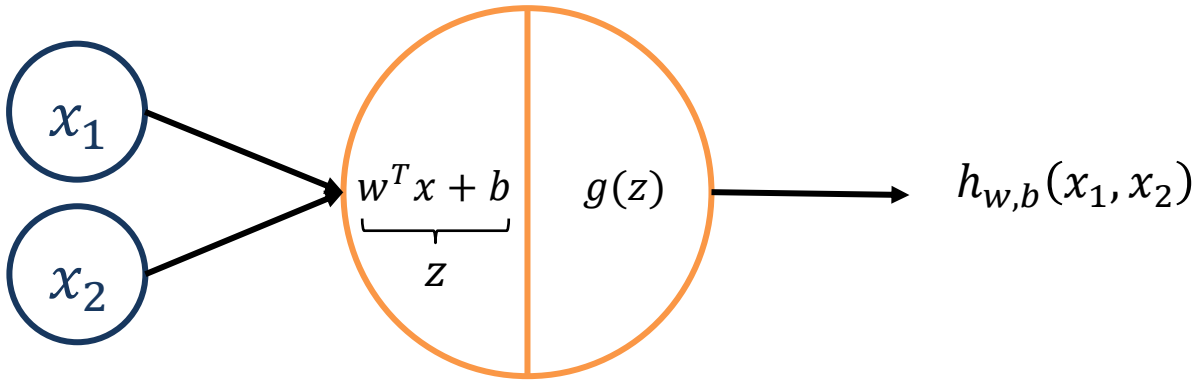
Neural Networks

Prof. Giuseppe Serra

Neural Network Representation

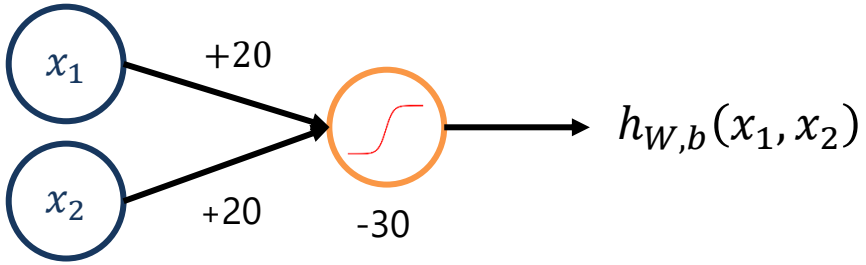
This network means that to compute the value of the decision function, we need to multiply first input x_1 with θ_1, second input x_2 with θ_2, then add two values and b, then apply the sigmoid function



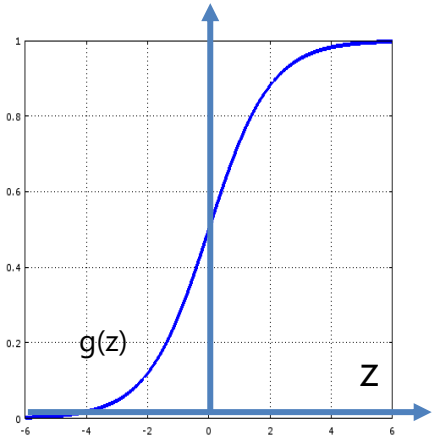
Simple Neural Networks

Example: AND Logic Port

$$y = x_1 \text{ AND } x_2$$



$$h_{w,b}(x_1, x_2) = g(-30 + 20x_1 + 20x_2)$$



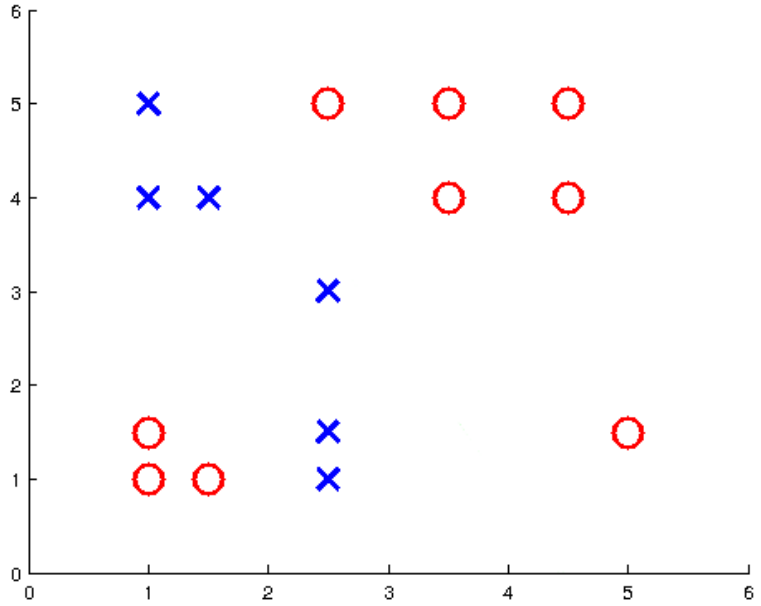
x_1	x_2	$h_{w,b}(x_1, x_2)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

A Complex Problem

Now consider Susan that has different movies tastes

Logistic Regression Algorithm is not able to solve it.

