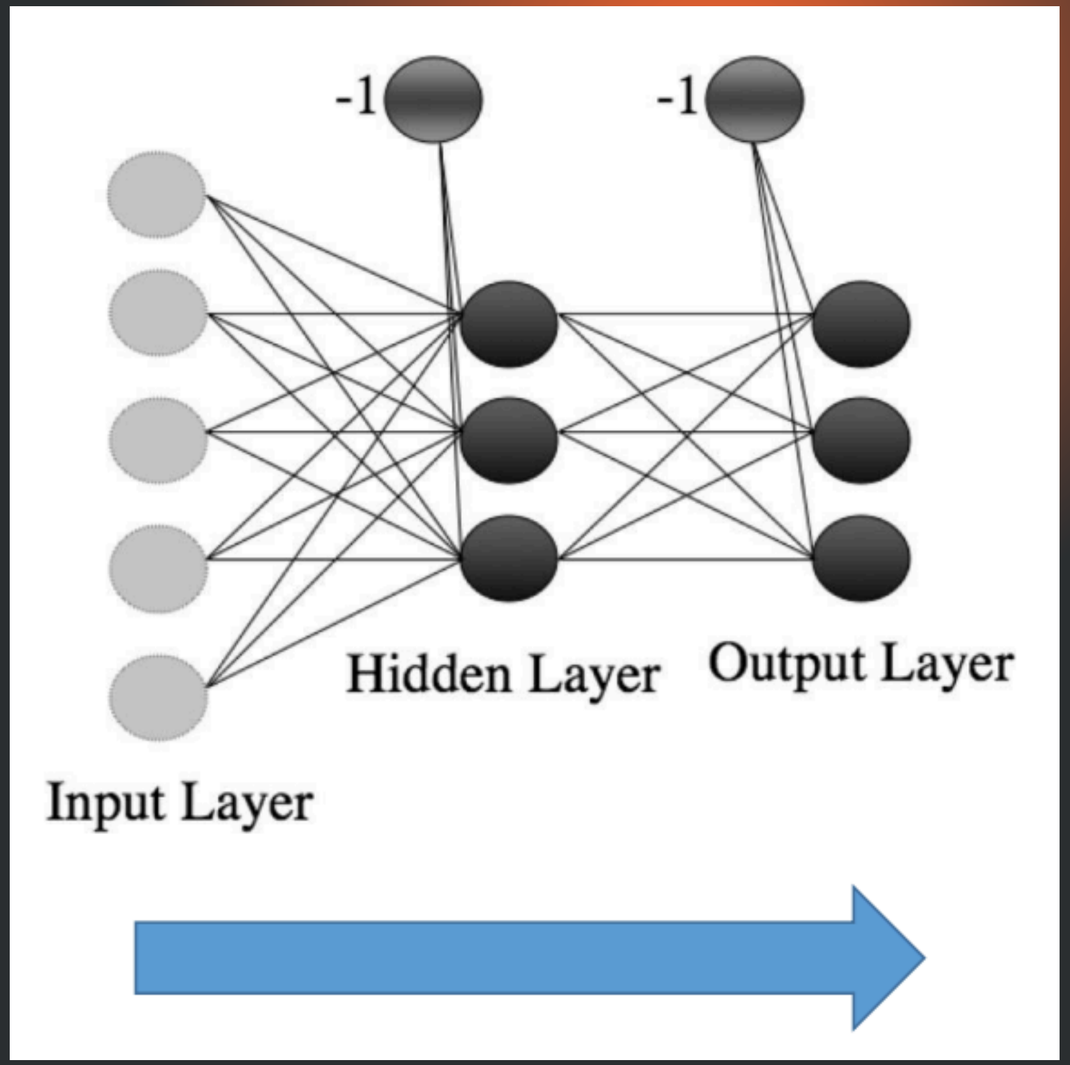
2. Bias Inclusion: In addition to the weighted sum of the inputs, a bias term is added.

3. The input values are multiplied with the weights and summed into each hidden node.

4. The weighted sum is then passed through an activation function.

5. The output from the activation function becomes the input for the next layer.

6. The final output of the neural network is the result of the forward pass through all the layers.

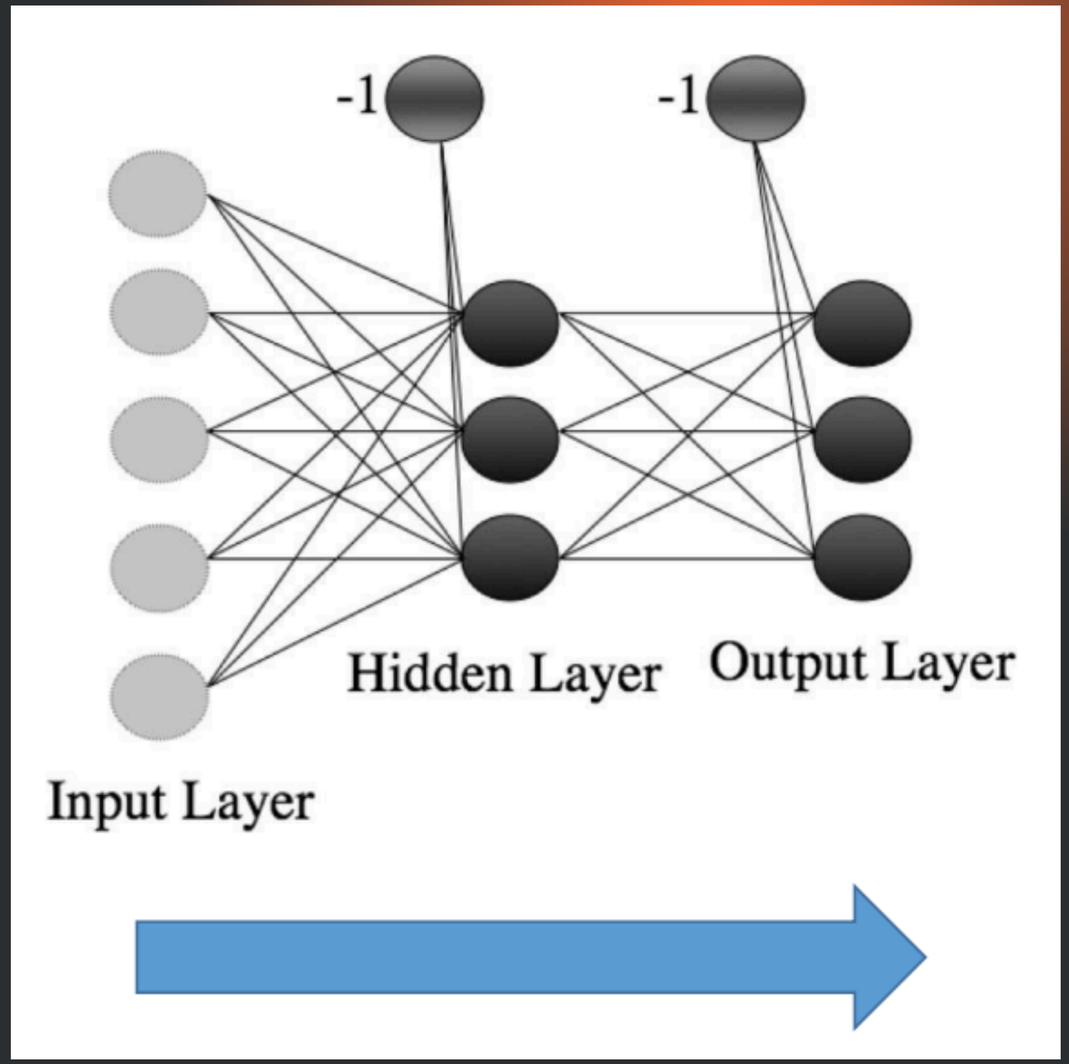


Hidden Layer

A nonlinear activation function is used in the hidden layers of a neural network.

Typical activation functions:

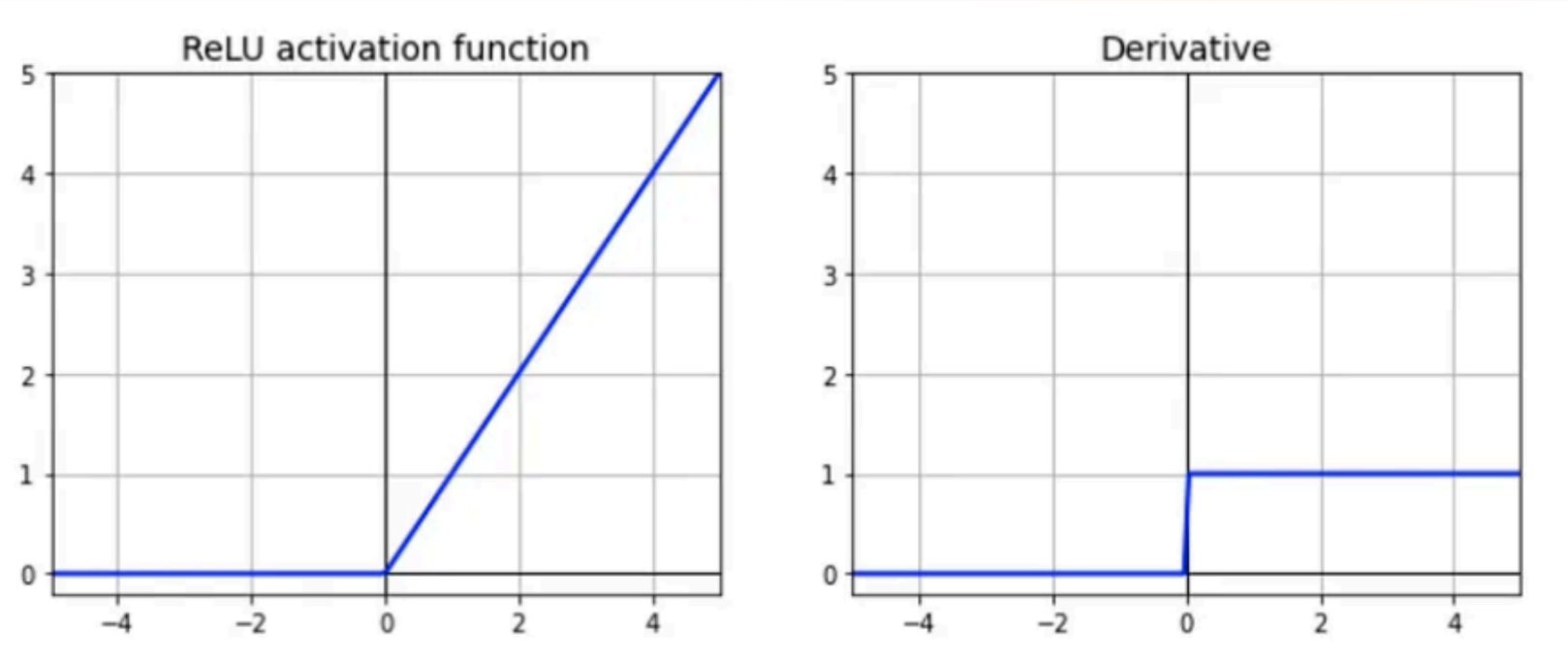
- Rectified Linear Activation (ReLU)
- Logistic (Sigmoid)
- Hyperbolic Tangent (Tanh)



Rectified Linear Activation (ReLU)

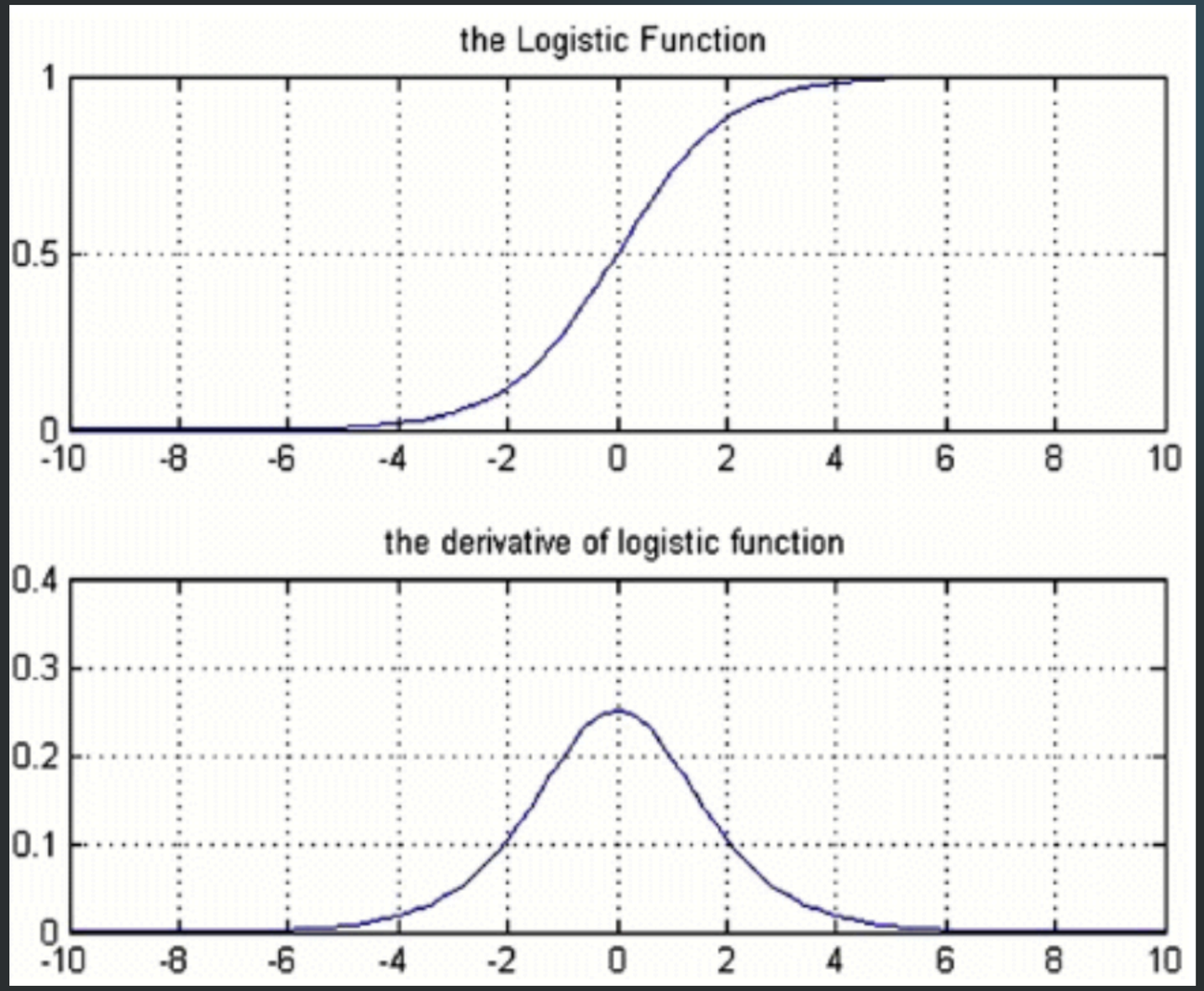
- returns o if the input is negative, but for any positive input, it returns that value back.

$$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$



The Logistic Function

- $y = \sigma(z) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
- Differentiable
- $y' = y(1-y)$

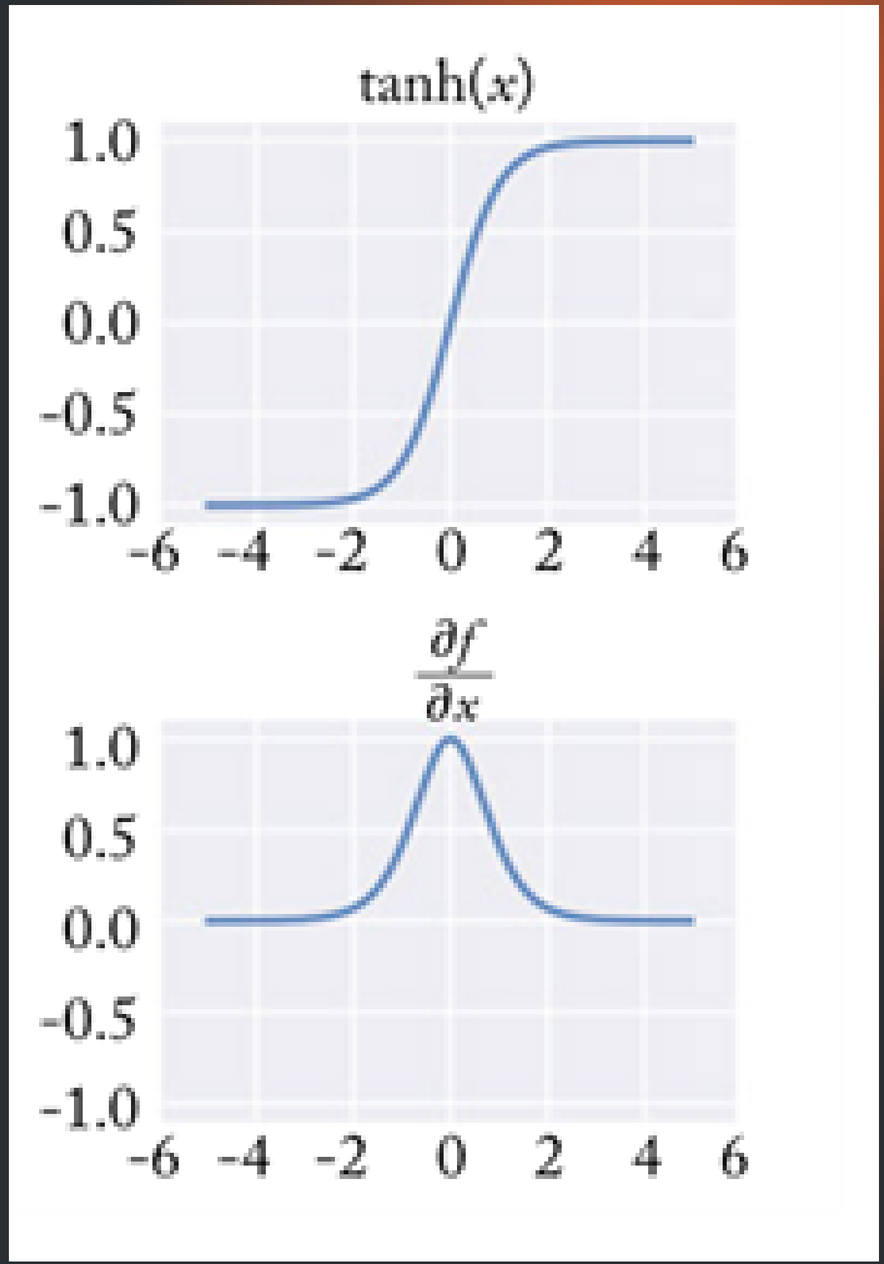


Hyperbolic Tangent (Tanh)

Hyperbolic Tangent function

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{d}{dx}\tanh(x) = 1 - (\tanh(x))^2$$

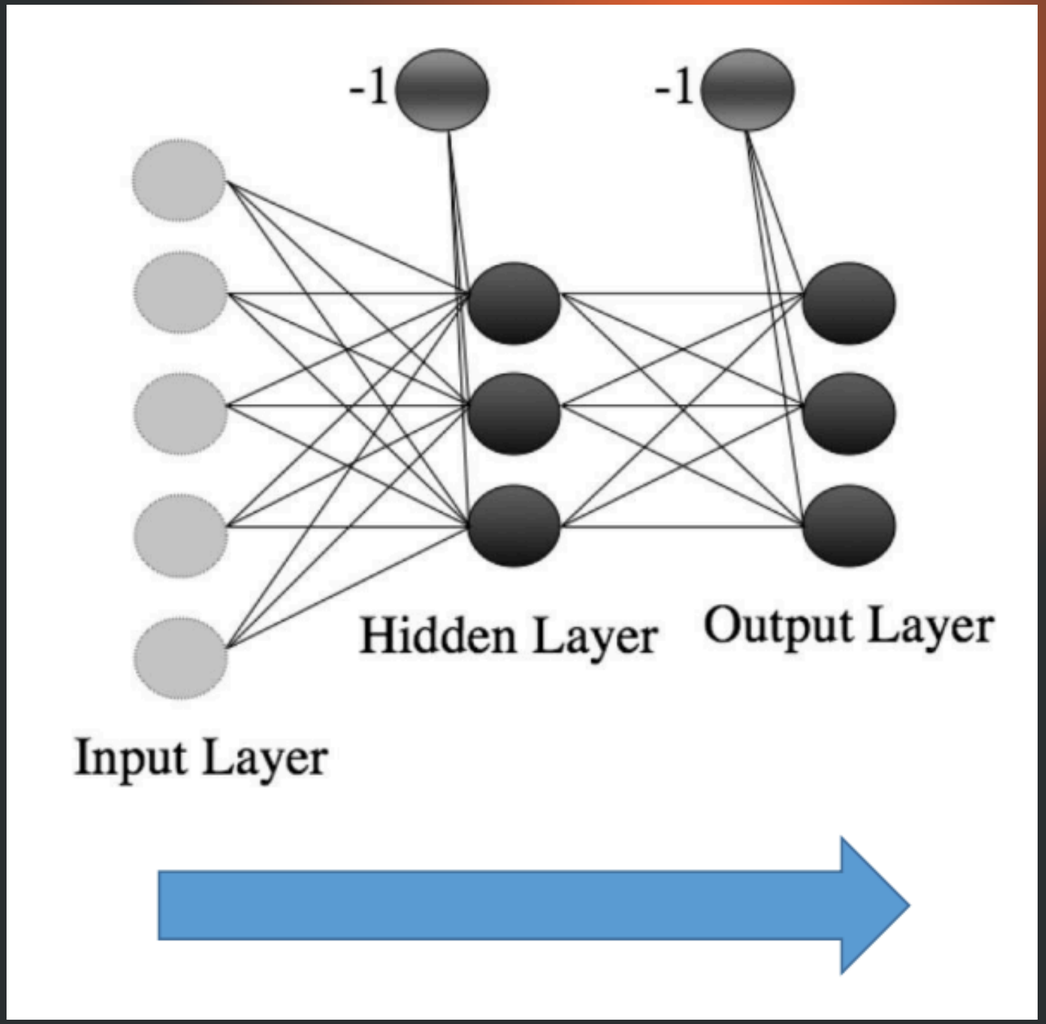


The output layer

Apply the activation function to the weighted sum to obtain the output of each neuron in the output layer.

