# Neural Network (Perceptrons)

# Classification dataset

| Hours Study | Pass |
| --- | --- |
| 2 | No |
| 3 | No |
| 4 | Yes |
| 5 | No |
| 6 | Yes |
| 7 | Yes |
| 8 | Yes |
| 9 | Yes |
| 10 | ? |

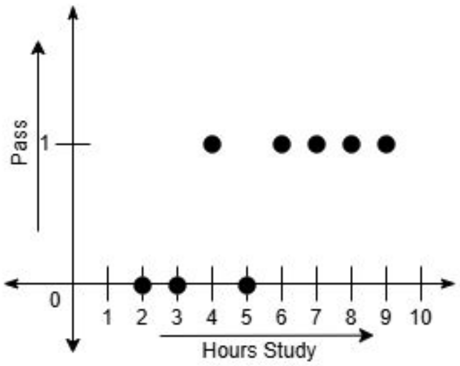
| Hours Study | Pass |
| --- | --- |
| 2 | 0 |
| 3 | 0 |
| 4 | 1 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |
| 10 | ? |

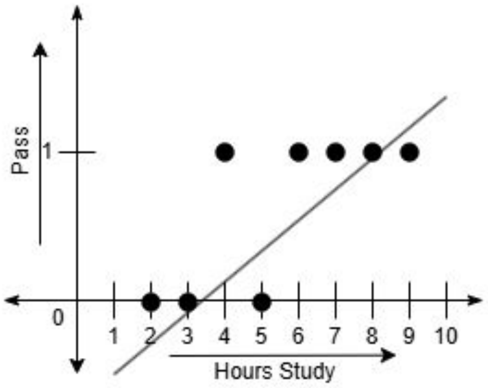
- Here, let’s say we are trying to predict whether a student is going to pass based on hours study
- Hours study, in this case is the feature and Pass, is a label
- Pass is a categorical variable consisting of two values {Yes, No}
- Such categorical variable outcome prediction is referred to as classification problem
- It is a task of supervised learning

| Hours Study | Pass |
|---|---|
| 2 | No |
| 3 | No |
| 4 | Yes |
| 5 | No |
| 6 | Yes |
| 7 | Yes |
| 8 | Yes |
| 9 | Yes |
| 10 | ? |

| Hours Study | Pass |
|---|---|
| 2 | 0 |
| 3 | 0 |
| 4 | 1 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |
| 10 | ? |

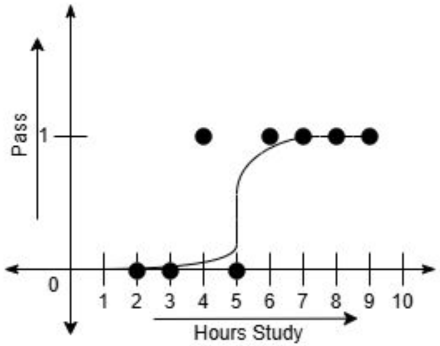
- Let's try to plot the dataset




- Linear models can be problematic to fit such data

- Thus, we convert the linear hypothesis to a non-linear one to fit the data




- One such non-linear activation function here used is Sigmoid
- Such an activation function converts a linear hypothesis into non-linear one
- Such nonlinear classification is known as logistic regression
- There are other activation functions available (ReLU, Tanh)

# Sigmoid function

- If we are trying to fit a linear equation z = w.x+b to a non-linear pattern, we convert it to a nonlinear equation instead
- The conversion is done through non-linear activation functions
- One such function is the sigmoid function.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Where, z = w.x+b [The linear function that we are trying to convert]

w = weight [Learnable parameter]

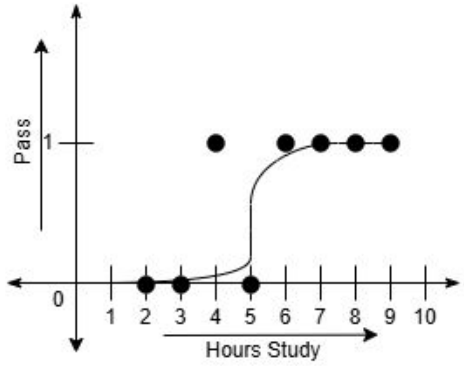
b = bias [Learnable parameter]

x = feature value [For multiple features, it’s a matrix multiplied with a weight matrix]