Hours spent on the laboratory activities

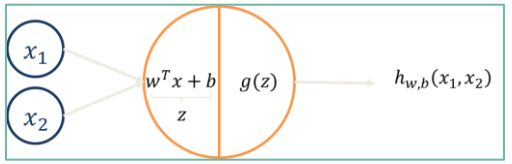


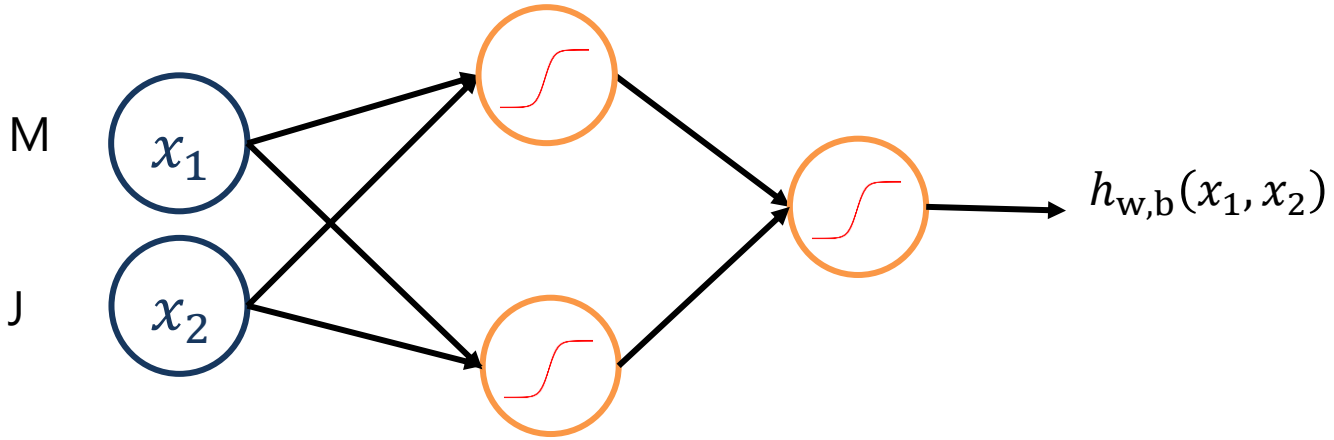
N. of lectures you attend

O corresponds to “Pass”
X corresponds to “Fail”

Neural Network with One Hidden Layer

Let's try to a Neural Network with one hidden layer!

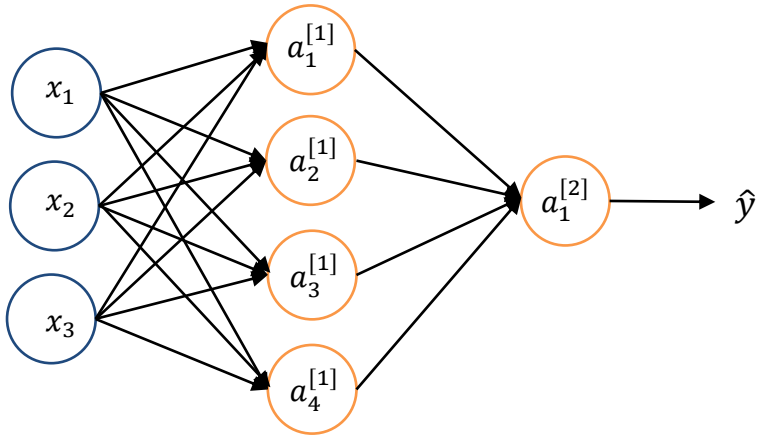




Input Layer Hidden Layer Output Layer

Let's try by yourself: https://lecture-demo.ira.uka.de/neural-network-demo/?preset=Binary%20Classifier%20for%20XOR

Neural Network Representation



$W^{[j]}$ = matrix of weights controlling function mapping from layer $j-1$ to layer j
$b^{[j]}$ = bias
$a^{[j]}$ = activation of vector for layer j
$g^{[j]}$ = activation function for layer j