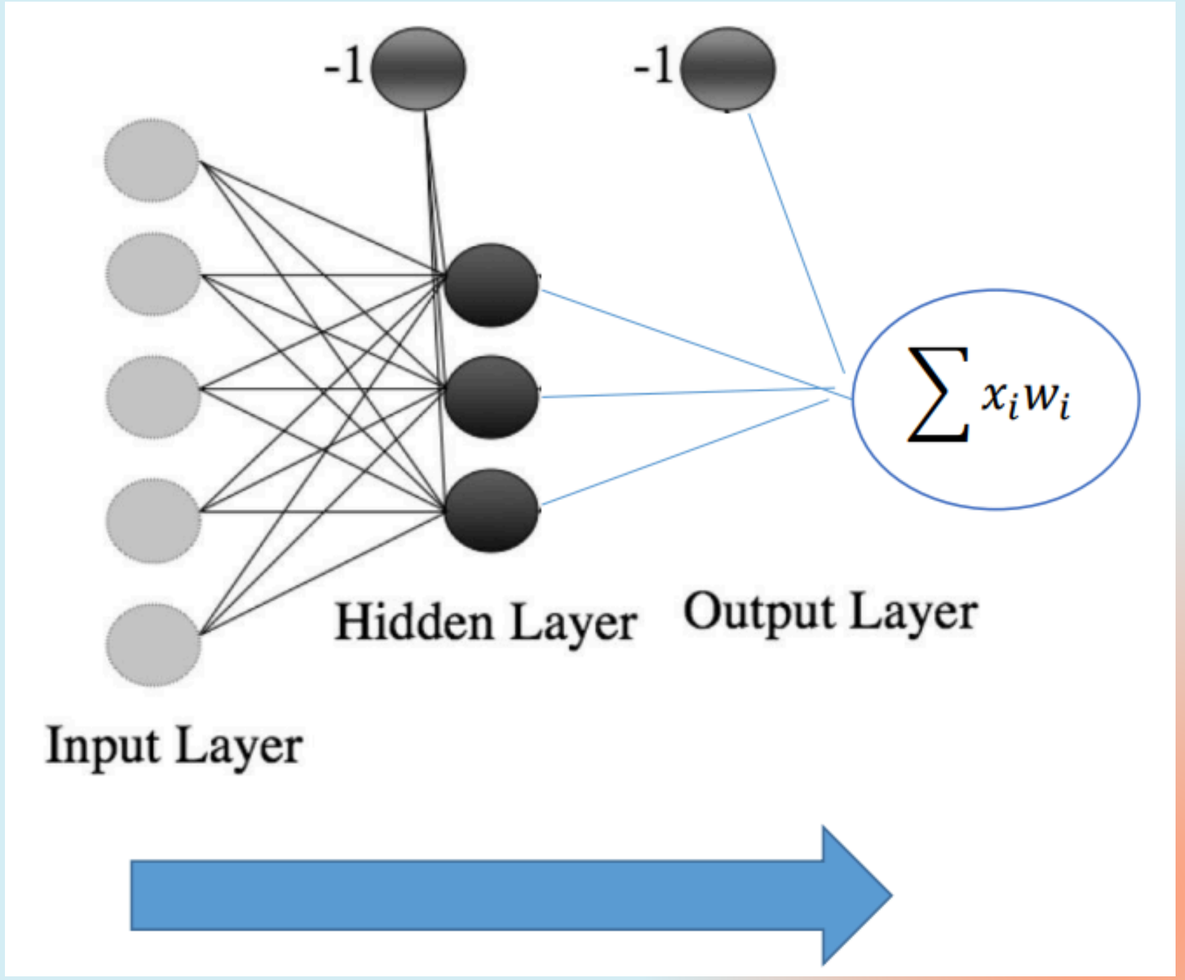
Several possibilities, depending on the task, including:

1. Regression
2. Binary classification
3. Multi-label classification
4. Multi-class classification



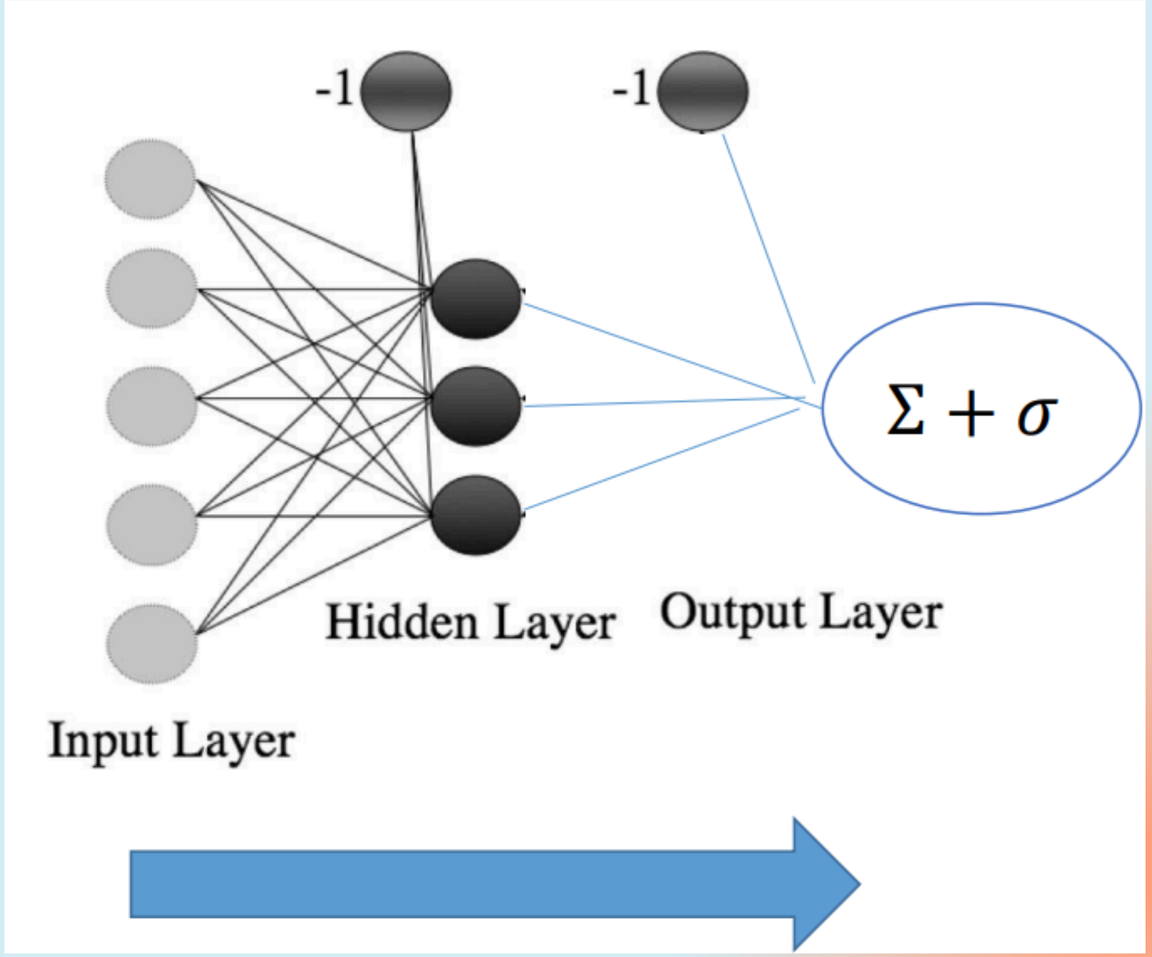
Regression

- The output layer contains a single neuron, representing the continuous output value. No activation function is typically applied in the output layer for regression tasks.
- This can predict non-linear functions.



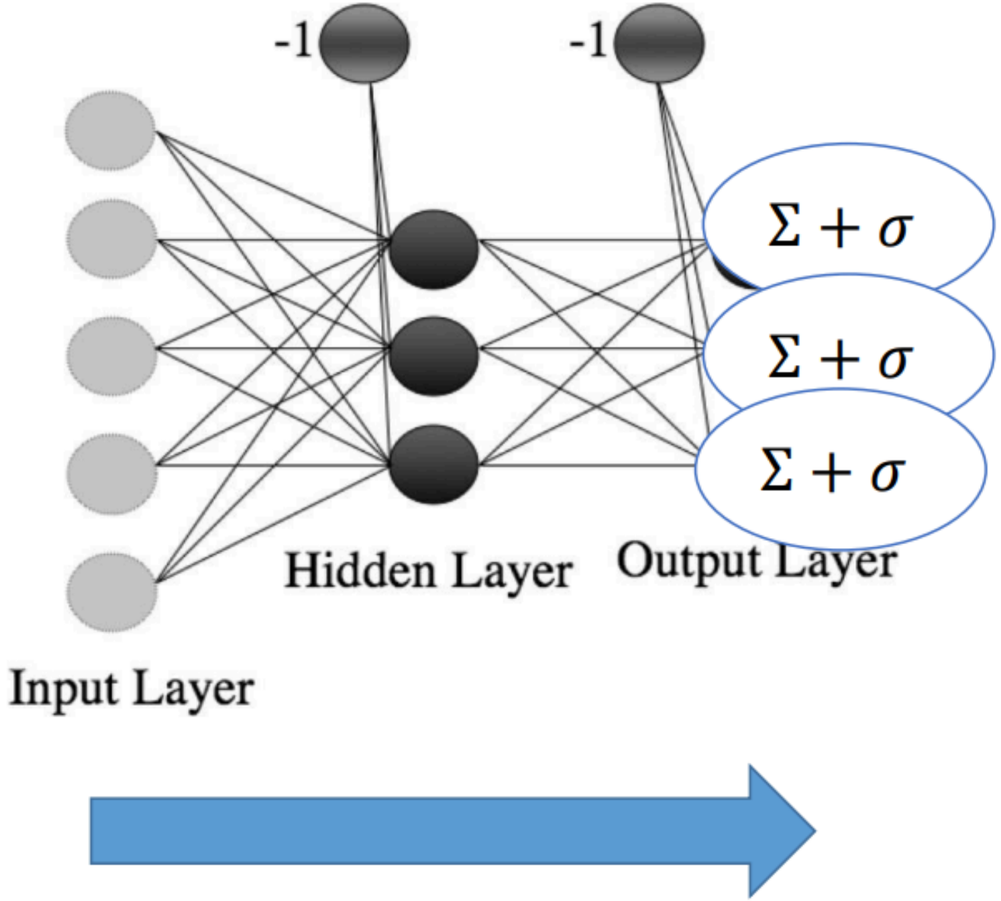
Binary classification

- The output layer contains a single neuron
- Logistic activation function in the output layer
- Similar to logistic regression



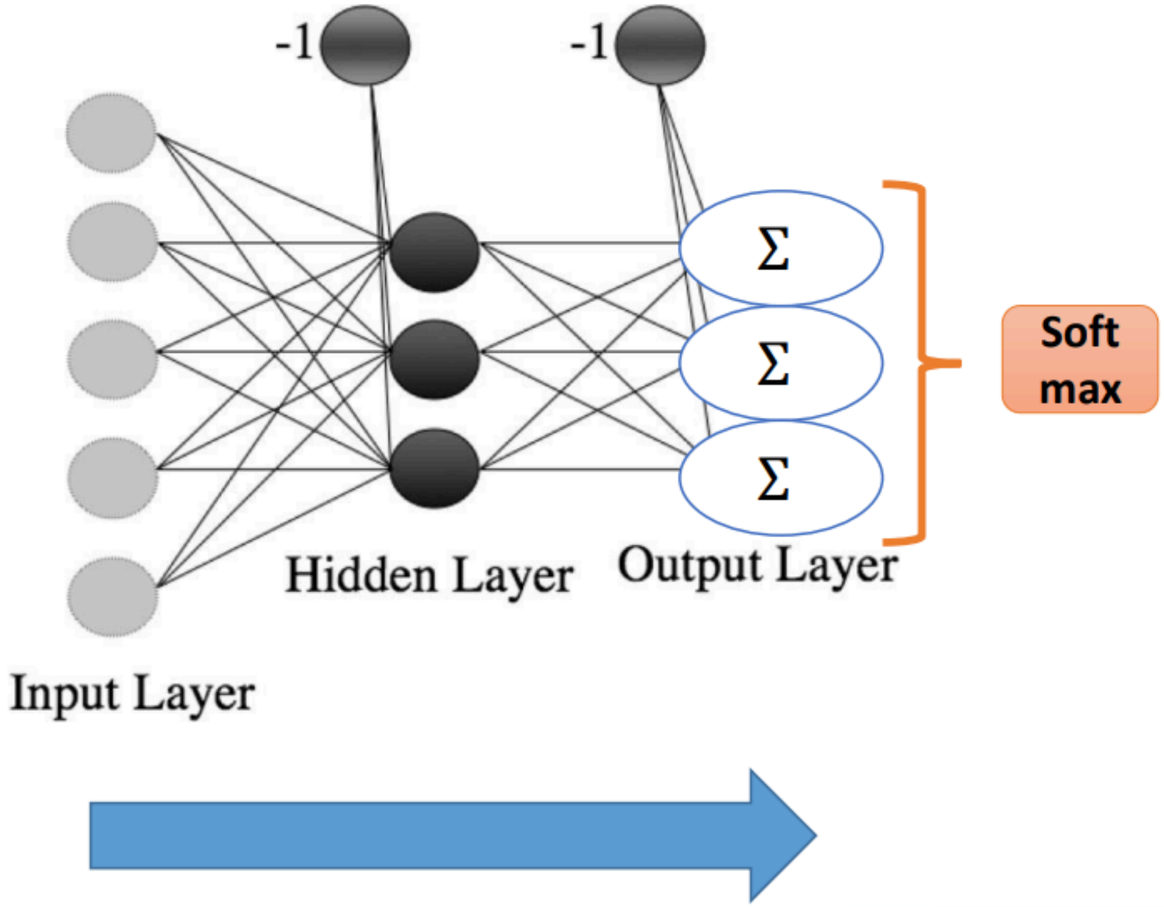
Multi label classification

- Several output nodes
- Logistic activation function
- Can be made multi-class classification by one vs. rest.



Multi class classification

- Several output nodes
- Sum the weighted inputs at each nodes
- The sums are brought together in the soft-max



Softmax function

The **softmax function** is a mathematical function that transforms a vector of raw scores (also called logits) into probabilities. It is commonly used in machine learning, especially in the output layer of a neural network for multi-class classification tasks.

$$softmax\left(x_j\right)=\frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$$

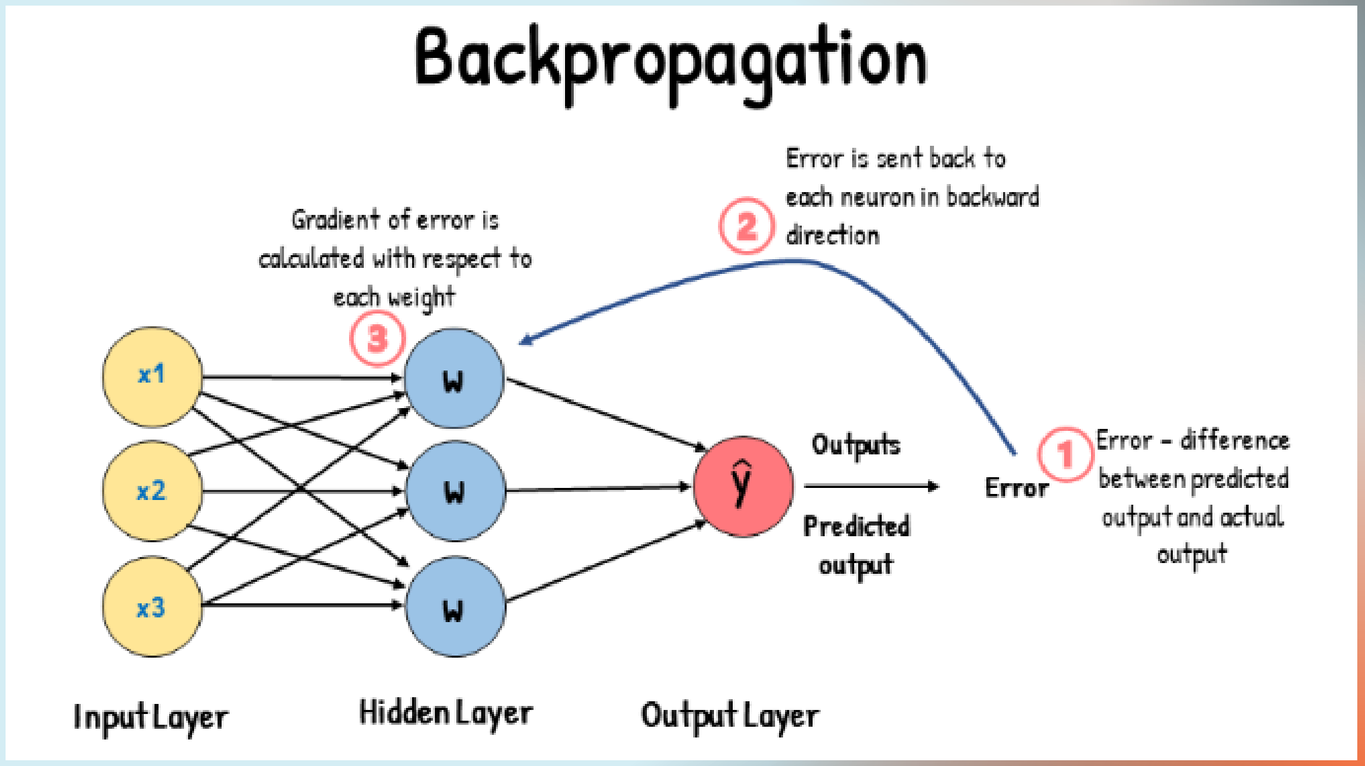
1. **Multi-Class Classification**: The softmax function is mainly used in the output layer of neural networks when the goal is to classify an input into one of several possible classes. It converts the raw output scores (logits) from the network into probabilities, allowing the model to make probabilistic predictions across multiple classes.
2. **Normalization**: The softmax function ensures that the output is normalized, meaning that all probabilities sum to 1. This is important for classification problems because it allows us to interpret the output of the network as a probability distribution.