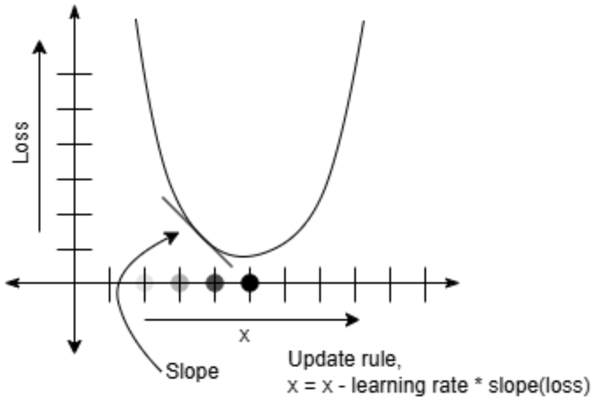
Let’s assume,

Hours Study = x

Pass = y




- We’ll have to figure out z = w.x+b, and pass it through the nonlinear function
- The challenge is to find appropriate values for w and b, which we can do through gradient descent
- For gradient descent, we calculate the derivative of the loss function, and subtract the derivative from the respective parameter to update it



# Loss function

Logistic regression uses the following loss function:

loss = -y.log(a) - (1-y).log(1-a)

Where,

y = Ground truth

a = Predicted value

Intuition behind the loss function:

If y = 1 and a = 0.000005 [Extreme case of misclassification]

loss = -1.log0 - (1-1).log(1-0) = 5.301 [High loss value]

If y = 1 and a = 1 [Best case of correct classification]

loss = -1.log1 - (1-1).log(1-1) = 0 [Low loss value]

If y = 0 and a = .000095 [Extreme case of misclassification]

loss = -0.log1 - (1-0).log(1-1) = 4.022 [High loss value]

If y = 0 and a = 0 [Best case of correct classification]

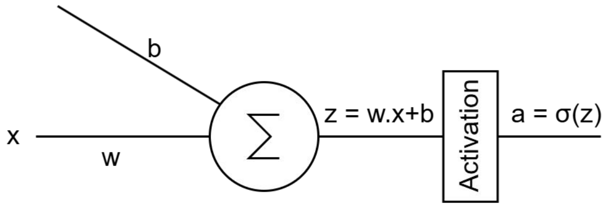
loss = -0.log0 - (1-0).log(1-0) = 0 [Low loss value]

# Steps of logistic regression

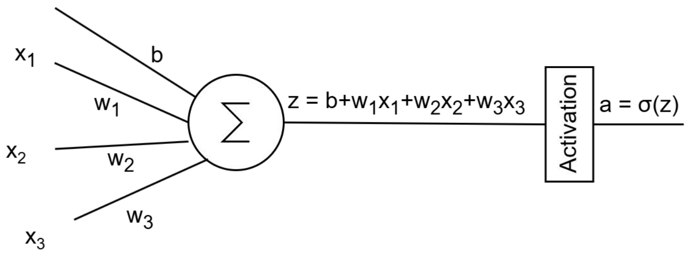
- Randomly initialize the w and b for $z=w.x+b$
- Calculate the z for the initial w and b
- Calculate the loss
- Apply gradient descent to update the w and b
- Repeat the process until convergence

# Logistic regression to neural networks

- Logistic regression can be expressed as a single neuron neural network
- Where, for a single feature and a single neuron, the network looks like:



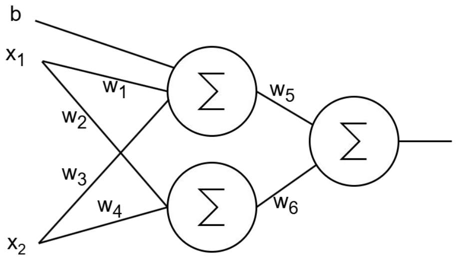

- For multiple features on a single neuron, the architecture may look like:



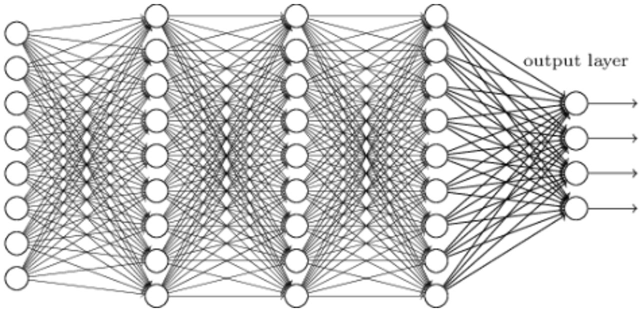

- Single layer neural networks are referred to as perceptrons

# Logistic regression to neural networks

- A network can involve multiple neurons

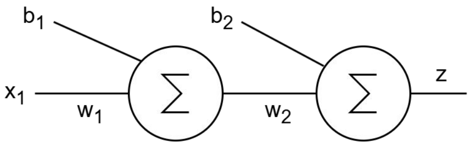



- And it can become fairly large and complex



# Neural networks

- A method of processing based on multiple connected processing unit
- Each neuron is a linear function, followed by a nonlinear activation function
- The connectivity of the neurons build up a nested function
- Can learn complex patterns




- In the neural network above: $z = \sigma(\sigma(x_1.w_1+b_1).w_2+b_2)$
- In a larger network, the function becomes much more nested and complex with many learnable parameters, giving it the ability to learn complex patterns
- The learning process stays the same. After initialization, the weights $\{w_1, w_2, w_3,\ldots, w_n\}$ and biases $\{b_1, b_2, b_3,..., b_m\}$ have to be updated through gradient descent

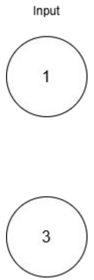
# Neural networks

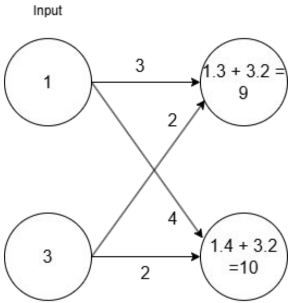
- The types of neural networks that we talked about until now is known as feedforward neural networks
- There are also other forms of neural networks
- Some of the basic types are:
  - Feedforward Neural Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Generative Adversarial Networks
  - Transformers
  - … and Many More!
- For this lecture, we are going to talk about the feedforward networks only

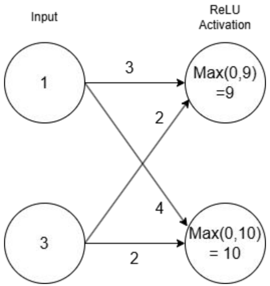
# Training a Neural Network

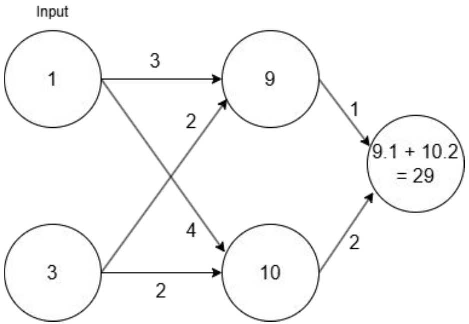
- Collect dataset
- Define the architecture of the Neural Network model that is to be trained on the data
- Initialize all the weights and biases
- Calculate output based on the initialized weights and bias, which is called forward propagation
- Calculate loss using an appropriate loss function
- Update the last layer weights and bias using the gradient of the loss
- Propagate the gradient to update the backwards layers, known as backpropagation
- Run in a loop until convergence

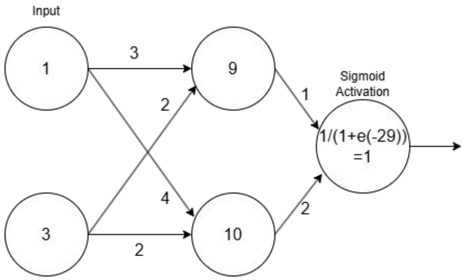
# Forward Propagation



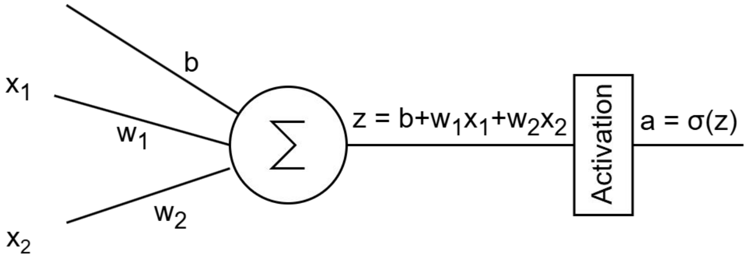



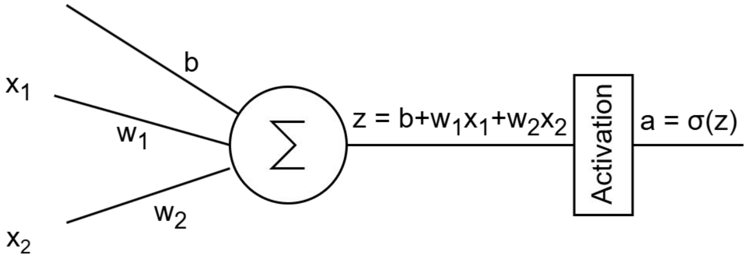



# Back to the Single Neuron Perceptron




- Here, there are two input feature values ($x_1$, $x_2$), output is (a)
- The loss function discussed in the logistic regression can be used here, referred to as sigmoid loss function