Given in input X:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1) - Dimensionality

$$a^{[1]} = g^{[1]}(z^{[1]})$$
(4,1) (4,1) - Dimensionality

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1) - Dimensionality

$$a^{[2]} = g^{[2]}(z^{[2]})$$
(1,1) (1,1) - Dimensionality

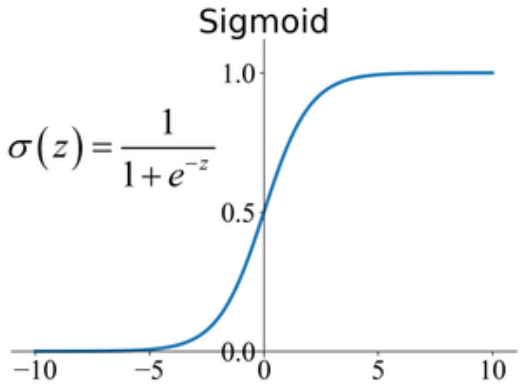
Activation Functions

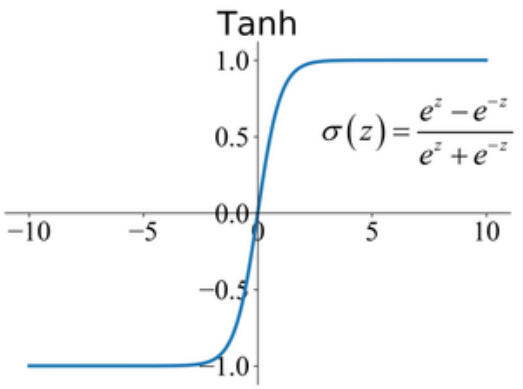
Sigmoid Function:

- it has seen frequent use historically, yet now it is rarely used in the intermediate layers, because it kills the gradient (remember $\frac{d\,\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$).
- It is commonly use in the last layer is you want a prediction value between [0,1]

Tanh Function:

- "…the hyperbolic tangent activation function typically performs better than the logistic sigmoid." – Page 195, Deep Learning, 2016
- $\frac{d\,tanh(z)}{dz} = \ldots = 1 - tanh^2(z)$





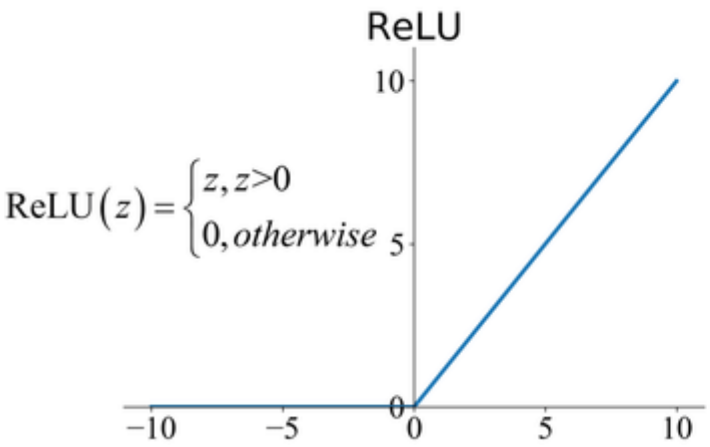
Activation Functions

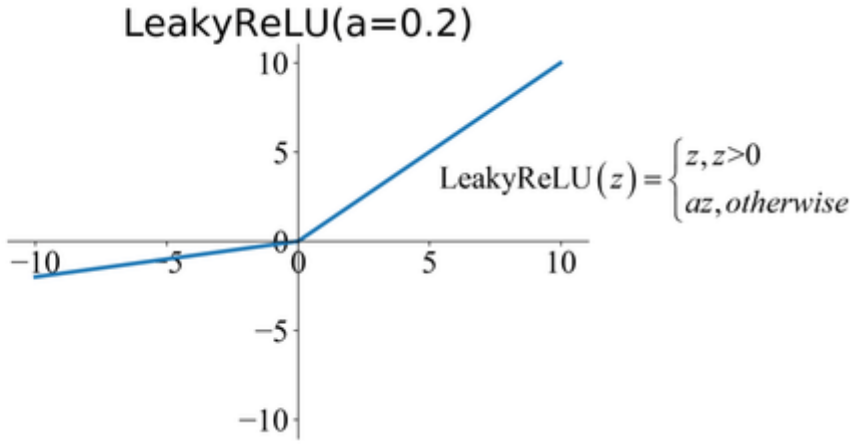
Rectified linear unit (ReLU):

- It has become the **default activation function** for many types of neural networks, because a model that uses **it is easier to train** and often achieves better performance. The derivative of the rectified linear function is also **easy to calculate**. **The slope for negative values is 0.0 and the slope for positive values is 1.0. Derivative in zero is not defined** (empirically is not a problem)

Leaky version of a Rectified Linear Unit (LeakyReLU)

- ReLU is affect to the **Dying ReLU Problem**: some **ReLU Neurons essentially die** for all inputs and remain inactive no matter what input is supplied, here no gradient flows (**the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn**). **Empirically performance is similar to the ReLU**.





Why do you need non-linear functions?