Backpropagation

What is Backpropogation

Backpropagation is an algorithm that is designed to test for errors working back from output nodes to input nodes.

So in general refers to the whole process of calculating gradient of the
loss funcyion and its use in stochastic gradient descent in calculating the new weights



Backpropagation

Algorithm

Step 1 : Initializing of weights
Weights are initialized with random values

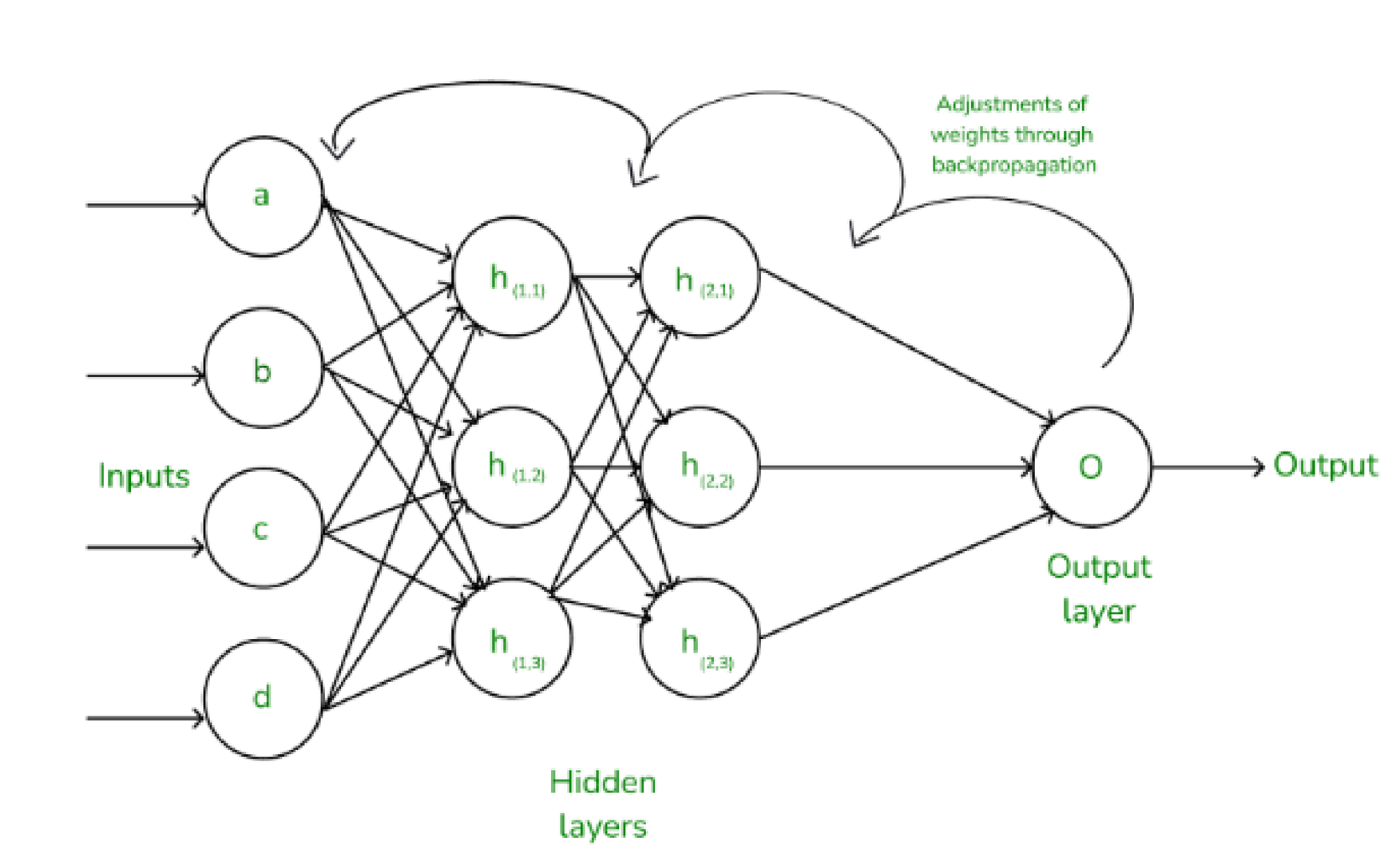
Step 2 : Feedforward
Each input unit in the input layer recieves input signals and transmits the signals to the hidden layers, which after applying the activation function sends the signal z to the output units in the output layer. The output layer then gives the output value y.

step 3: Backpropagation of errors
Each output unit compares the value y with its target value t, and an error is calculated. This is propagated back or distributed to all the units in the previous layers

Step 4: Updating of weights and biases

Backpropagation

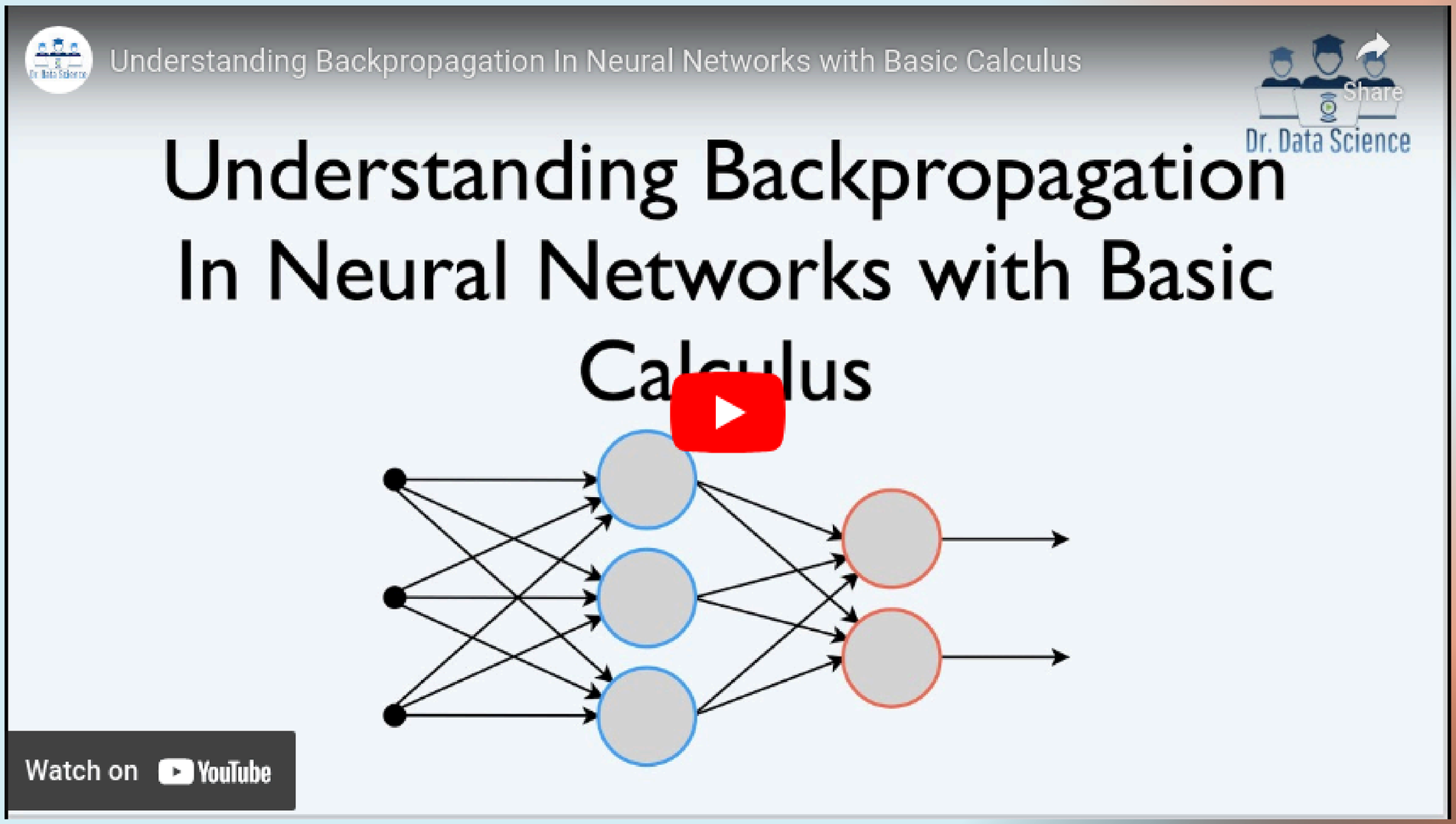
Example 1



https://www.geeksforgeeks.org/backpropagation-in-neural-network/

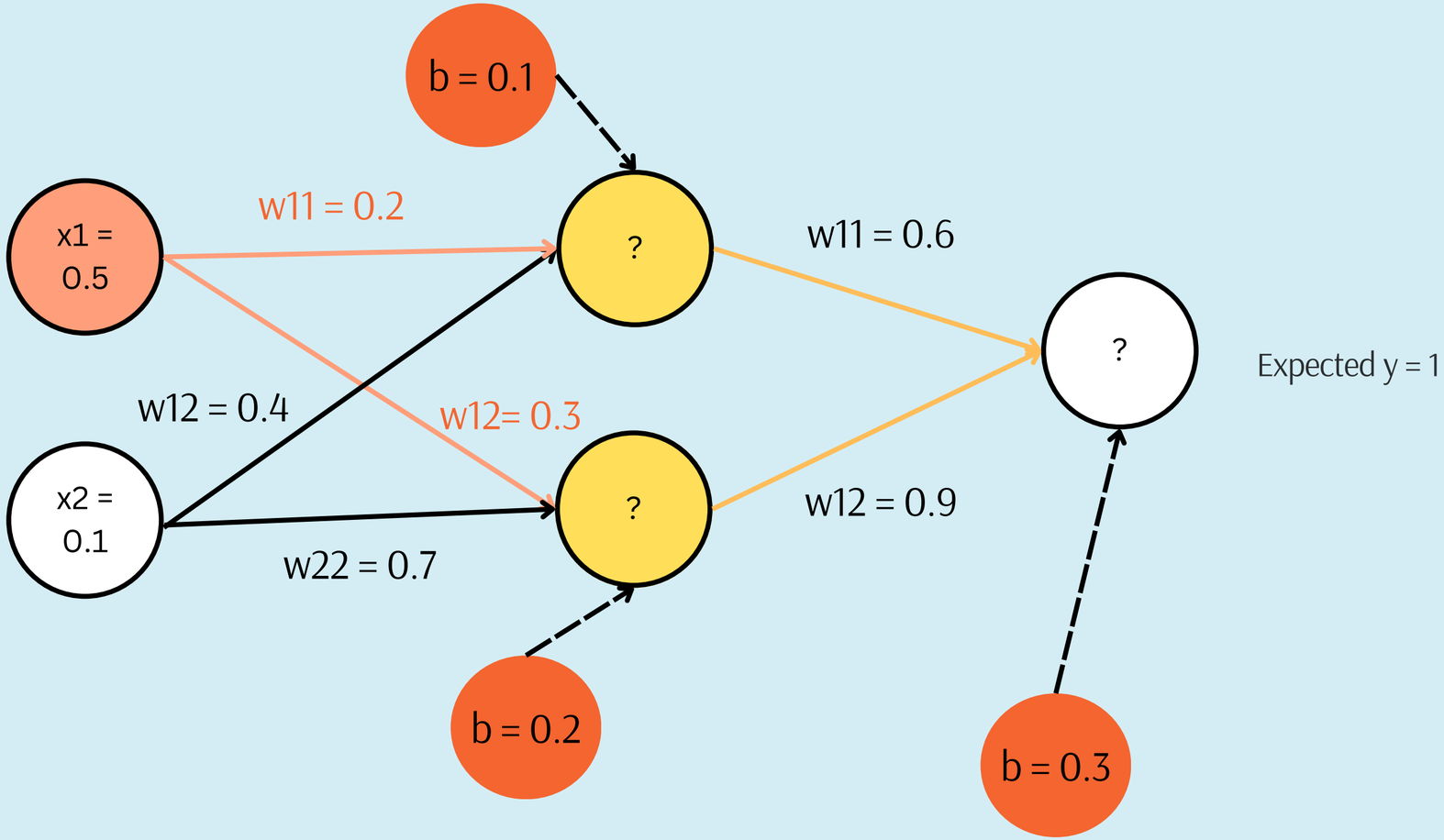
Backpropagation

Example 2



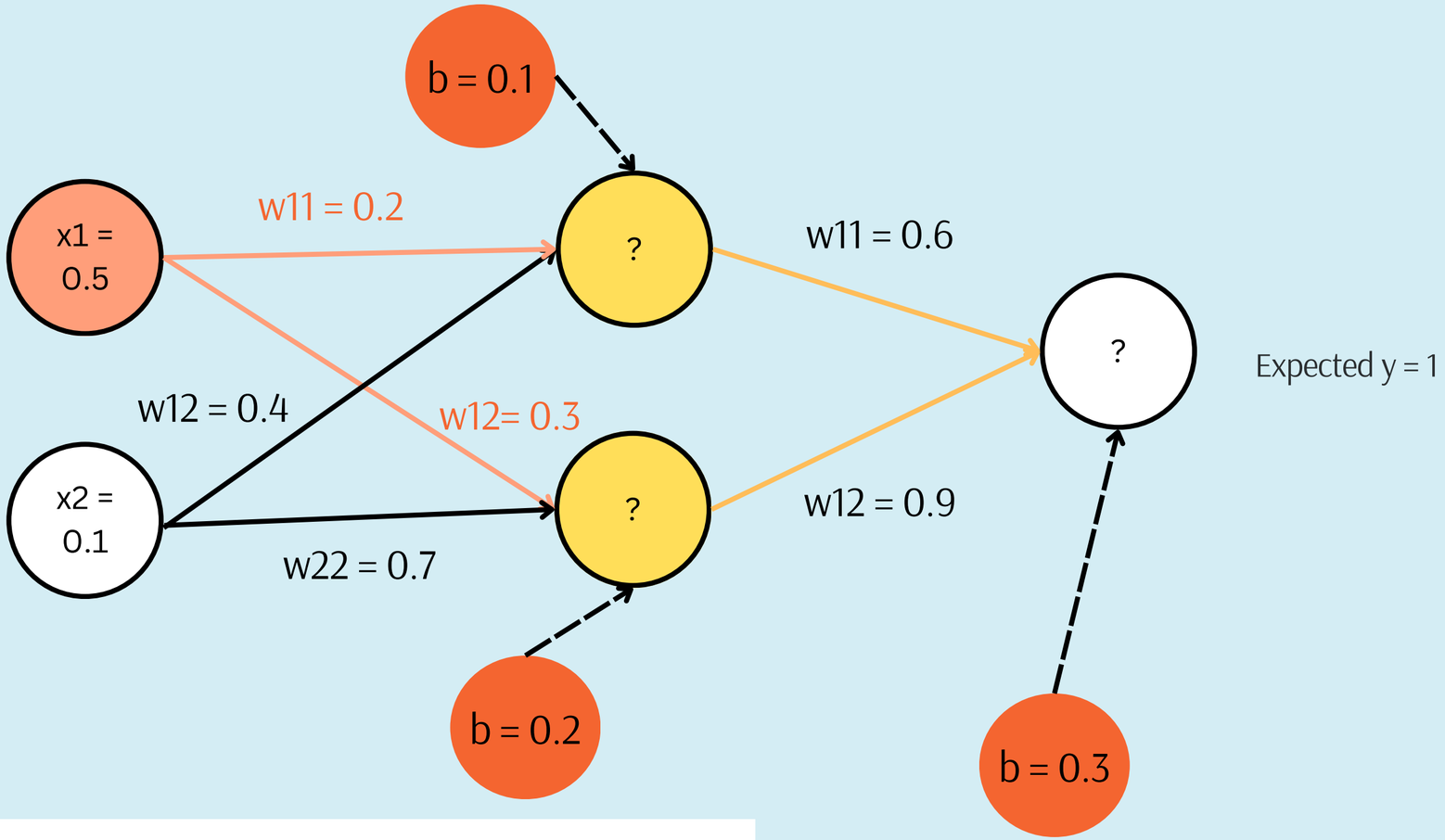
Backpropagation

Example 3



Backpropagation

Example 3



Step 1: Forward Pass

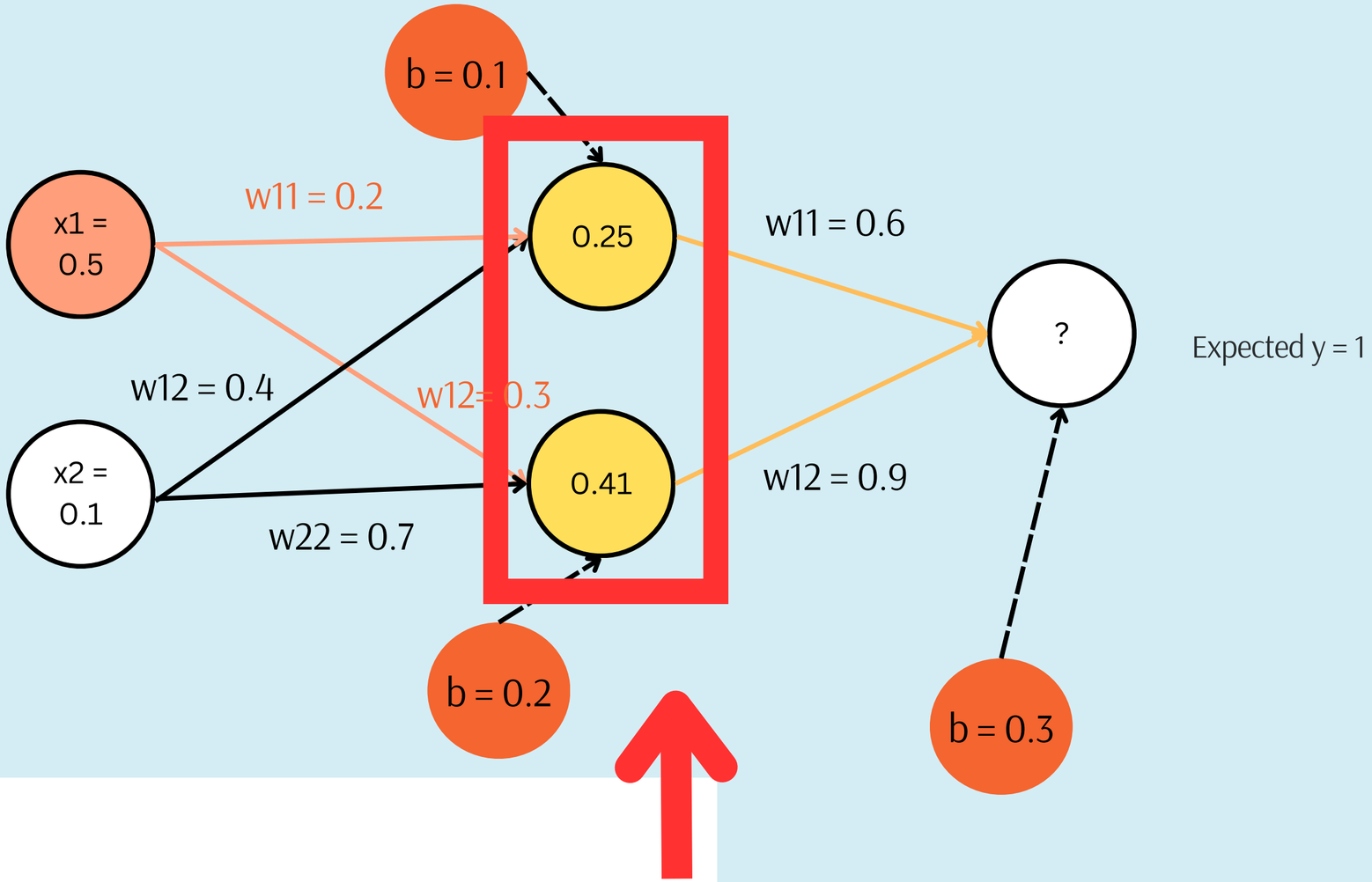
1. **Calculate the input to the hidden layer:**

$$z_1^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{12}^{(1)} \cdot x_2 + b_1^{(1)} = 0.2 \cdot 0.5 + 0.4 \cdot 0.1 + 0.1 = 0.25$$

$$z_2^{(1)} = w_{21}^{(1)} \cdot x_1 + w_{22}^{(1)} \cdot x_2 + b_2^{(1)} = 0.3 \cdot 0.5 + 0.7 \cdot 0.1 + 0.2 = 0.41$$