# Update process

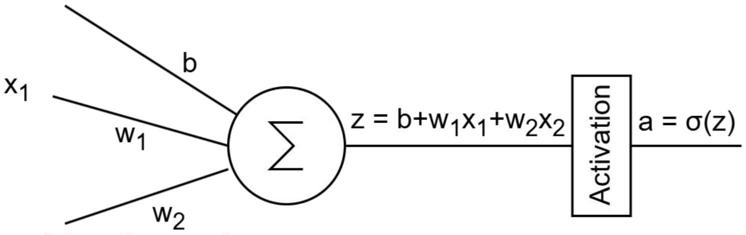



- Calculate output *a* for all the data points in the dataset
- Calculate loss for *n* number of data points using:

$$L = (\sum_{i=1}^{n} -y_i log a_i - (1-y_i) log(1-a_i))/n$$

- To update $x_i$, Calculate derivative of loss using $\frac{dL}{dx_i}$
- Update $x_i$ using $x_i = x_i - \alpha * \frac{dL}{dx_i}$
- Where, α is the learning rate
- Update all the weights and biases in the network

# Gradient Calculation




$L = -yloga - (1-y)log(1-a)$

Where, $a = \sigma(z) = \frac{1}{1+e^{-z}}$

Where, $z = x_1w_1 + x_2w_2 + b$

To update the weight $w_1$, we need to perform $w_1 = w_1 - \frac{dL}{dw_1}$

All these individual functions are nested to each other.

Thus, to calculate the derivative $\frac{dL}{dw_1}$, according to the chain rule,

First, we need to calculate $\frac{dL}{da}$ [Derivative of L with respect to a]

Then, we need to calculate $\frac{da}{dz}$ [Derivative of a with respect to z]

Afterward, we need to calculate $\frac{dz}{dw_1}$ [Derivative of $W_1$ with respect to w]

Finally, $\frac{dL}{dw_1}$ will be, according to the chain rule:

$$\frac{dL}{dw_1} = \frac{dz}{dw_1} * \frac{da}{dz} * \frac{dL}{da}$$

# Let's calculate the derivatives

$$L = -ylog(a) - (1-y)log(1-a)$$

$$\frac{dL}{da} = -\frac{d}{da}ylog(a) - \frac{d}{da}(1-y)log(1-a)$$

$$= -y\frac{d}{da}log(a) - (1-y)\frac{d}{da}log(1-a)$$

$$= -y\frac{1}{a}\frac{d}{da}(a) - (1-y)\frac{1}{1-a}\frac{d}{da}(1-a)$$

$$= -\frac{y}{a} * 1 - \frac{1-y}{1-a}(\frac{d}{da}(1) - \frac{d}{da}(a))$$

$$= -\frac{y}{a} - \frac{1-y}{1-a}(-1)$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