Given in input X:

$z^{[1]} = W^{[1]}x + b^{[1]}$

(4,1) (4,3) (3,1) (4,1) - Dimensionality

$a^{[1]} = g^{[1]}(z^{[1]})$

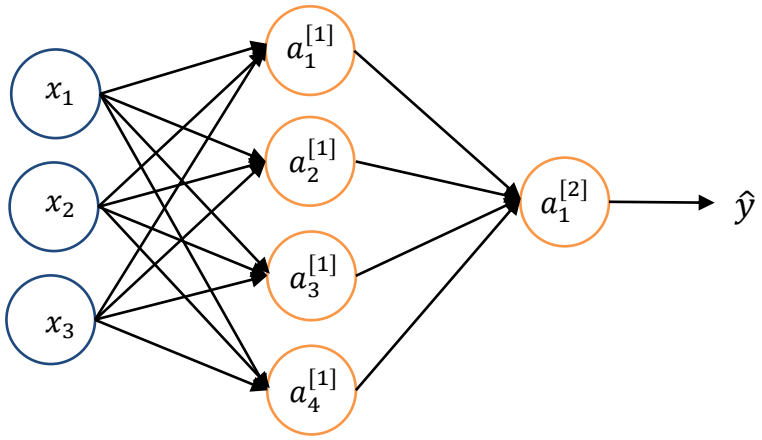
(4,1) (4,1) - Dimensionality

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

(1,1) (1,4) (4,1) (1,1) - Dimensionality

$a^{[2]} = g^{[2]}(z^{[2]})$

(1,1) (1,1) - Dimensionality



- Let's suppose $g^{[1]}(z) = g^{[2]}(z) = z$ (identity function or linear activation function) then:
- $a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} = (W^{[2]}W^{[1]})x + (W^{[2]}b^{[1]} + b^{[2]})$
- Let's define $W' = W^{[2]}W^{[1]}$ and $b' = (W^{[2]}b^{[1]} + b^{[2]})$
- $a^{[2]} = W'x + b'$ (Thus, if you are using identity activation functions, the output neural networks is linear combination of the input)
- Linear activation is often useless, unless in the last layer of a neural network when you are working in a regression task.
- Let's try again by yourself (use linear activation functions): https://lecture-demo.ira.uka.de/neural-network-demo/?preset=Binary%20Classifier%20for%20XOR

Loss Function

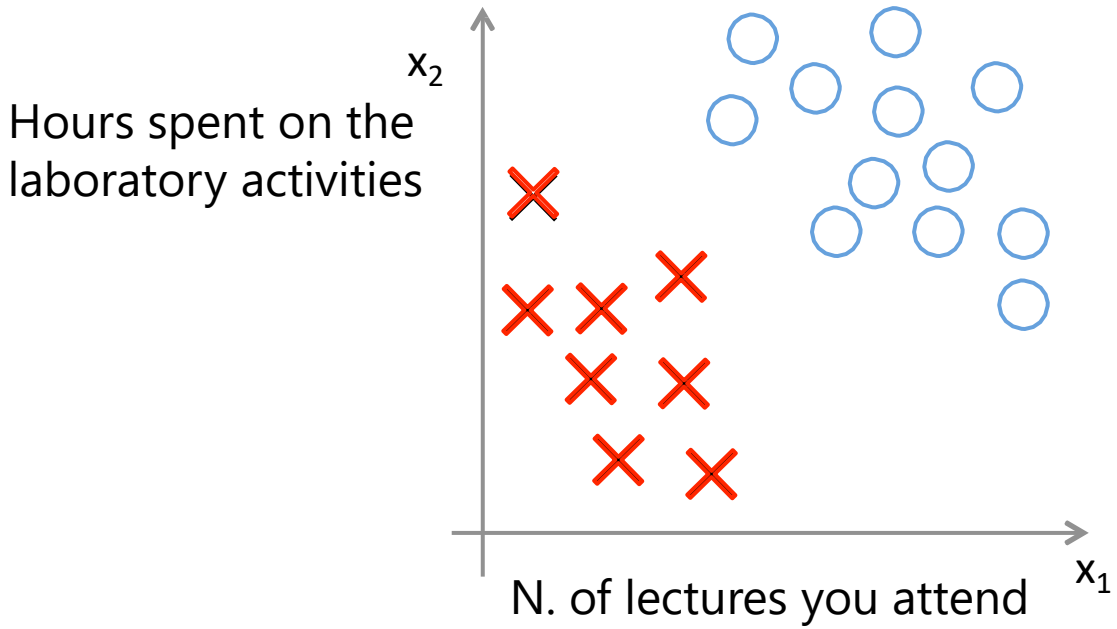
Example Problem: Will I Pass this Class?