Backpropagation

Example 3



Step 1: Forward Pass

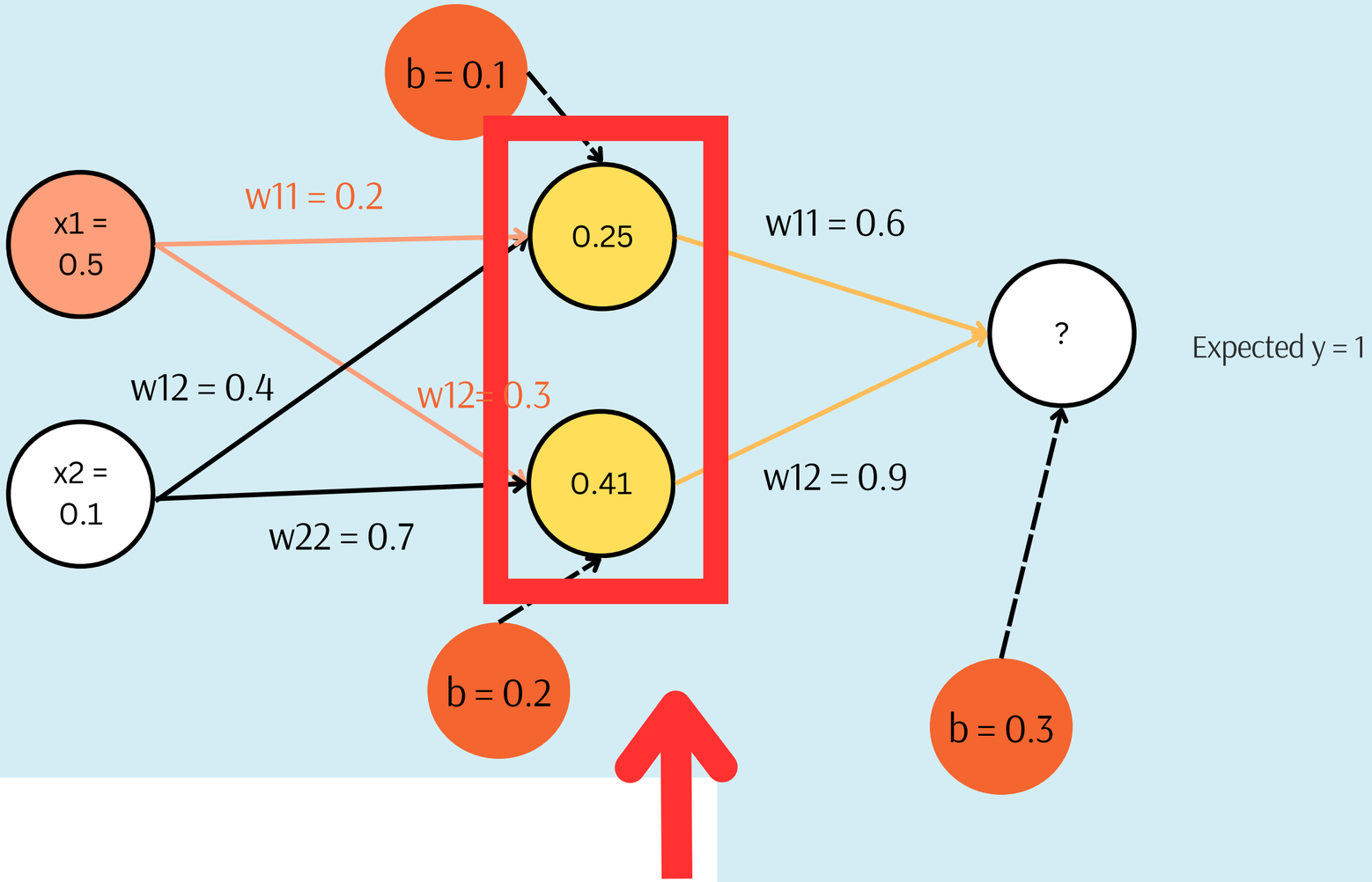
1. **Calculate the input to the hidden layer:**

$$z_1^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{12}^{(1)} \cdot x_2 + b_1^{(1)} = 0.2 \cdot 0.5 + 0.4 \cdot 0.1 + 0.1 = 0.25$$

$$z_2^{(1)} = w_{21}^{(1)} \cdot x_1 + w_{22}^{(1)} \cdot x_2 + b_2^{(1)} = 0.3 \cdot 0.5 + 0.7 \cdot 0.1 + 0.2 = 0.41$$

Backpropagation

Example 3



Step 1: Forward Pass

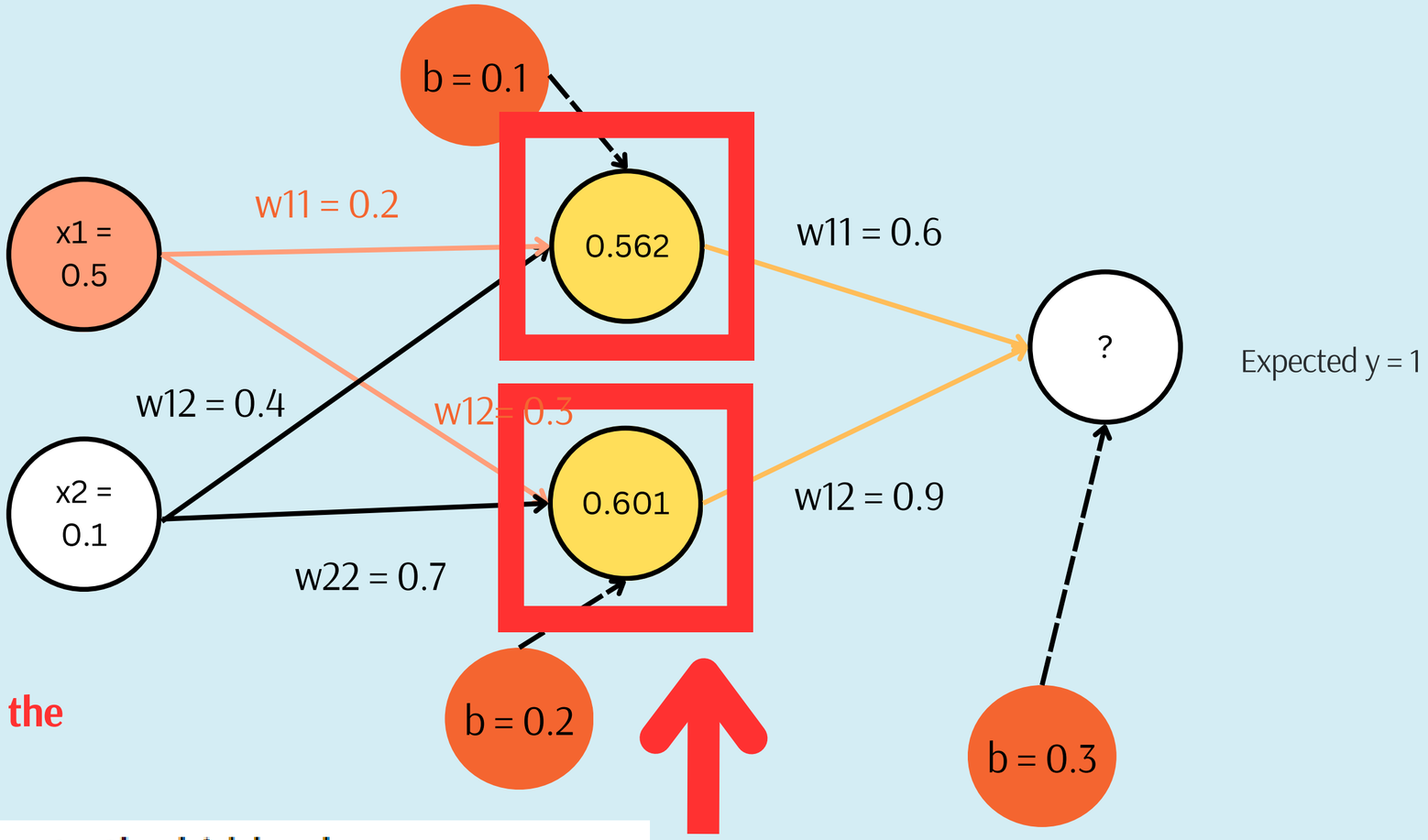
1. **Calculate the input to the hidden layer:**

$$z_1^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{12}^{(1)} \cdot x_2 + b_1^{(1)} = 0.2 \cdot 0.5 + 0.4 \cdot 0.1 + 0.1 = 0.25$$

$$z_2^{(1)} = w_{21}^{(1)} \cdot x_1 + w_{22}^{(1)} \cdot x_2 + b_2^{(1)} = 0.3 \cdot 0.5 + 0.7 \cdot 0.1 + 0.2 = 0.41$$

Backpropagation

Example 3



Since we had sigmoid activation function on the inner layers, we have to calculate it as well

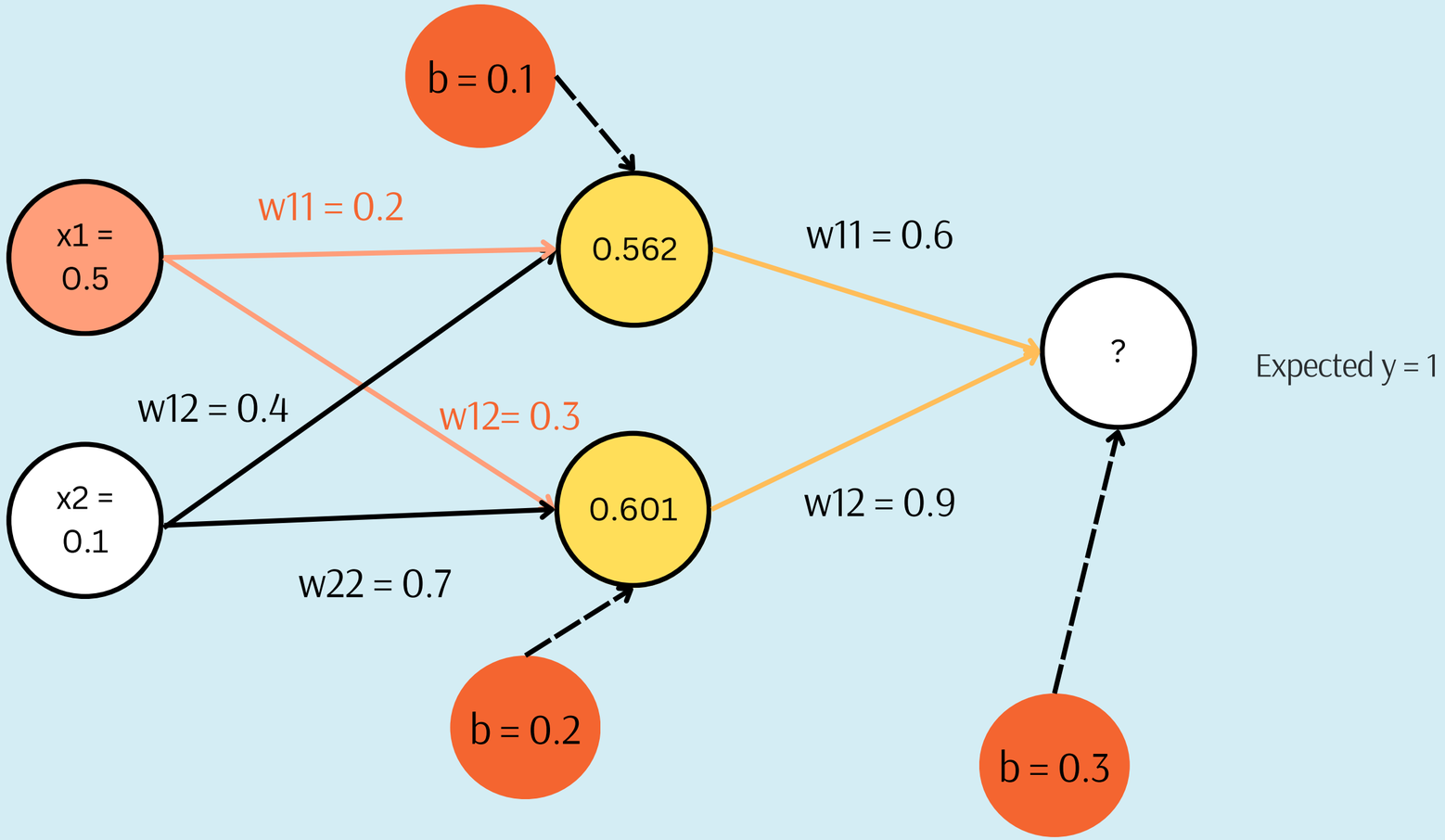
2. Apply the sigmoid activation function to the hidden layer:

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \frac{1}{1 + e^{-0.25}} \approx 0.562$$

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \frac{1}{1 + e^{-0.41}} \approx 0.601$$

Backpropagation

Example 3

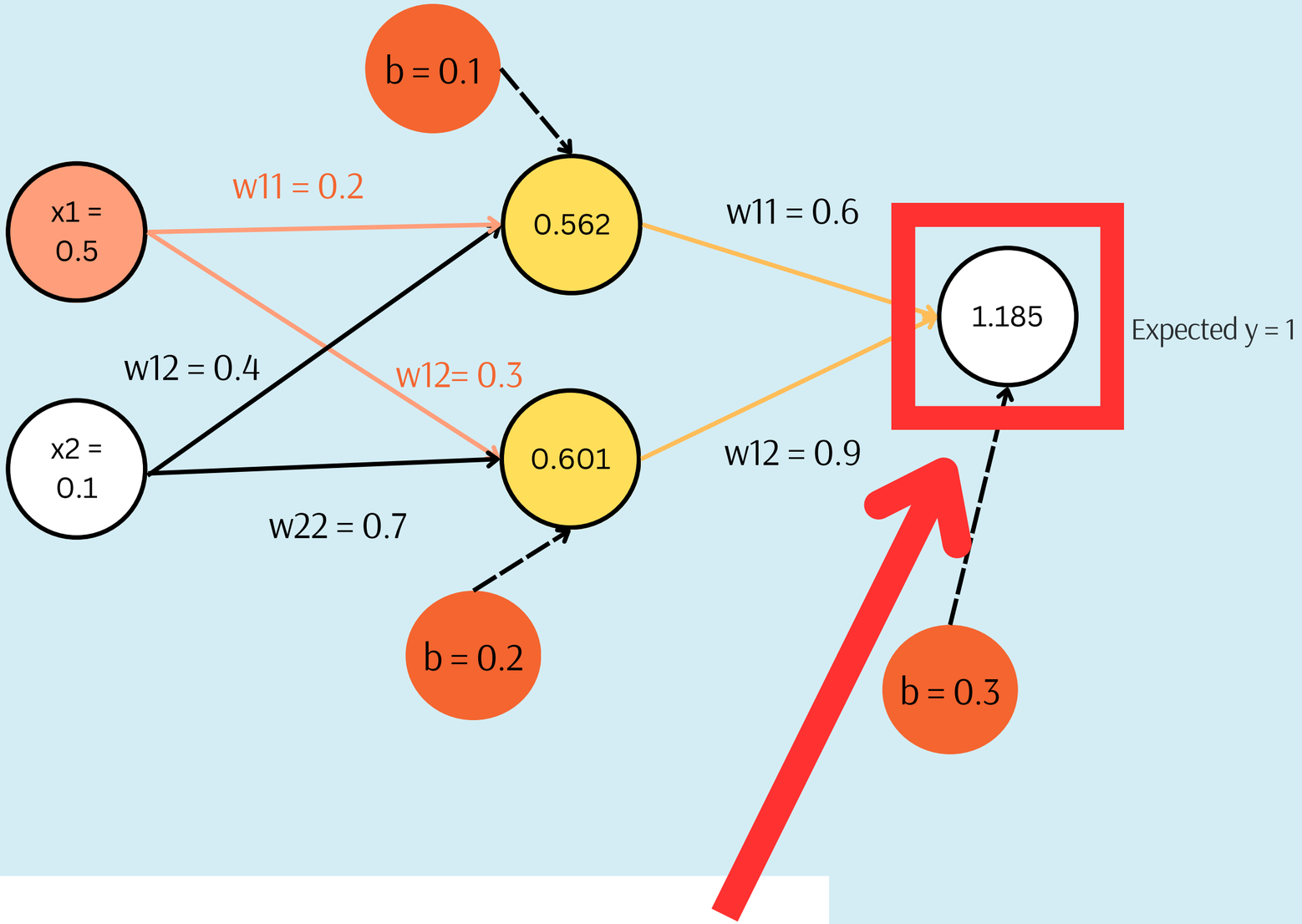


3. **Calculate the input to the output layer:**

$$z_1^{(2)} = w_{11}^{(2)} \cdot a_1^{(1)} + w_{12}^{(2)} \cdot a_2^{(1)} + b_1^{(2)} = 0.6 \cdot 0.562 + 0.9 \cdot 0.601 + 0.3 \approx 1.185$$