# Let's calculate the derivatives

Meanwhile, $a = \frac{1}{1+e^{-z}} = (1+e^{-z})^{-1}$

$\frac{da}{dz} = \frac{d}{dz}(1+e^{-z})^{-1}$

$= -1(1+e^{-z})^{-2}\frac{d}{dz}(1+e^{-z})$

$= -\frac{1}{(1+e^{-z})^2}(\frac{d}{dz}1 + \frac{d}{dz}e^{-z})$

$= -\frac{1}{(1+e^{-z})^2}(0 - e^{-z})$

$= \frac{e^{-z}}{(1+e^{-z})^2}$

Also, $z = x_1w_1 + x_2w_2 + b$

$\frac{dz}{dw_1} = \frac{d}{dw_1}(x_1w_1 + x_2w_2 + b)$

$= \frac{d}{dw_1}x_1w_1 + \frac{d}{dw_1}x_2w_2 + \frac{d}{dw_1}b$

$= x_1 + 0 + 0$

$= x_1$

# Let's calculate the derivatives

Finally, $\frac{dL}{dw_1} = \frac{dz}{dw_1} * \frac{da}{dz} * \frac{dL}{da}$

$$= (-\frac{y}{a} + \frac{1-y}{1-a})(\frac{e^{-z}}{(1+e^{-z})^2})x_1$$

$$= (-\frac{y}{a} + \frac{1-y}{1-a})(\frac{e^{-z}.x_1}{(1+e^{-z})^2})$$

Simplifying,

$$= (-\frac{y}{\frac{1}{1+e^{-z}}} + \frac{1-y}{1-\frac{1}{1+e^{-z}}}) * \frac{e^{-z}x_1}{(1+e^{-z})^2}$$

$$= (-\frac{y}{\frac{1}{1+e^{-z}}} + \frac{1-y}{\frac{1-e^{-z}-1}{1+e^{-z}}}) * \frac{e^{-z}x_1}{(1+e^{-z})^2}$$

$$= (-\frac{y}{\frac{1}{1+e^{-z}}} + \frac{1-y}{\frac{e^{-z}}{1+e^{-z}}}) * \frac{e^{-z}x_1}{(1+e^{-z})^2}$$

$$= (-y(1+e^{-z}) + (1-y)\frac{1+e^{-z}}{e^{-z}}) * \frac{e^{-z}x_1}{(1+e^{-z})^2}$$

$$= (-y + \frac{1-y}{e^{-z}}) * (1+e^{-z}) * \frac{e^{-z}x_1}{(1+e^{-z})^2}$$

$$= (-y + \frac{1-y}{e^{-z}}) * \frac{e^{-z}x_1}{1+e^{-z}}$$

$$= \frac{-e^{-z}+1-y}{e^{-z}} * \frac{e^{-z}x_1}{1+e^{-z}}$$

$$= \frac{-e^{-z}+1-y}{1+e^{-z}} * x_1$$

$$= (\frac{1}{1+e^{-z}} + \frac{-y-e^{-z}y}{1+e^{-z}}) * x_1$$

$$= (\frac{1}{1+e^{-z}} + \frac{-y(1+e^{-z})}{1+e^{-z}}) * x_1$$

$$= (\frac{1}{1+e^{-z}} - y) * x_1$$

So, for a single neuron perceptron,

The gradient with respect to $w_1$ is,

$$\frac{dL}{dw_1} = (\frac{1}{1+e^{-z}} - y) * x_1$$

The gradient with respect to $w_2$ would be,

$$\frac{dL}{dw_2} = (\frac{1}{1+e^{-z}} - y) * x_2$$

This is for a single data point. For *n* number of samples, all the gradients have to be summed up and then averaged

$$\frac{dL}{dw_1} = (\sum_{i=1}^{n}(\frac{1}{1+e^{-z_i}} - y_i) * x_{1i})/n$$

$$\frac{dL}{dw_2} = (\sum_{i=1}^{n}(\frac{1}{1+e^{-z_i}} - y_i) * x_{2i})/n$$

$$\vdots$$

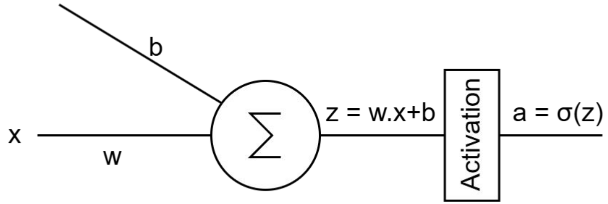
$$\frac{dL}{dw_k} = (\sum_{i=1}^{n}(\frac{1}{1+e^{-z_i}} - y_i) * x_{ki})/n$$

# Some Basic Simulations

Let's have a toy dataset

| X | Y |
|---|---|
| .1 | 0 |
| .2 | 0 |
| .3 | 1 |
| .4 | ? |

We are trying to fit it with the following perceptron




*w* and *b* are the unknowns, which have to be figured out

For three data points, the loss function is:

$$L = (\sum_{i=1}^{3} -y_i log a_i - (1 - y_i) log(1 - a_i))/3$$

Initializing,

W = .7

b = .1

Thus, z = .7x+.1

- As the weights are randomly initialized, it should not fit our data points well, resulting in large error.
- Let’s calculate the error through the loss function

(Please note that we should use the normalized version of all the values for faster

and appropriate convergence. For simplicity, we are ignoring that step here.)