Let's start with a simple two feature model:

x_1 = Number of lectures you attend

x_2 = Hours spent on the laboratories activities

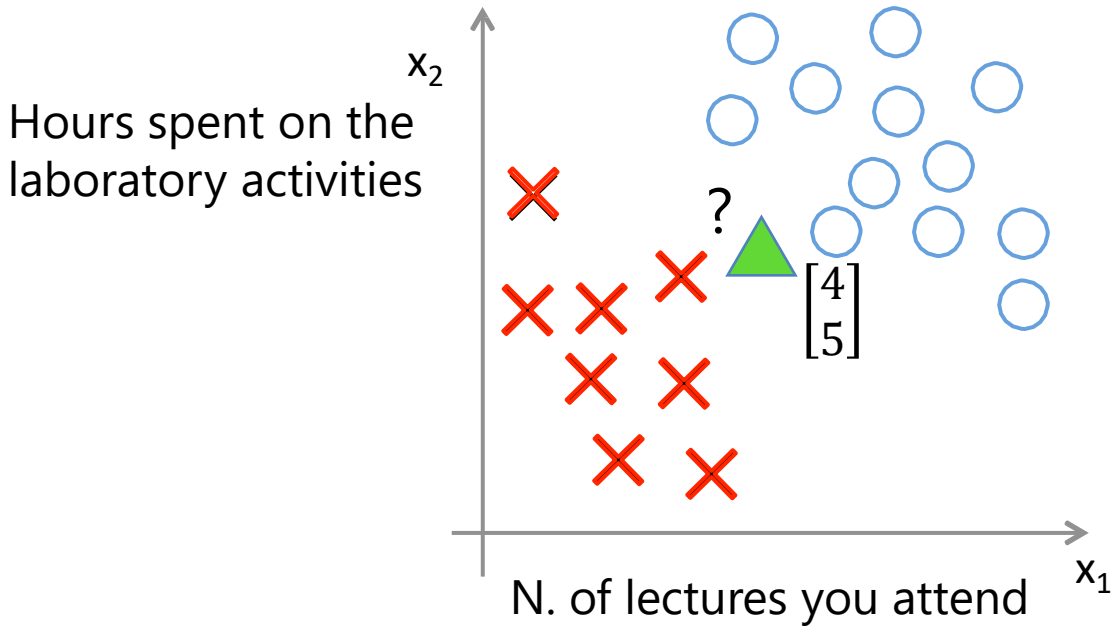
Example Problem: Will I Pass this Class?



O corresponds to “Pass”

X corresponds to “Fail”

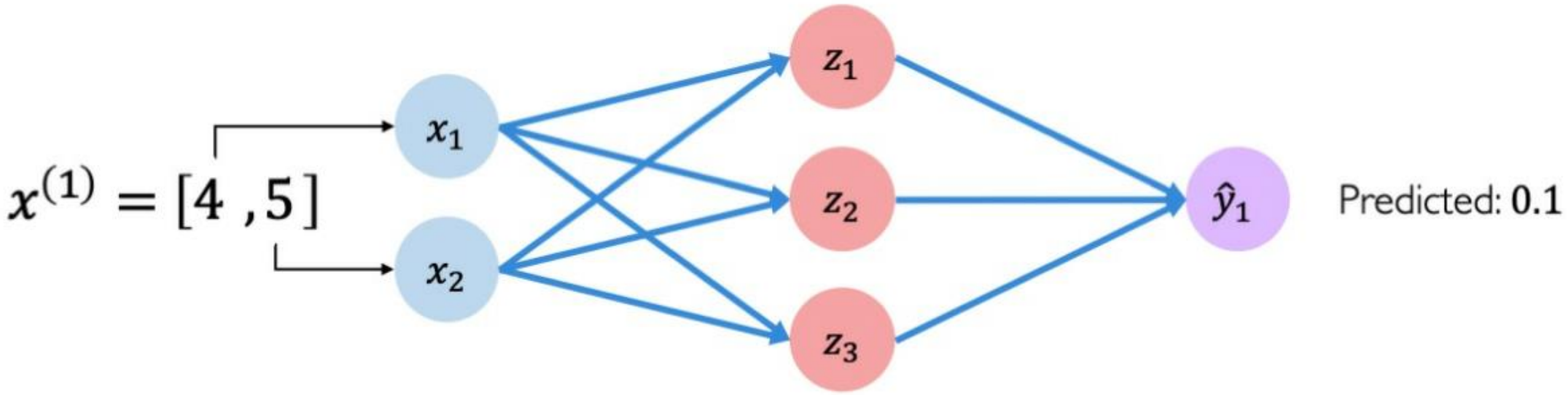
Example Problem: Will I Pass this Class?