Backpropagation

Example 3

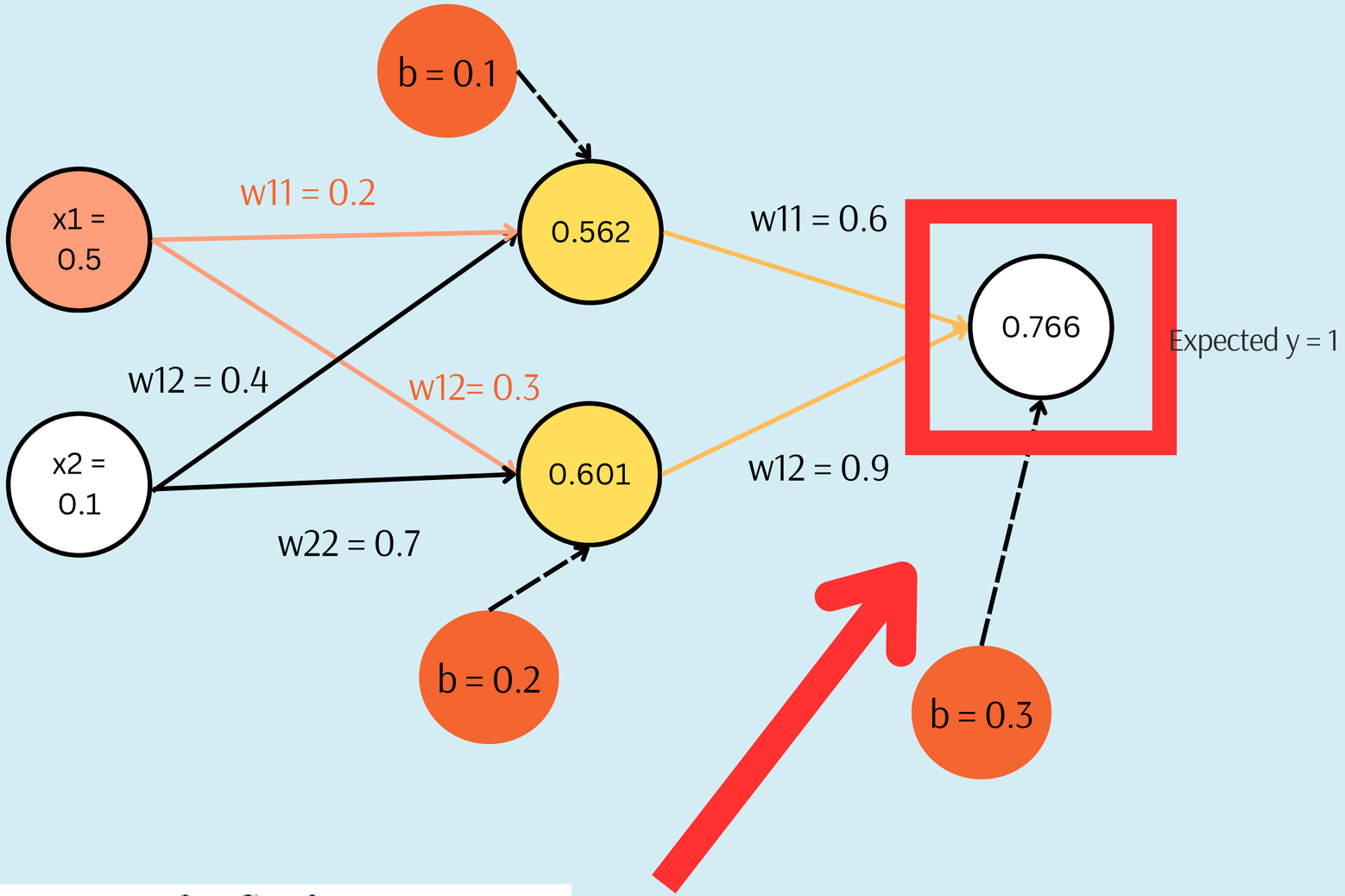


3. **Calculate the input to the output layer:**

$$z_1^{(2)} = w_{11}^{(2)} \cdot a_1^{(1)} + w_{12}^{(2)} \cdot a_2^{(1)} + b_1^{(2)} = 0.6 \cdot 0.562 + 0.9 \cdot 0.601 + 0.3 \approx 1.185$$

Backpropagation

Example 3

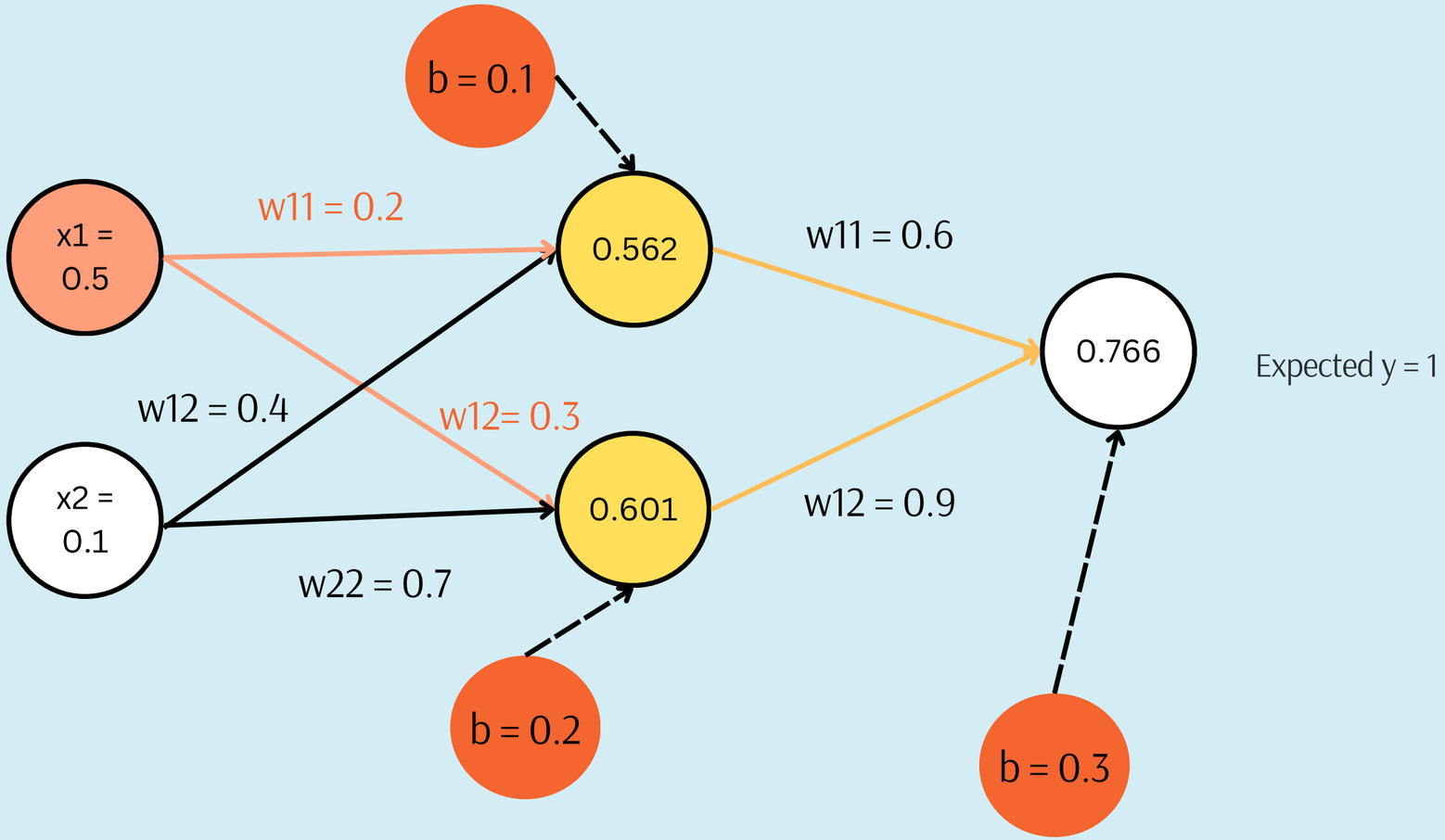


4. Apply the sigmoid activation function to get the final output:

$$\hat{y} = \sigma(z_1^{(2)}) = \frac{1}{1 + e^{-1.185}} \approx 0.766$$

Backpropagation

Example 3

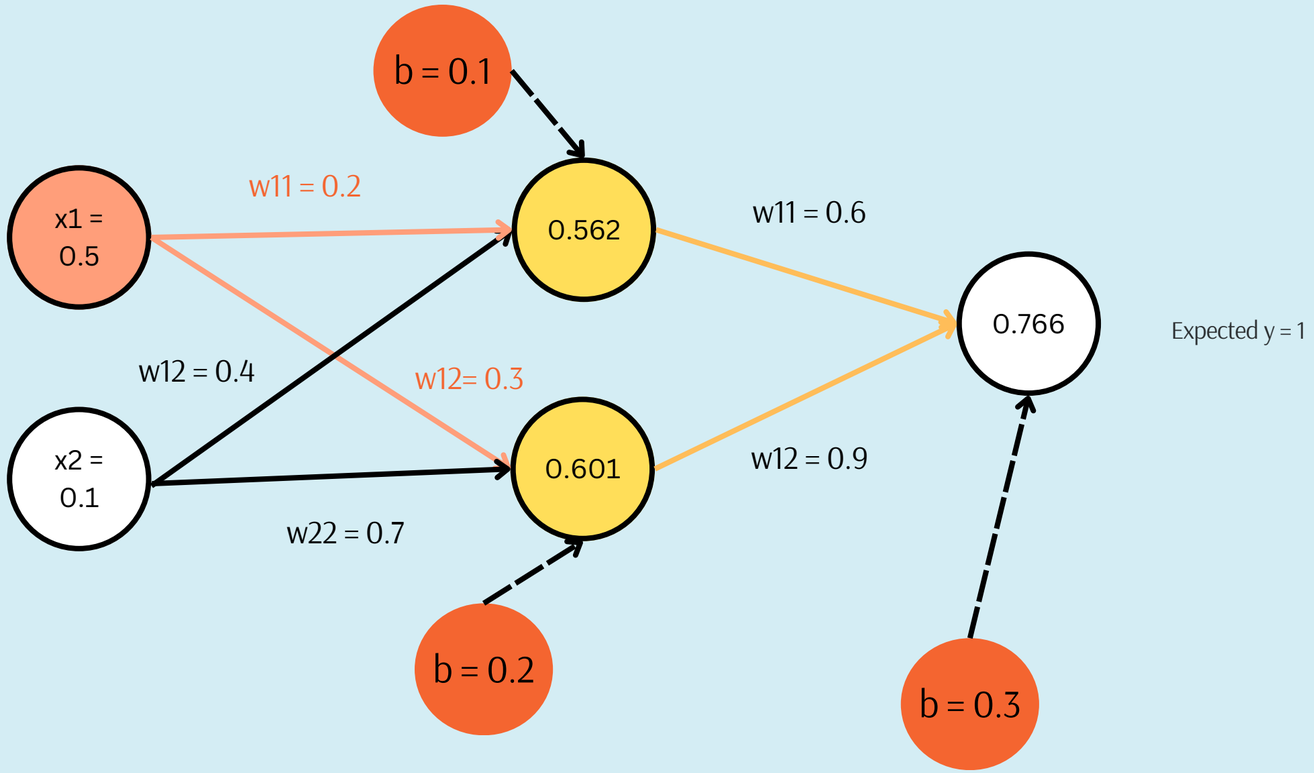


Output: 0.766

Expected y = 1

Backpropagation

Example 3



Step 2: Calculate the Loss (Binary Cross-Entropy Loss)

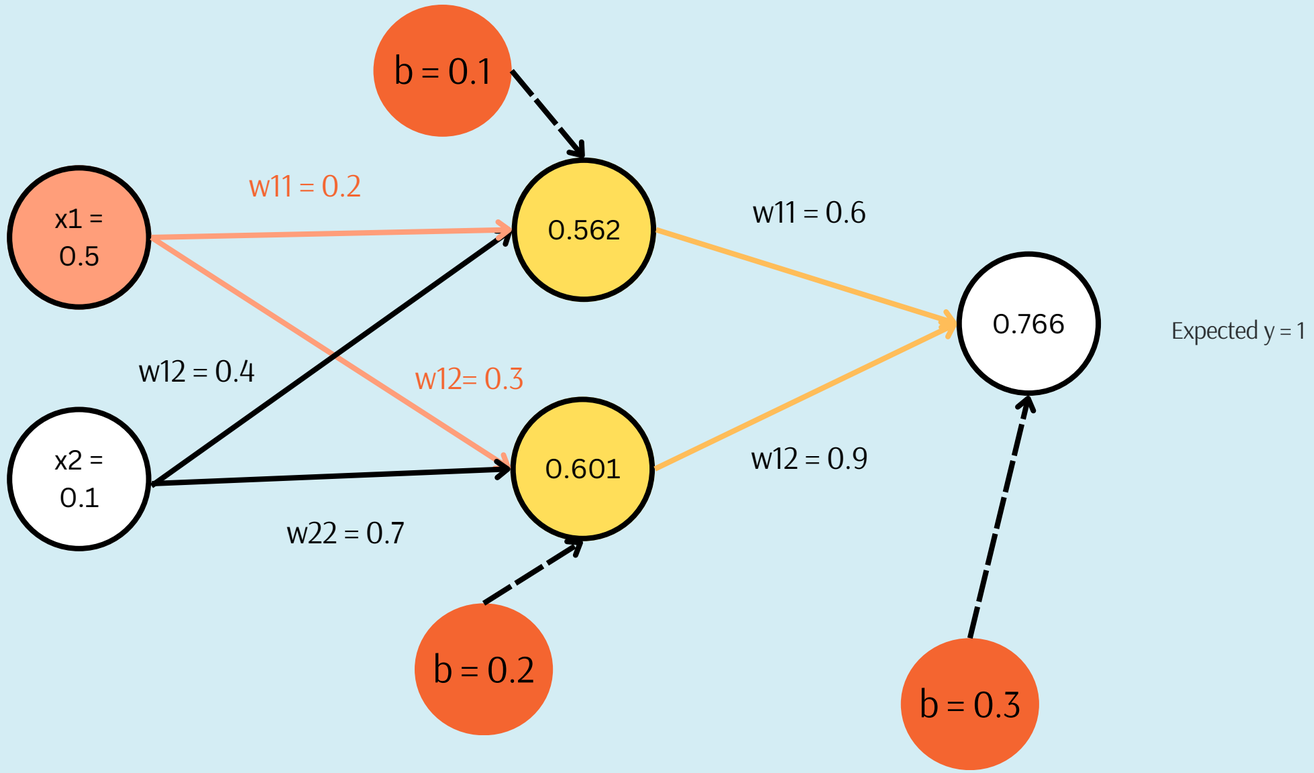
$$L = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})$$

For $y = 1$ and $\hat{y} = 0.766$, the loss becomes:

$$L = -1 \cdot \log(0.766) - (1 - 1) \cdot \log(1 - 0.766) = -\log(0.766) \approx 0.267$$

Backpropagation

Example 3



Step 3: Backward Pass

Now, we will calculate the gradients for backpropagation using the cross-entropy loss.

Backpropagation

Example 3

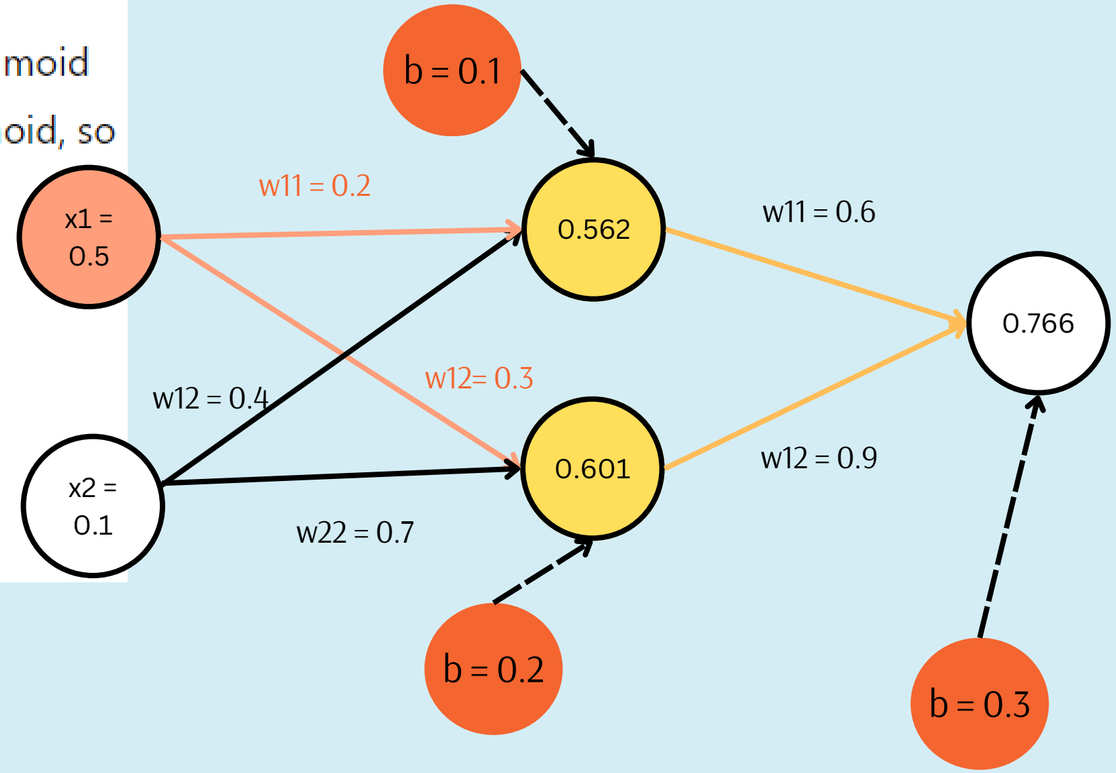
1. **Compute the gradient at the output layer:**
   - **Derivative of the loss with respect to the output:** For cross-entropy loss with sigmoid, we can directly compute the error as:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y = 0.766 - 1 = -0.234$$

   - **Derivative of the sigmoid activation function:** The derivative of the sigmoid activation is already accounted for when using cross-entropy with sigmoid, so we don't need to multiply by the derivative of the activation function explicitly.