- Now, calculating the sum of loss for the three points in the dataset:

$$L = -0log\frac{1}{1+e^{-.7*.1-.1}} - (1-0)log(1-\frac{1}{1+e^{-.7*.1-.1}})$$ [For the first data point]

$$-0log\frac{1}{1+e^{-.7*.2-.1}} - (1-0)log(1-\frac{1}{1+e^{-.7*.2-.1}})$$ [For the second data point]

$$-1log\frac{1}{1+e^{-.7*.3-.1}} - (1-1)log(1-\frac{1}{1+e^{-.7*.3-.1}})$$ [For the third data point]

$$= -log(1-\frac{1}{1+e^{-.7*.1-.1}}) - log(1-\frac{1}{1+e^{-.7*.2-.1}}) - log\frac{1}{1+e^{-.7*.3-.1}}$$

$$= -log(1-.54) - log(1-.55) - log.57$$

$$= -log.46 - log.45 - log.57$$

$$= 0.77 + 0.79 + 0.56$$

$$= 2.12$$

Average loss value across the three data points is 2.12/3 = 0.71

- Let’s try to reduce the loss value
- We’ll have to update both w and b using:

$$\frac{dL}{dw} = (\sum_{i=1}^{n}(\frac{1}{1+e^{-z_i}} - y_i) * x_i)/n$$

$$w = w - \alpha * \frac{dL}{dw}$$

$$\frac{dL}{db} = (\sum_{i=1}^{n}(\frac{1}{1+e^{-z_i}} - y_i))/n$$

$$b = b - \alpha * \frac{dL}{db}$$

- Where, α is the learning rate

$$\frac{dL}{dw} = ((\frac{1}{1+e^{.7*.1-.1}} - 0) * .1 + (\frac{1}{1+e^{.7*.2-.1}} - 0) * .2 + (\frac{1}{1+e^{.7*.3-.1}} - 1) * .3)/3$$

$$= (.054 + .112 - .127)/3$$

$$= .039/3$$

$$= .013$$

$$\frac{dL}{db} = ((\frac{1}{1+e^{.7*.1-.1}} - 0) + (\frac{1}{1+e^{.7*.2-.1}} - 0) + (\frac{1}{1+e^{.7*.3-.1}} - 1))/3$$

$$= (.54 + .56 - .42)/3$$

$$= .68/3$$

$$= .23$$

- Thus, if α=1, the updated w and b are:
- w = .7 - 1 * .013 = 0.687
- B = .1 - 1 * .23 = -0.13

Recalculating the sum of loss:

$$L = -0log\frac{1}{1+e^{-.687*.1+.13}} - (1-0)log(1 - \frac{1}{1+e^{-.687*.1+.13}})$$

$$-0log\frac{1}{1+e^{-.687*.2+.13}} - (1-0)log(1 - \frac{1}{1+e^{-.687*.2+.13}})$$

$$-1log\frac{1}{1+e^{-.687*.3+.13}} - (1-1)log(1 - \frac{1}{1+e^{-.687*.3+.13}})$$

$$= -log(1 - \frac{1}{1+e^{-.687*.1+.13}}) - log(1 - \frac{1}{1+e^{-.687*.2+.13}}) - log\frac{1}{1+e^{-.687*.3+.13}}$$

$$= -log(1 - .48) - log(1 - .50) - log.52$$

$$= -log.52 - log.50 - log.52$$

$$= 0.65 + 0.69 + 0.65$$

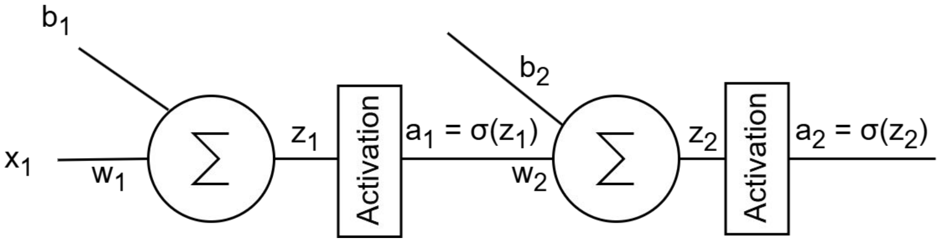
$$= 1.99$$

Average = 1.99 / 3 = 0.66

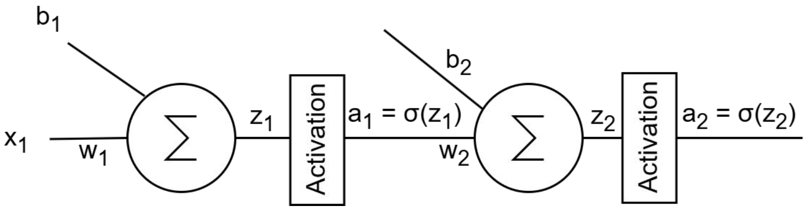
Slightly better than before!