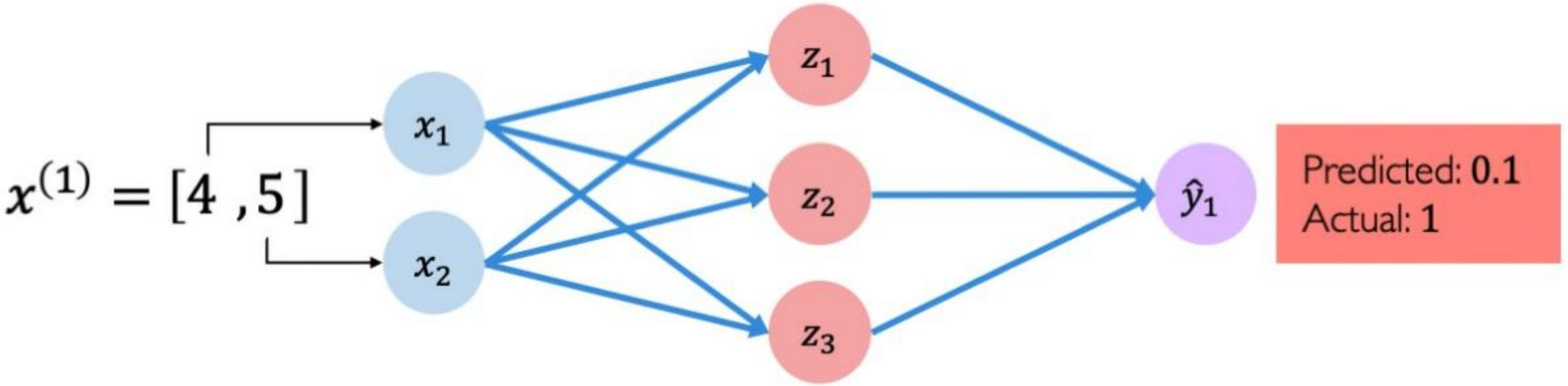
O corresponds to “Pass”

X corresponds to “Fail”

Example Problem: Will I Pass this Class?

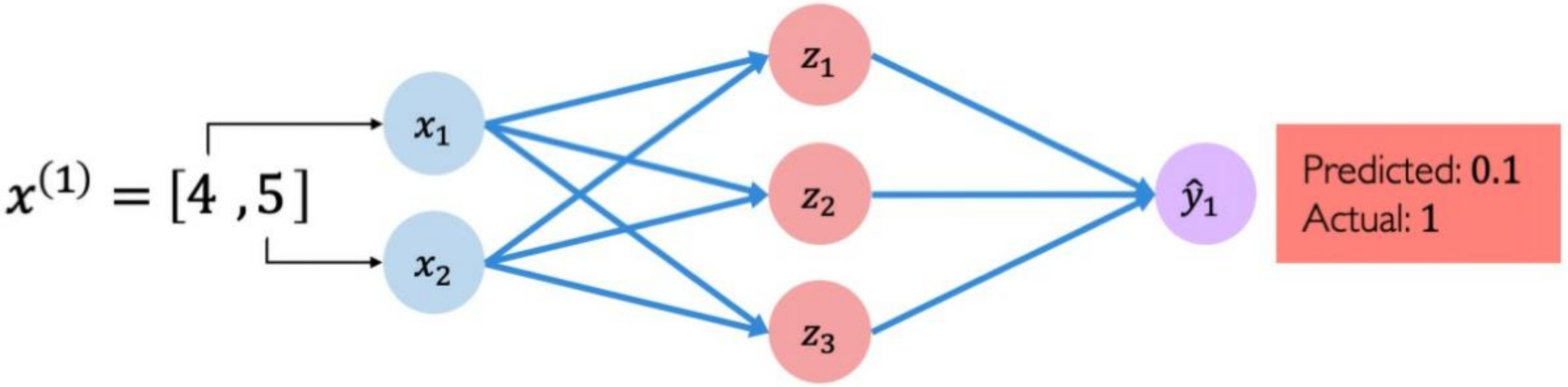


Example Problem: Will I Pass this Class?