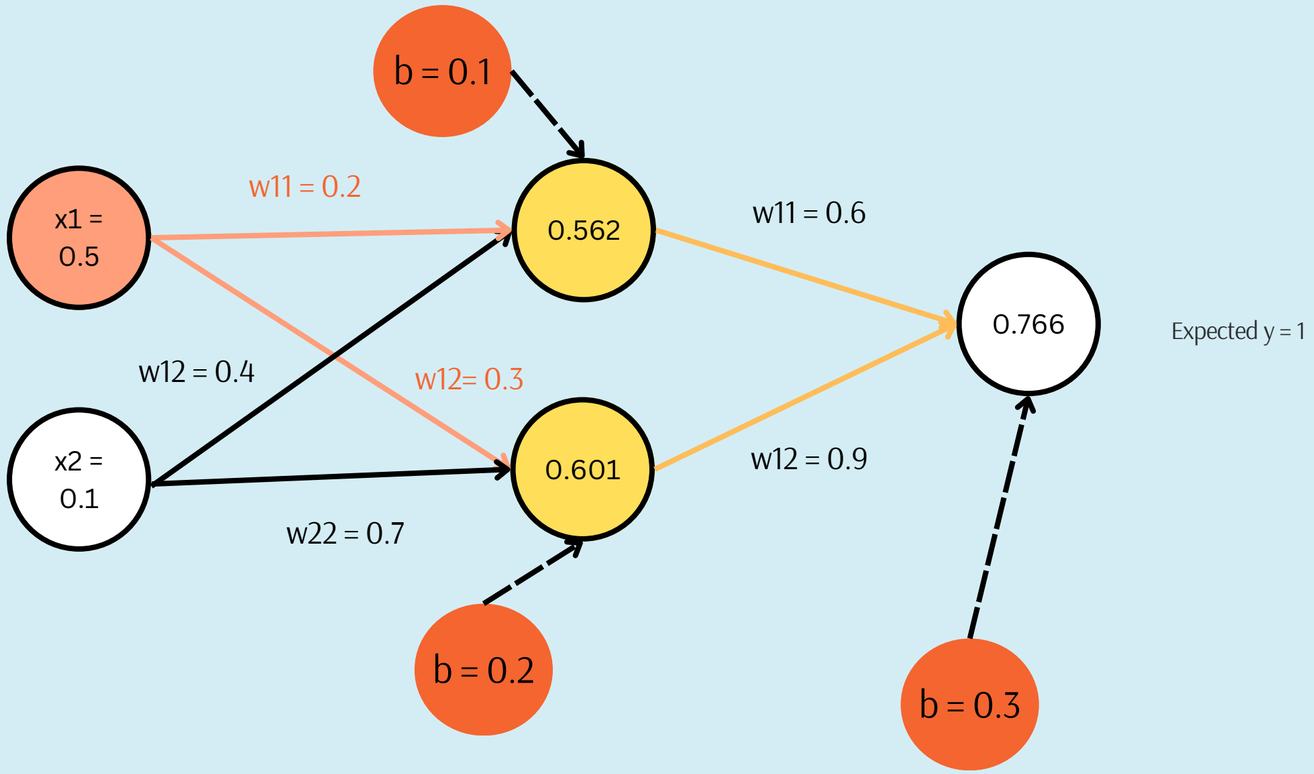
Therefore, the **gradient at the output layer** is simply:

$$\delta^{(2)} = \frac{\partial L}{\partial z_1^{(2)}} = \hat{y} - y = -0.234$$



Backpropagation

Example 3



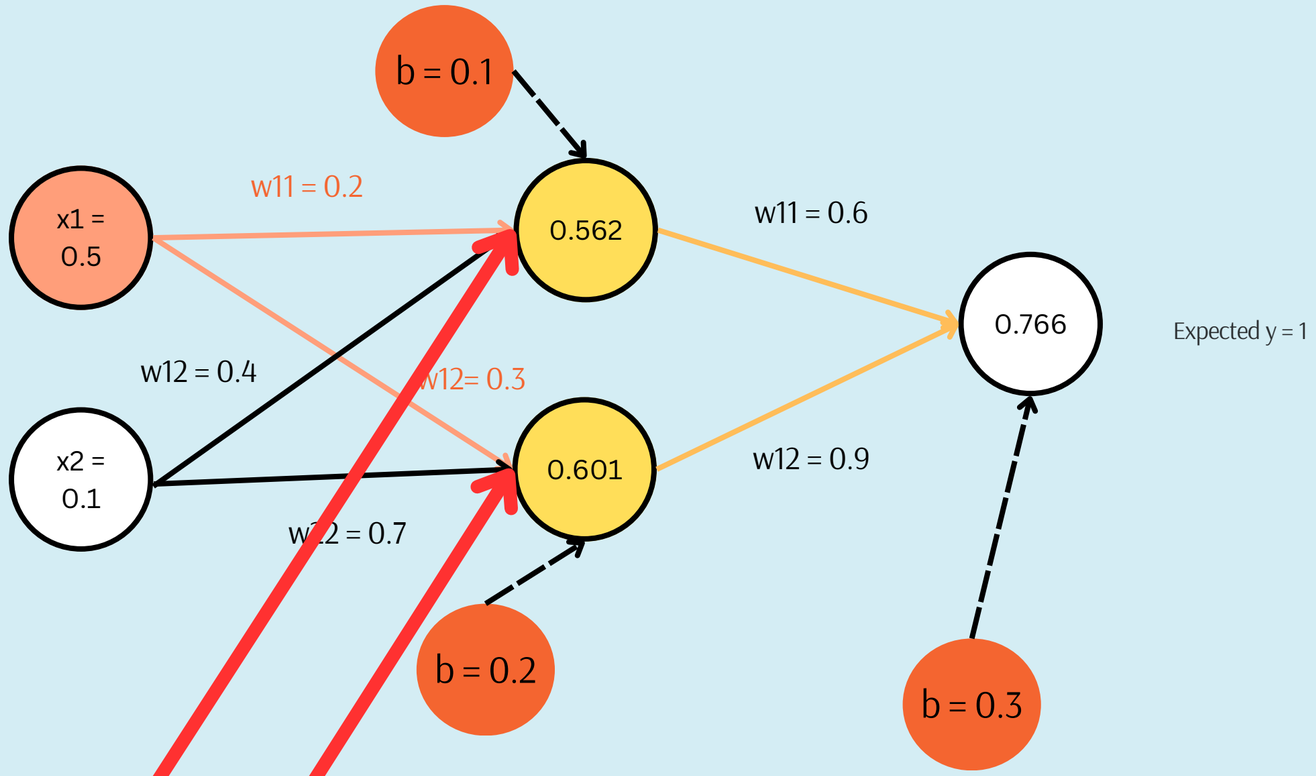
2. **Compute the gradients with respect to the weights between the hidden layer and the output layer:**

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \delta^{(2)} \cdot a_1^{(1)} = -0.234 \cdot 0.562 \approx -0.131$$

$$\frac{\partial L}{\partial w_{12}^{(2)}} = \delta^{(2)} \cdot a_2^{(1)} = -0.234 \cdot 0.601 \approx -0.141$$

Backpropagation

Example 3



Gradient of the output layer
(Calculated from previous slide)

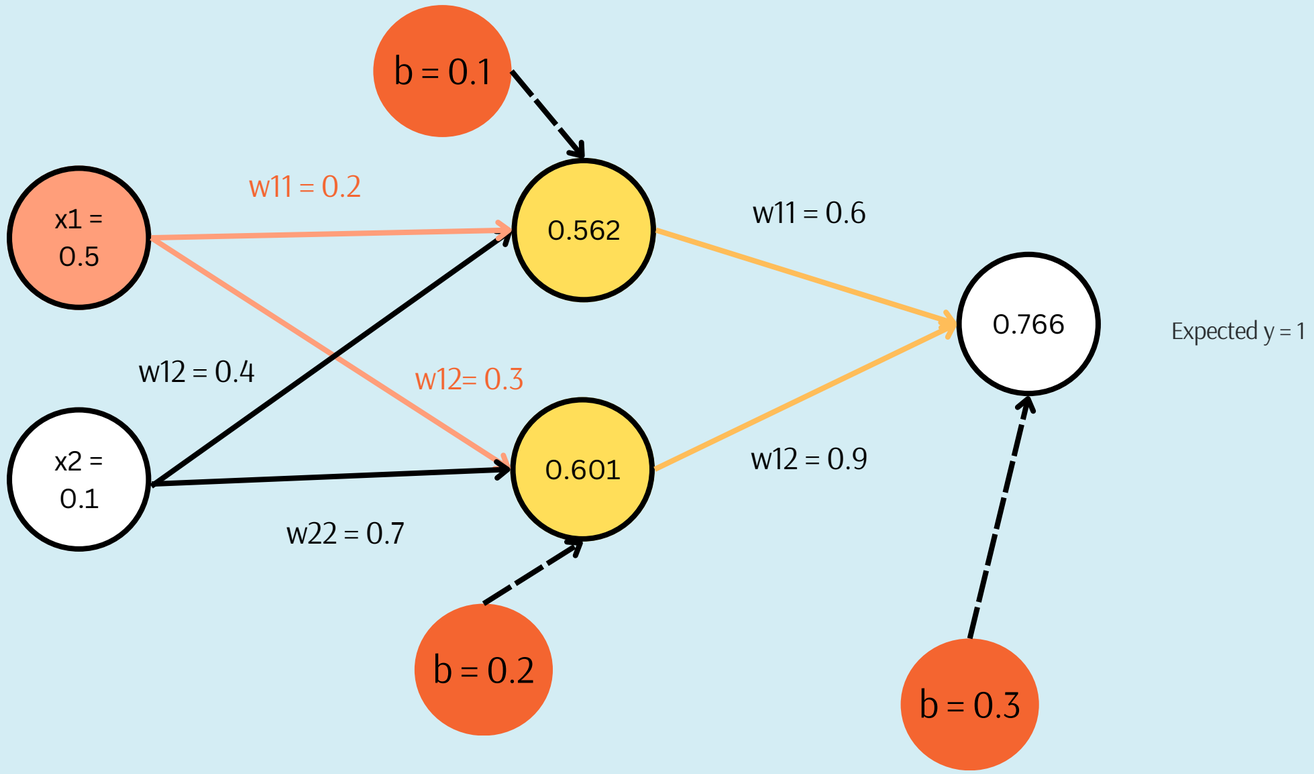
2. Compute the gradients with respect to the weights between the hidden layer and the output layer:

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \delta^{(2)} \cdot a_1^{(1)} = -0.234 \cdot 0.562 \approx -0.131$$

$$\frac{\partial L}{\partial w_{12}^{(2)}} = \delta^{(2)} \cdot a_2^{(1)} = -0.234 \cdot 0.601 \approx -0.141$$

Backpropagation

Example 3

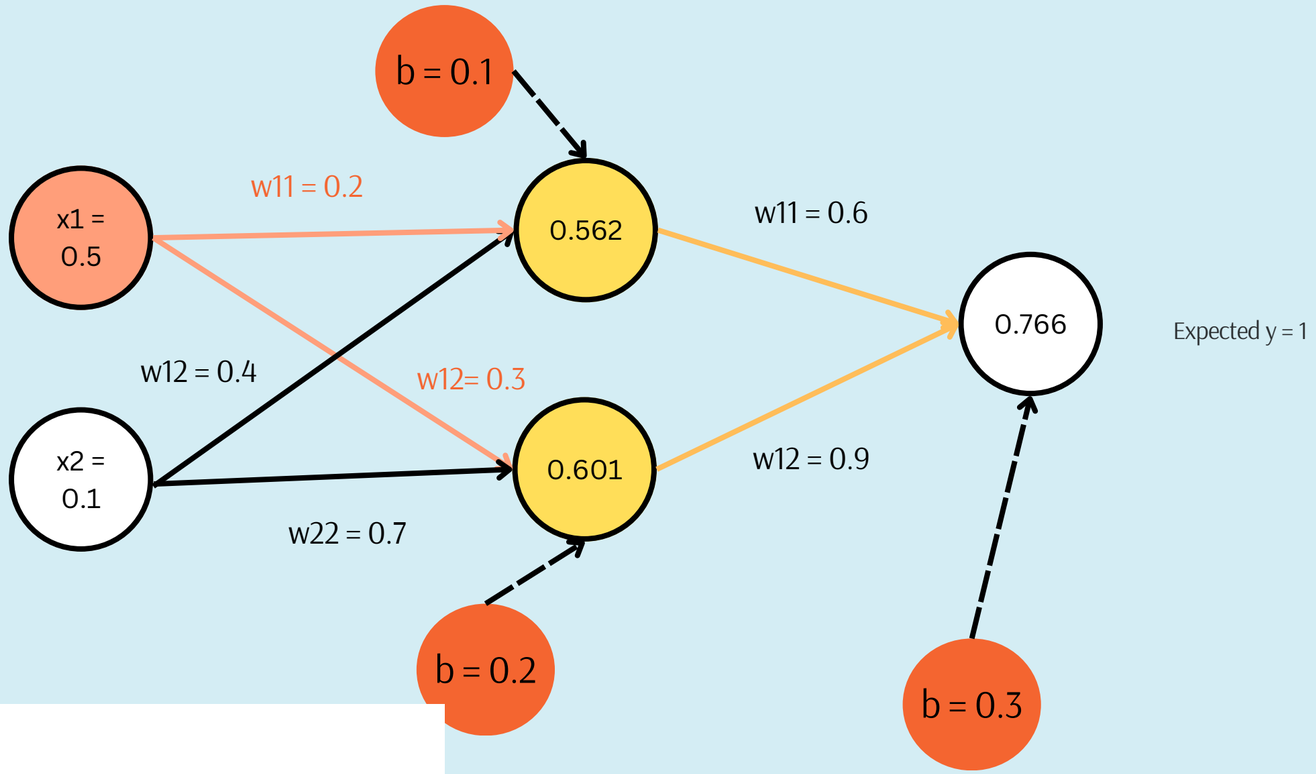


3. **Compute the gradient at the hidden layer:**

   For each neuron in the hidden layer, calculate the gradient by propagating the error backward from the output layer:

Backpropagation

Example 3



- For the first hidden layer neuron:

$$\sigma'(z_1^{(1)}) = a_1^{(1)} \cdot (1 - a_1^{(1)}) = 0.562 \cdot (1 - 0.562) \approx 0.247$$

$$\delta_1^{(1)} = \delta^{(2)} \cdot w_{11}^{(2)} \cdot \sigma'(z_1^{(1)}) = -0.234 \cdot 0.6 \cdot 0.247 \approx -0.0346$$

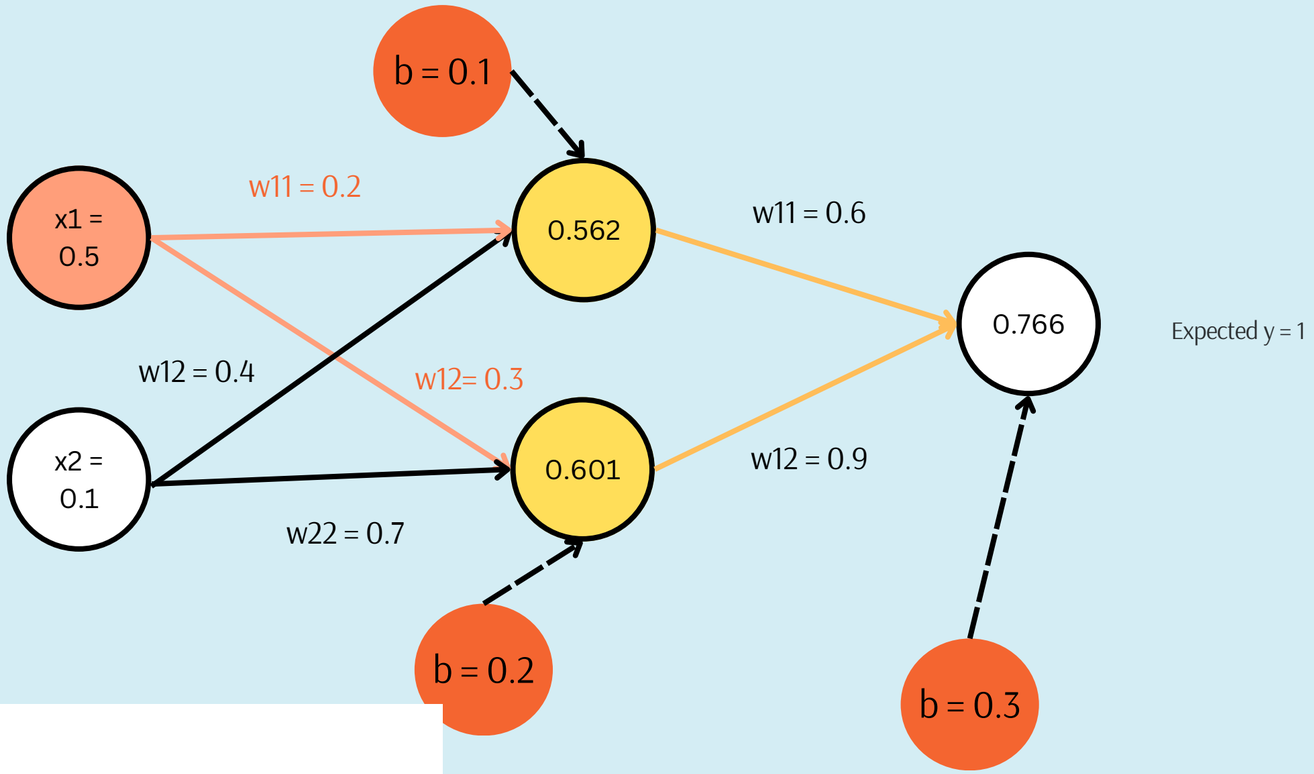
- For the second hidden layer neuron:

$$\sigma'(z_2^{(1)}) = a_2^{(1)} \cdot (1 - a_2^{(1)}) = 0.601 \cdot (1 - 0.601) \approx 0.240$$

$$\delta_2^{(1)} = \delta^{(2)} \cdot w_{12}^{(2)} \cdot \sigma'(z_2^{(1)}) = -0.234 \cdot 0.9 \cdot 0.240 \approx -0.0505$$

Backpropagation

Example 3



backpropagation for the hidden layer

derivative of thee activation function(sigmoid derivative)

- For the first hidden layer neuron:

$$\sigma'(z_1^{(1)}) = a_1^{(1)} \cdot (1 - a_1^{(1)}) = 0.562 \cdot (1 - 0.562) \approx 0.247$$

$$\delta_1^{(1)} = \delta^{(2)} \cdot w_{11}^{(2)} \cdot \sigma'(z_1^{(1)}) = -0.234 \cdot 0.6 \cdot 0.247 \approx -0.0346$$

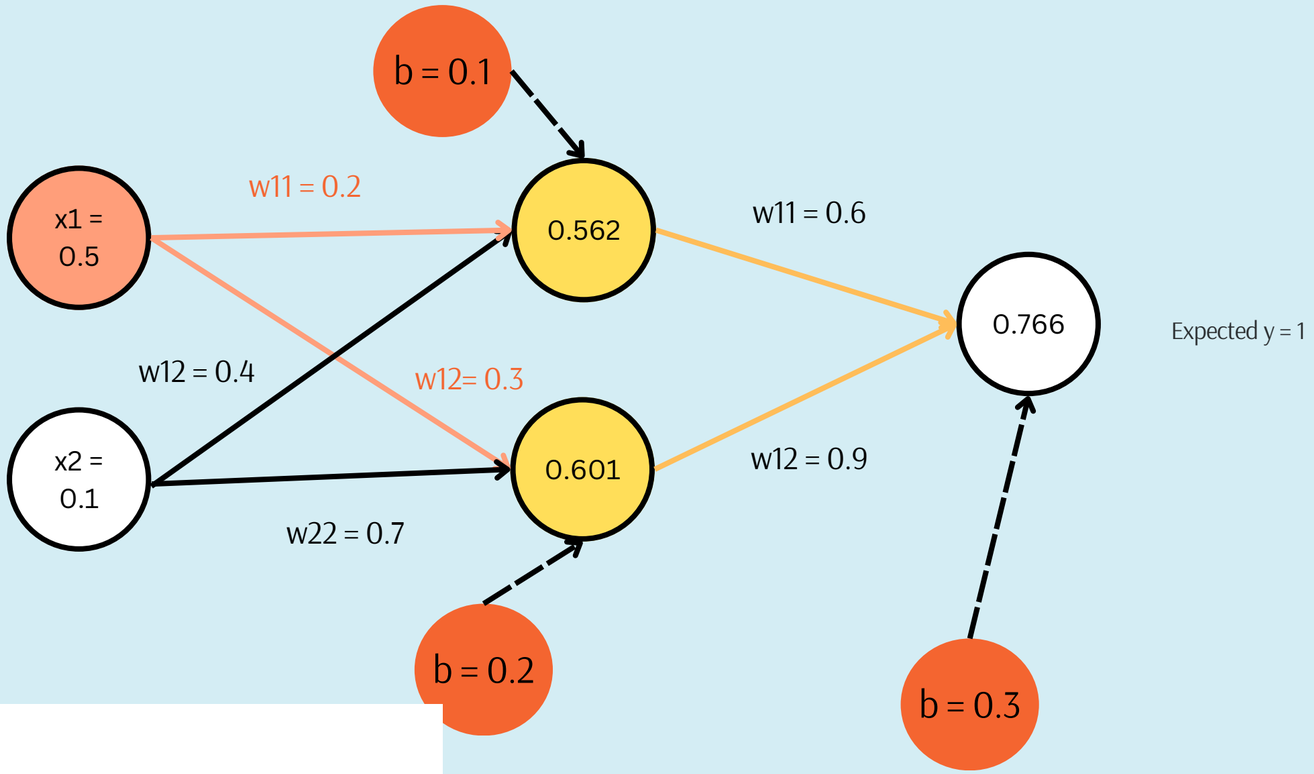
- For the second hidden layer neuron:

$$\sigma'(z_2^{(1)}) = a_2^{(1)} \cdot (1 - a_2^{(1)}) = 0.601 \cdot (1 - 0.601) \approx 0.240$$

$$\delta_2^{(1)} = \delta^{(2)} \cdot w_{12}^{(2)} \cdot \sigma'(z_2^{(1)}) = -0.234 \cdot 0.9 \cdot 0.240 \approx -0.0505$$