We need to run this many times in a loop to get further improvements. (Depending on the nature of data, good enough convergence might not be possible.)

# But what about a larger network?




$L = -yloga_2 - (1-y)log(1-a_2)$
Where, $a_2 = \sigma(z_2) = \frac{1}{1+e^{-z_2}}$
Where, $z_2 = a_1 w_2 + b_2$
Where, $a_1 = \sigma(z_1) = \frac{1}{1+e^{-z_1}}$
Where, $z_1 = x_1 w_1 + b_1$
At first, to update the weight $w_2$, we need to perform $w_2 = w_2 - \frac{dL}{dw_2}$
All these individual functions are nested to each other.
Thus, to calculate the derivative $\frac{dL}{dw_2}$, according to the chain rule,
First, we need to calculate $\frac{dL}{da_2}$ [Derivative of L with respect to $a_2$]
Then, we need to calculate $\frac{da_2}{dz_2}$ [Derivative of $a_2$ with respect to $z_2$]
Afterward, we need to calculate $\frac{dz_2}{dw_2}$ [Derivative of $z_2$ with respect to $w_2$]
Finally, $\frac{dL}{dw_2}$ will be, according to the chain rule:
$\frac{dL}{dw_2} = \frac{dz_2}{dw_2} * \frac{da_2}{dz_2} * \frac{dL}{da_2}$




Meanwhile, to update the weight $w_1$, we need to perform $w_1 = w_1 - \frac{dL}{dw_1}$

To calculate the derivative $\frac{dL}{dw_1}$, according to the chain rule,

First, we need to calculate $\frac{dL}{da_2}$ [Derivative of L with respect to $a_2$]

Then, we need to calculate $\frac{da_2}{dz_2}$ [Derivative of $a_2$ with respect to $z_2$]

Afterward, we need to calculate $\frac{dz_2}{da_1}$ [Derivative of $z_2$ with respect to $a_1$]

In the next step, calculate $\frac{da_1}{dz_1}$ [Derivative of $a_1$ with respect to $z_1$]

Next, calculate $\frac{dz_1}{dw_1}$ [Derivative of $z_1$ with respect to $w_1$]

Finally, $\frac{dL}{dw_1}$ will be, according to the chain rule:

$$\frac{dL}{dw_1} = \frac{dz_1}{dw_1} * \frac{da_1}{dz_1} * \frac{dz_2}{da_1} * \frac{da_2}{dz_2} * \frac{dL}{da_2}$$

$$\frac{dL}{dw_1} = \frac{dz_1}{dw_1} * \frac{da_1}{dz_1} * \frac{dz_2}{da_1} * \frac{da_2}{dz_2} * \frac{dL}{da_2}$$

- This is a fairly large differentiation and, with increasing layer count, can very quickly go out of hand. But wait…

$$\frac{dL}{dw_1} = \frac{dz_1}{dw_1} * \frac{da_1}{dz_1} * \frac{dz_2}{da_1} * \boxed{\frac{da_2}{dz_2} * \frac{dL}{da_2}}$$

- While calculating the derivative w.r.t. $w_{2,}$ the red-marked portion has already been calculated. We can just get the numerical values and plug them in.
- For the gradient of each of the layer, we just need to store the gradient value for the earlier layer and plug them in, making the derivative calculation process much simpler.
- In such a case, we are just propagating the gradient back from the later layer, hence the name ‘backpropagation’.

- No matter how complex an expression is, it can be broken down into simple arithmetic expressions.
- It is possible to easily calculate the derivative of these individual simple expressions, get their values, apply chain rule repeatedly, and plug them in to the earlier operation from the later operation.
- This method is known as automatic differentiation and is capable of calculating all the gradients in a single backward swoop.
- This is how deep learning libraries are capable of calculating derivatives even in very complex networks.
- Some other differentiation alternatives would be symbolic differentiation (can become too complex in a large network) and numerical differentiation (not very accurate).