Quantifying Loss

The Loss of our network measures the cost incurred from incorrect predictions

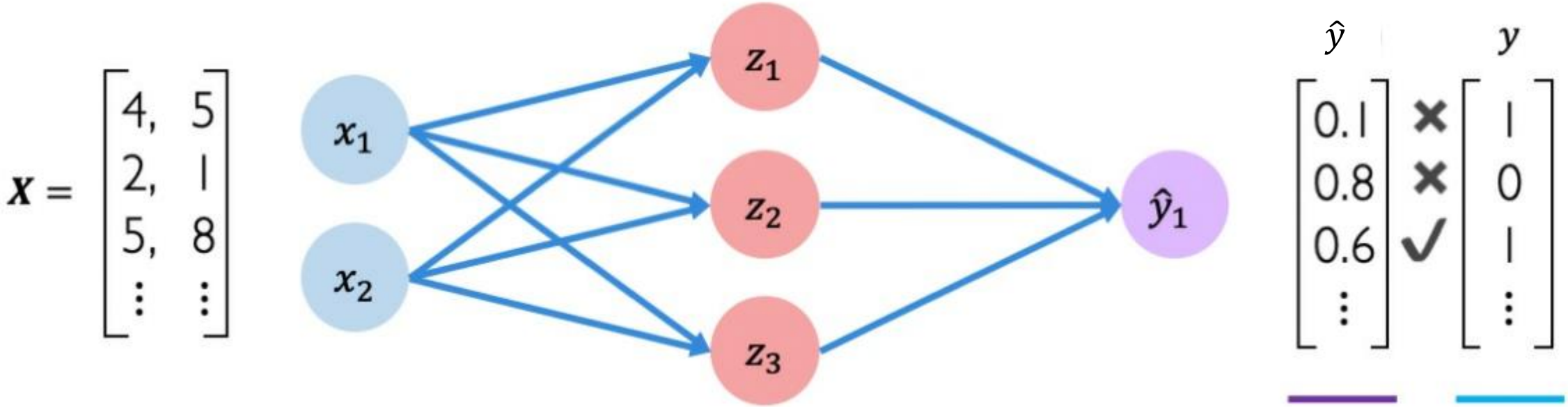


$$\mathcal{L}(\hat{y}, y)$$

Predicted Actual

Empirical Loss

The **empirical loss** measures the total loss over entire dataset

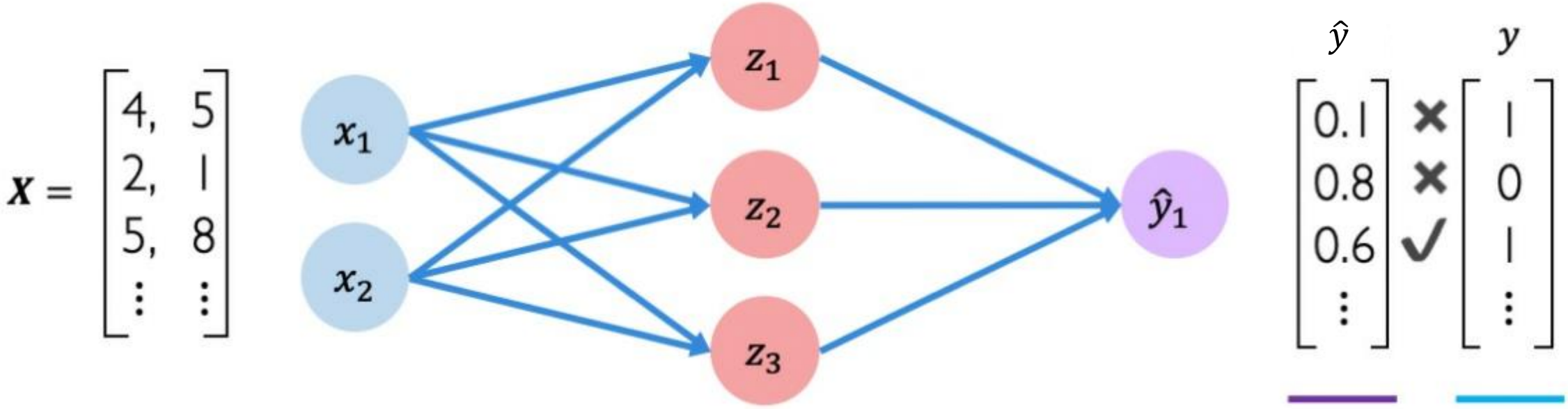


Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

Binary Cross Entropy Loss

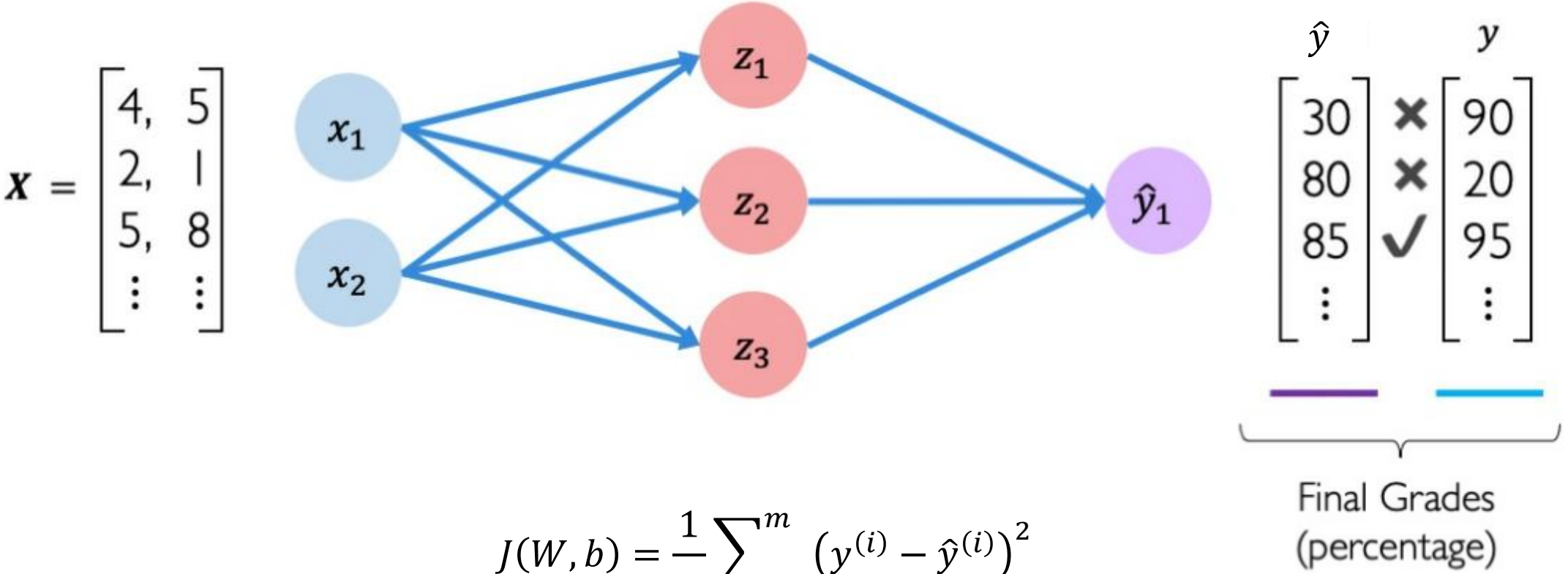
Binary Cross entropy loss can be used with models that output a probability between 0 and 1.