Backpropagation

Example 3



w between hidden and output layers

error from output layer

derivative of activation function

- For the first hidden layer neuron:

$$\sigma'(z_1^{(1)}) = a_1^{(1)} \cdot (1 - a_1^{(1)}) = 0.562 \cdot (1 - 0.562) \approx 0.247$$

$$\delta_1^{(1)} = \delta^{(2)} \cdot w_{11}^{(2)} \cdot \sigma'(z_1^{(1)}) = -0.234 \cdot 0.6 \cdot 0.247 \approx -0.0346$$

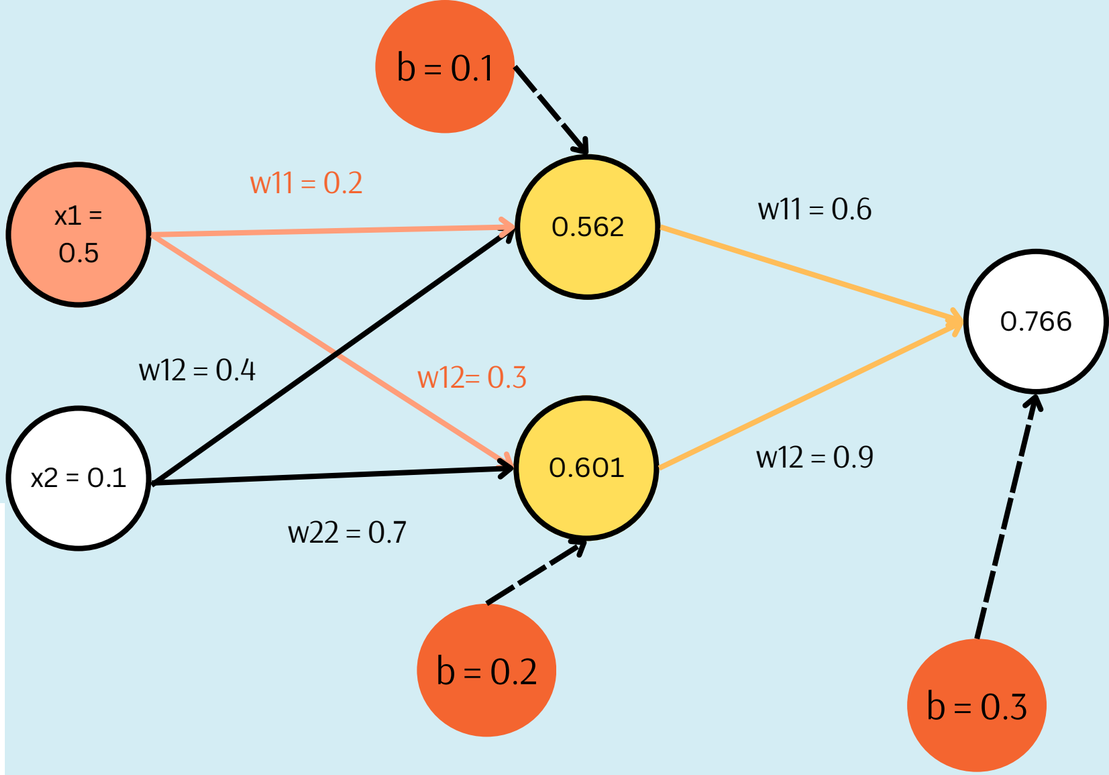
- For the second hidden layer neuron:

$$\sigma'(z_2^{(1)}) = a_2^{(1)} \cdot (1 - a_2^{(1)}) = 0.601 \cdot (1 - 0.601) \approx 0.240$$

$$\delta_2^{(1)} = \delta^{(2)} \cdot w_{12}^{(2)} \cdot \sigma'(z_2^{(1)}) = -0.234 \cdot 0.9 \cdot 0.240 \approx -0.0505$$

Backpropagation

Example 3



4. **Compute the gradients with respect to the weights between the input layer and the hidden layer:**

   - For the first neuron in the hidden layer:

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \delta_1^{(1)} \cdot x_1 = -0.0346 \cdot 0.5 \approx -0.0173$$

$$\frac{\partial L}{\partial w_{12}^{(1)}} = \delta_1^{(1)} \cdot x_2 = -0.0346 \cdot 0.1 \approx -0.00346$$

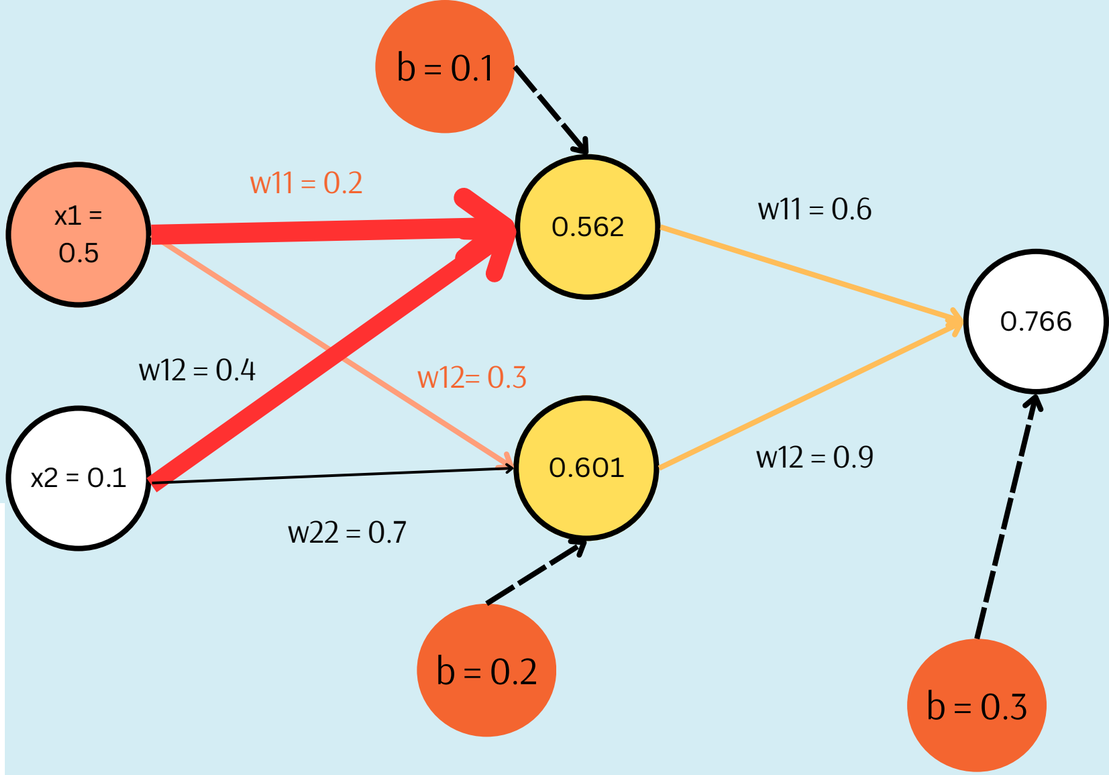
   - For the second neuron in the hidden layer:

$$\frac{\partial L}{\partial w_{21}^{(1)}} = \delta_2^{(1)} \cdot x_1 = -0.0505 \cdot 0.5 \approx -0.0253$$

$$\frac{\partial L}{\partial w_{22}^{(1)}} = \delta_2^{(1)} \cdot x_2 = -0.0505 \cdot 0.1 \approx -0.00505$$

Backpropagation

Example 3



4. **Compute the gradients with respect to the weights between the input layer and the hidden layer:**

- For the first neuron in the hidden layer:

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \delta_1^{(1)} \cdot x_1 = -0.0346 \cdot 0.5 \approx -0.0173$$

$$\frac{\partial L}{\partial w_{12}^{(1)}} = \delta_1^{(1)} \cdot x_2 = -0.0346 \cdot 0.1 \approx -0.00346$$

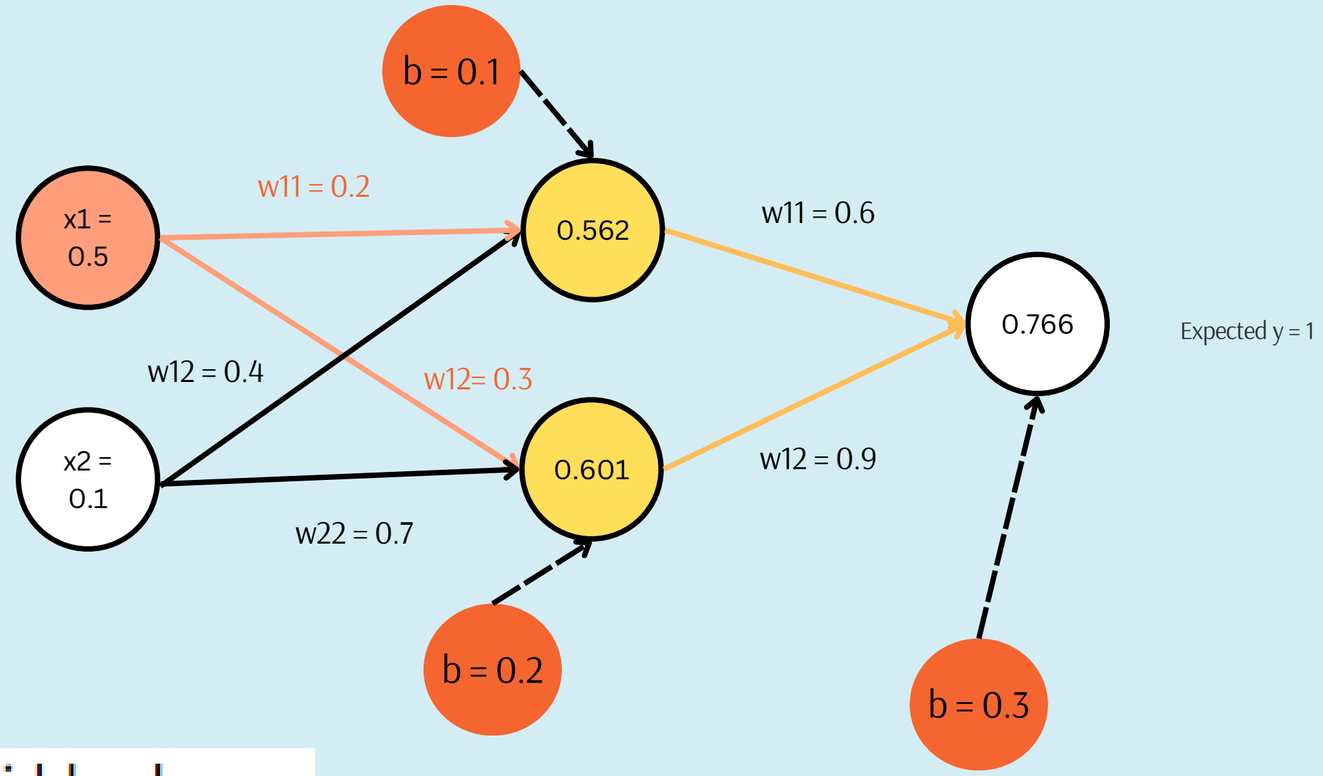
- For the second neuron in the hidden layer:

$$\frac{\partial L}{\partial w_{21}^{(1)}} = \delta_2^{(1)} \cdot x_1 = -0.0505 \cdot 0.5 \approx -0.0253$$

$$\frac{\partial L}{\partial w_{22}^{(1)}} = \delta_2^{(1)} \cdot x_2 = -0.0505 \cdot 0.1 \approx -0.00505$$

Backpropagation

Example 3



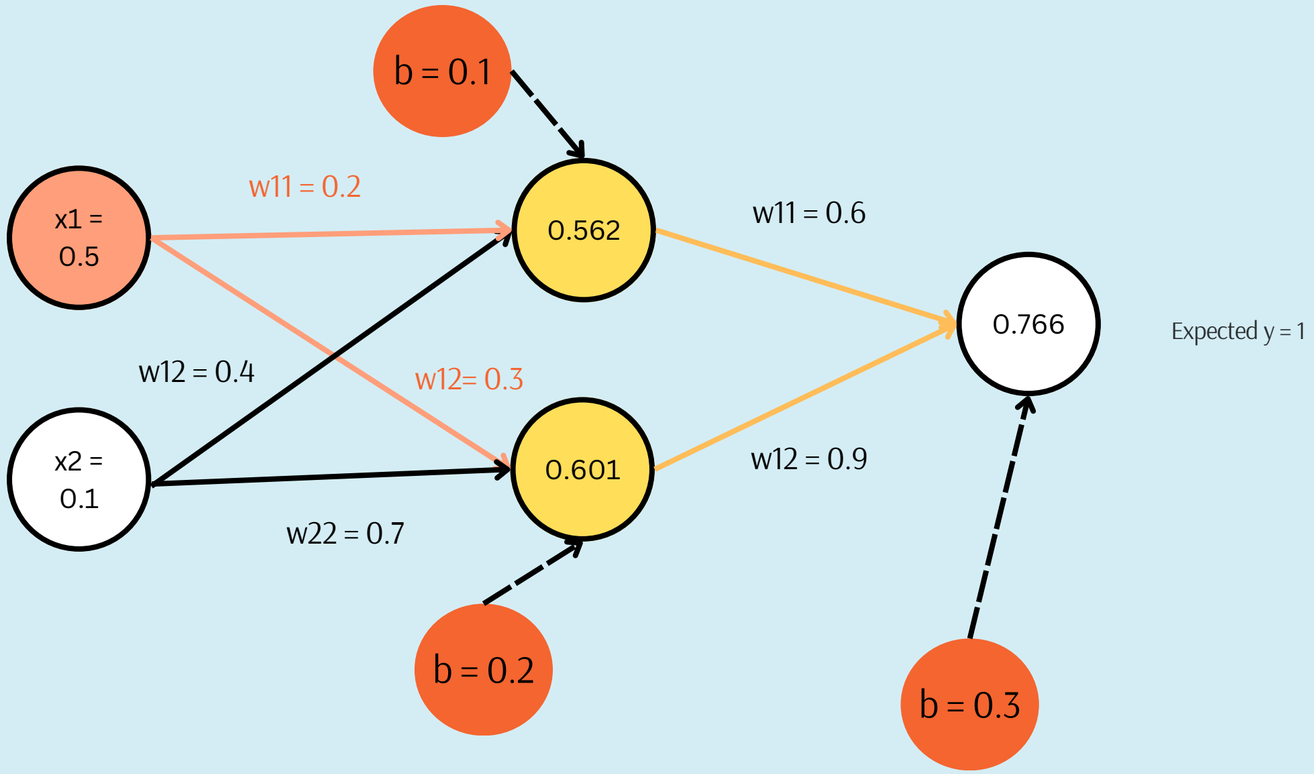
- Gradients with respect to the biases in the hidden layer:

$$\frac{\partial L}{\partial b_1^{(1)}} = \delta_1^{(1)} \approx -0.0346$$

$$\frac{\partial L}{\partial b_2^{(1)}} = \delta_2^{(1)} \approx -0.0505$$

Backpropagation

Example 3



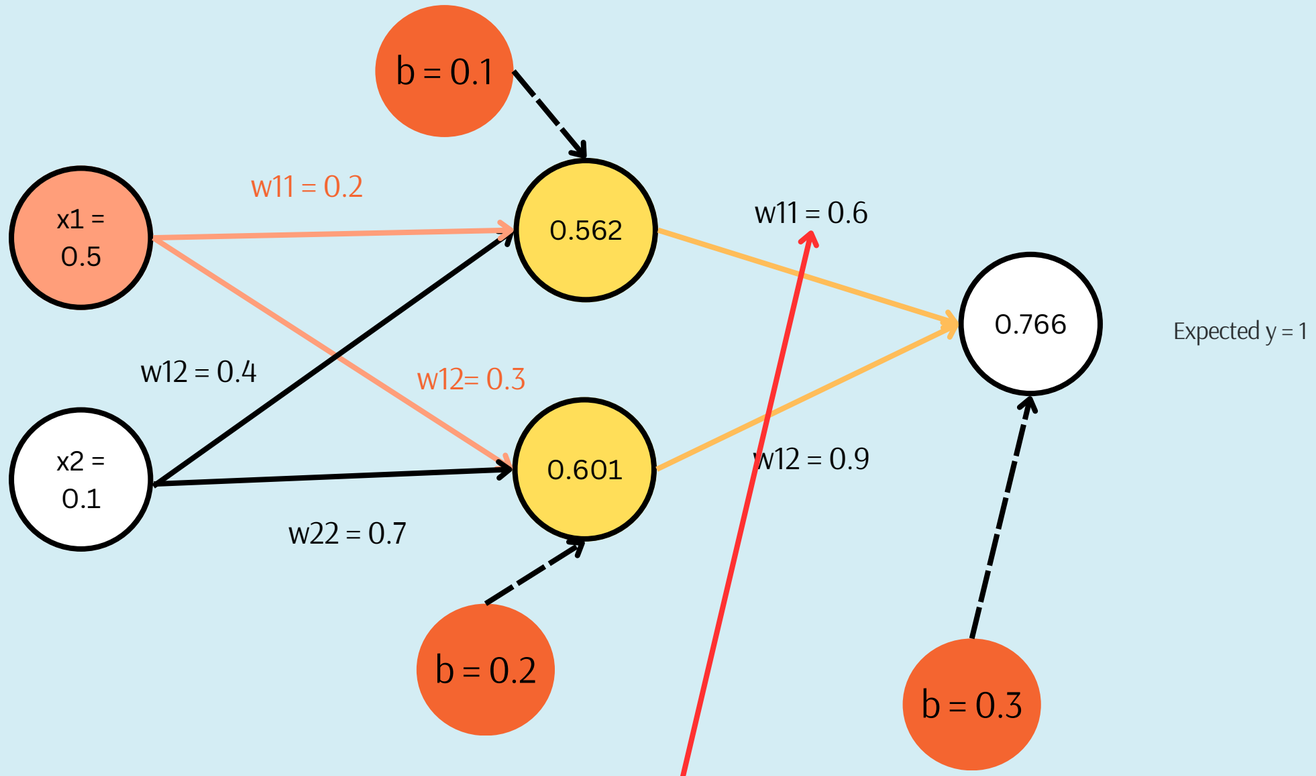
Step 4: Update Weights (Gradient Descent)

For each weight, update according to:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

Backpropagation

Example 3



Let's assume a learning rate $\eta = 0.01$.

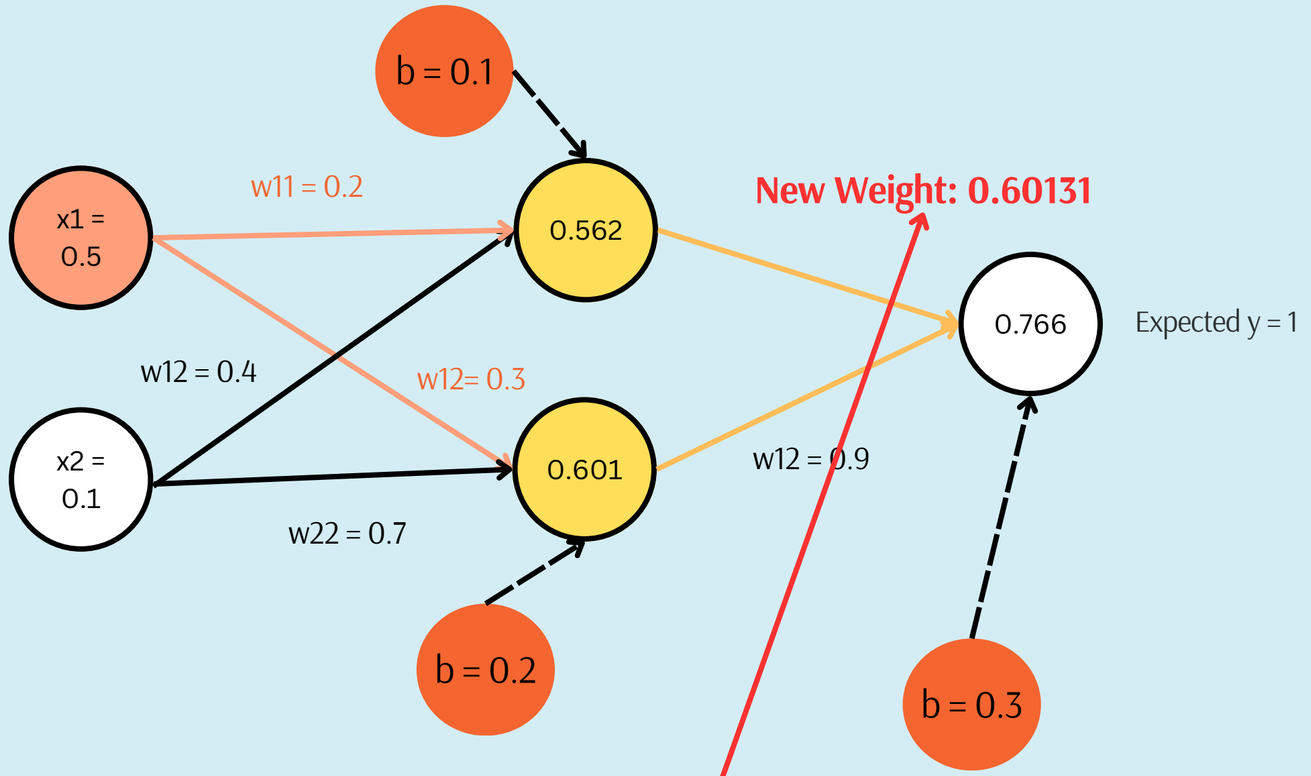
We would plug in the gradients computed above to update the weights. For example:

$$w_{11}^{(2)} = w_{11}^{(2)} - \eta \cdot \frac{\partial L}{\partial w_{11}^{(2)}} = 0.6 - 0.01 \cdot (-0.131) = 0.6 + 0.00131 = 0.60131$$

and so on for all the weights and biases.

Backpropagation

Example 3



Let's assume a learning rate $\eta = 0.01$.

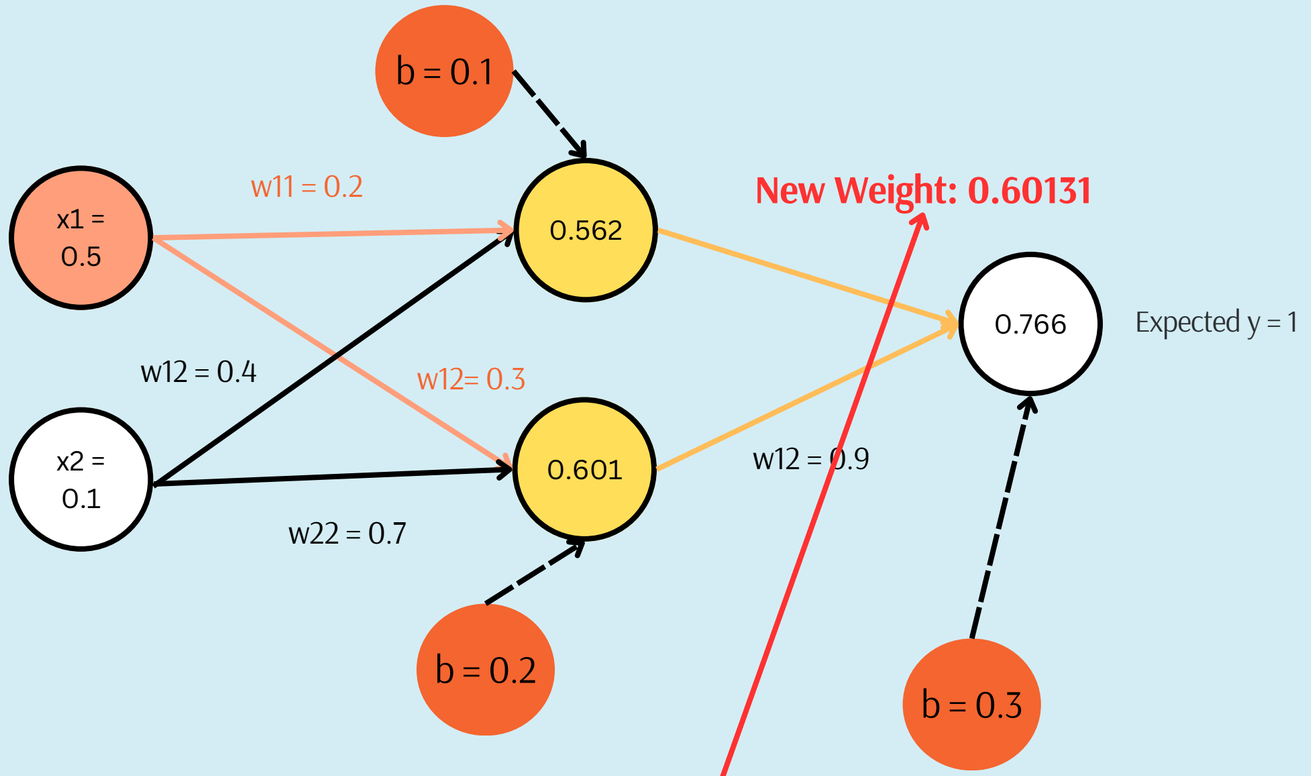
We would plug in the gradients computed above to update the weights. For example:

$$w_{11}^{(2)} = w_{11}^{(2)} - \eta \cdot \frac{\partial L}{\partial w_{11}^{(2)}} = 0.6 - 0.01 \cdot (-0.131) = 0.6 + 0.00131 = 0.60131$$

and so on for all the weights and biases.

Backpropagation

Example 3



Let's assume a learning rate $\eta = 0.01$.

We would plug in the gradients computed above to update the weights. For example:

$$w_{11}^{(2)} = w_{11}^{(2)} - \eta \cdot \frac{\partial L}{\partial w_{11}^{(2)}} = 0.6 - 0.01 \cdot (-0.131) = 0.6 + 0.00131 = 0.60131$$

and so on for all the weights and biases.

Weight Updates (Using Gradient Descent)

Using a learning rate $\eta = 0.01$, the weight updates are:

- For $w_{11}^{(2)}$:

$$w_{11}^{(2)} = 0.6 - 0.01 \cdot (-0.131) = 0.6 + 0.00131 = 0.60131$$

- For $w_{12}^{(2)}$:

$$w_{12}^{(2)} = 0.9 - 0.01 \cdot (-0.141) = 0.9 + 0.00141 = 0.90141$$

- For $b_1^{(2)}$:

$$b_1^{(2)} = 0.3 - 0.01 \cdot (-0.234) = 0.3 + 0.00234 = 0.30234$$

- For $w_{11}^{(1)}$:

$$w_{11}^{(1)} = 0.2 - 0.01 \cdot (-0.0173) = 0.2 + 0.000173 = 0.200173$$

- For $w_{12}^{(1)}$:

$$w_{12}^{(1)} = 0.4 - 0.01 \cdot (-0.00346) = 0.4 + 0.0000346 = 0.400035$$

- For $w_{21}^{(1)}$:

$$w_{21}^{(1)} = 0.3 - 0.01 \cdot (-0.0253) = 0.3 + 0.000253 = 0.300253$$

- For $w_{22}^{(1)}$:

$$w_{22}^{(1)} = 0.7 - 0.01 \cdot (-0.00505) = 0.7 + 0.0000505 = 0.700051$$

- For $b_1^{(1)}$:

$$b_1^{(1)} = 0.1 - 0.01 \cdot (-0.0346) = 0.1 + 0.000346 = 0.100346$$

- For $b_2^{(1)}$:

$$b_2^{(1)} = 0.2 - 0.01 \cdot (-0.0505) = 0.2 + 0.000505 = 0.200505$$

Backpropagation

Some things to keep in mind

- Initializing the weights
  should not be initialized to 0,should be random numbers between -1 and 1

- Early stopping
  Use a validation set V different from the training set.
  After k rounds for some fixed k (e.g., 100):
  • check the loss on V
  • if the loss starts to increase, stop training!

- Scaling
  The $z = \boldsymbol{w} \cdot \boldsymbol{x}$ shouldn't be too large for this to work,
  roughly z shouldn't be much more than 1

Vanishing gradient

The vanishing gradient problem is a challenge that can occur during the training of deep neural networks, particularly in the context of gradient-based optimization algorithms like backpropagation. It refers to the phenomenon where the gradients of the loss function with respect to the weights become extremely small as the network's depth increases, making it difficult for the network to learn effectively.

Vanishing gradient

“Solve the problem”

ReLU (Rectified Linear Unit):

- **Formula**: $f(x) = \max(0, x)$
- **Why it's good**:
  - ReLU is the most commonly used activation function in deep neural networks. It helps mitigate the vanishing gradient problem by allowing gradients to flow through without squashing them for positive inputs.
  - It has a constant gradient of 1 for $x > 0$, which prevents gradients from vanishing for these positive values.

“Cause the problem”

Sigmoid:

- **Formula**: $f(x) = \frac{1}{1+e^{-x}}$
- **Why it's bad**:
  - Sigmoid activation outputs values between 0 and 1, which can lead to very small gradients, especially for inputs far from 0.
  - When inputs are very large or small, the derivative of the sigmoid becomes near zero, causing the gradients to vanish as they propagate back.
  - Leads to **saturation**, where gradients are almost zero for large positive or negative inputs, making learning slow or ineffective.

Vanishing gradient

“Cause the problem”

Tanh (Hyperbolic Tangent):

- **Formula**: $f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$
- **Why it's bad**:
  - Like the sigmoid function, tanh squashes its output (between -1 and 1), which causes the gradients to shrink, especially for inputs far from zero.
  - Although tanh centers its output around 0 (which can be better for gradient flow than sigmoid), it still suffers from the vanishing gradient problem for large positive or negative inputs.

Softmax (in the hidden layers):

- **Formula**: $f(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$
- **Why it's bad**:
  - Softmax is typically used in the output layer for classifica tasks, but when used in hidden layers, it can cause gradi to diminish because it squashes the output into very sm ranges.
  - Using softmax in the hidden layers is generally not recommended for deep networks.

Useful links:

FNN and backpropagation in code

https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/

Backpropagation theory and calculus

https://www.youtube.com/watch?v=Ilg3gGewQ5U

https://www.youtube.com/watch?v=tIeHLnjs5U8

Quiz

Q2) What is the primary purpose of the activation function in a neural network?

Quiz

Q3) In the context of neural networks, what is the chain rule used for?

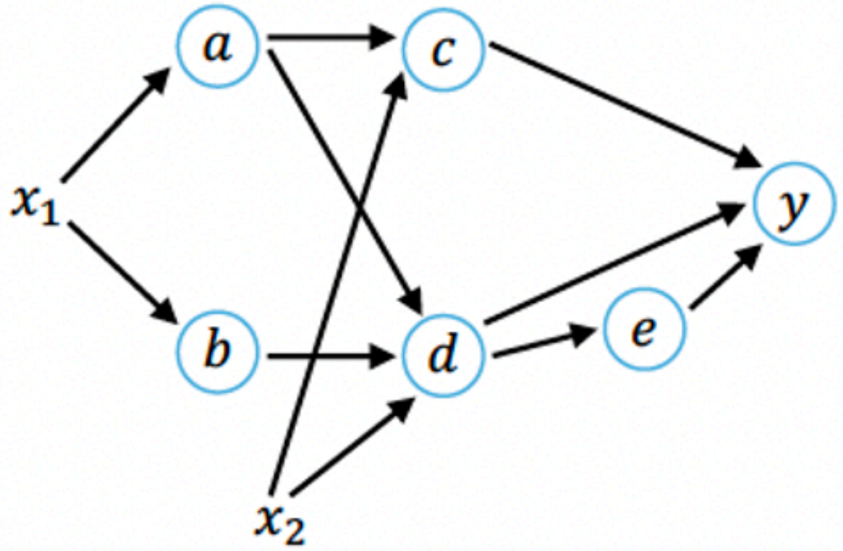
Quiz

Q4) Name 4 activation functions, and also give activation function which is zero-centered and often used to mitigate the vanishing gradient problem?

Quiz

Problem 3

Consider the following neural network:



The output (activation) for any neuron $q \in \{a, b, c, \ldots\}$ is given by:

$$g\left(b + \sum_l v_l x_l + \sum_u w_u u\right) \tag{1}$$

where $g(\cdot)$ is some activation function, b is the bias term, $v_l = 0$ for neurons that have no direct inputs x_l, and $w_u = 0$ for neurons that have no inputs from neuron u.

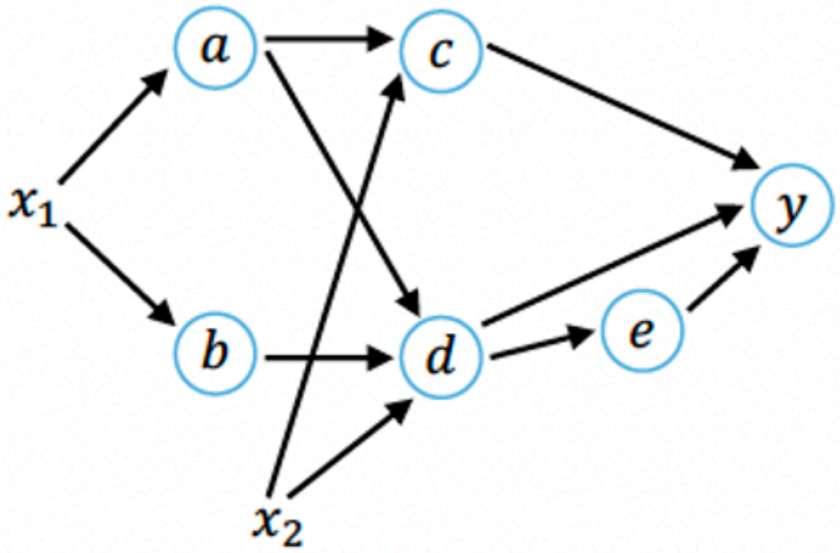
This task asks you to write down expressions for gradients of this network. Please consider the following advices before doing so:

- Write the expression in terms of:
  - $\frac{\partial q_1}{\partial q_2}$, where q_2 is direct input to q_1, and
  - $\frac{\partial q}{\partial x_l}$, where the network input x_l is direct input to neuron q.
- This task is about the sequences of partial derivatives along the edges.
- You **do not need** to plug in or compute how $\frac{\partial q_1}{\partial q_2}$ or $\frac{\partial q}{\partial x_l}$ looks like.
- You **do not need** to multiply out terms in parentheses, so $(a+b)c$ or $((a+b)c+(d+e)f)g$ is fine to keep like that.

Quiz

Problem 3

Consider the following neural network:



The output (activation) for any neuron $q \in \{a, b, c, \ldots\}$ is given by:

$$g\left(b + \sum_l v_l x_l + \sum_u w_u u\right)$$

where $g(\cdot)$ is some activation function, b is the bias term, $v_l = 0$ for neurons that have no direct inputs x_l, and $w_u = 0$ for neurons that have no inputs from neuron u.

This task asks you to write down expressions for gradients of this network. Please consider the following advices before doing so:

- Write the expression in terms of:
  - $\frac{\partial q_1}{\partial q_2}$, where q_2 is direct input to q_1, and
  - $\frac{\partial q}{\partial x_l}$, where the network input x_l is direct input to neuron q.
- This task is about the sequences of partial derivatives along the edges.
- You **do not need** to plug in or compute how $\frac{\partial q_1}{\partial q_2}$ or $\frac{\partial q}{\partial x_l}$ looks like.
- You **do not need** to multiply out terms in parentheses, so $(a+b)c$ or $((a+b)c+(d+e)f)g$ is fine to keep like that.

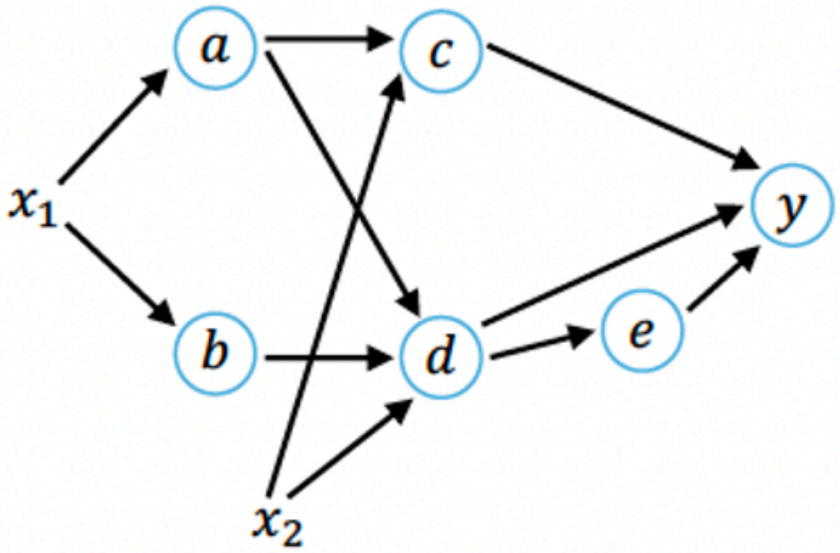
Write down an expression for the following gradients of this network:

(1) **3a** $\frac{\partial y}{\partial x_2}$ **3b** $\frac{\partial y}{\partial x_1}$

Quiz - Solution

Problem 3

Consider the following neural network:



The output (activation) for any neuron $q \in \{a, b, c, \ldots\}$ is given by:

$$g\left(b + \sum_l v_l x_l + \sum_u w_u u\right)$$

3a

$\frac{\partial y}{\partial x_2}$

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial c}\frac{\partial c}{\partial x_2} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial x_2} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial x_2}$$
$$= \frac{\partial y}{\partial c}\frac{\partial c}{\partial x_2} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial x_2} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial x_2}$$

3b

$\frac{\partial y}{\partial x_1}$

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial c}\frac{\partial c}{\partial x_1} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial x_1} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial x_1}$$
$$= \frac{\partial y}{\partial c}\frac{\partial c}{\partial x_1} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial x_1} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial x_1}$$
$$= \frac{\partial y}{\partial c}\frac{\partial c}{\partial a}\frac{\partial a}{\partial x_1} + \left(\frac{\partial y}{\partial d} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial d}\right)\frac{\partial d}{\partial x_1}$$
$$= \frac{\partial y}{\partial c}\frac{\partial c}{\partial a}\frac{\partial a}{\partial x_1} + \left(\frac{\partial y}{\partial d} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial d}\right)\left(\frac{\partial d}{\partial a}\frac{\partial a}{\partial x_1} + \frac{\partial d}{\partial b}\frac{\partial b}{\partial x_1}\right)$$

Q1) Learning rate = 0.1

input : (x1,x2) = (2,-1)

activation function: sigmoid

a)find output at node h2

b) find output at node h3

Now supposing there is no activation function at the output layer

c) find the final output