$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} -\left(y^{(i)} ln\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \ln\left(1 - \hat{y}^{(i)}\right)\right)$$

Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers.



$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

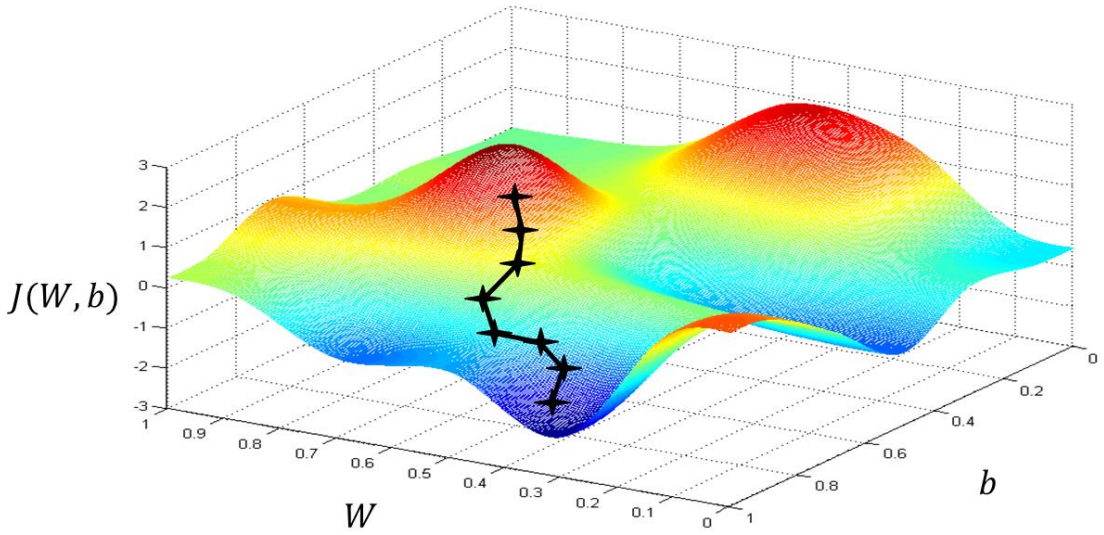
Training Neural Networks

Loss Optimization

We want to find the network weights and biases that achieve the lowest loss

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$$[w^*, b^*] = \min_{W,b} J(W,b)$$

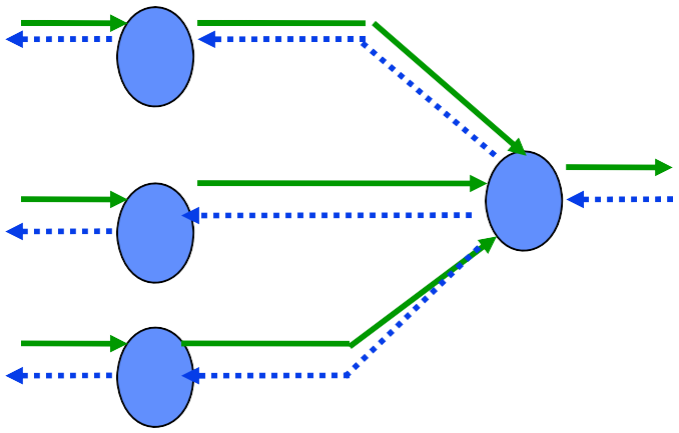


Training a Neural Networks (computing weights)

Random initialization of the weights

Backpropagation consists of the repeated application of the following two steps:

- Forward step: in this step the network is activated on one example and the error of (each neuron of) the output layer is computed.
- Backward step: in this step the network error is used for updating the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.



Network Activation
Forward Step

Error
propagation
Backward Step

Derivatives of Activation Functions

Derivatives

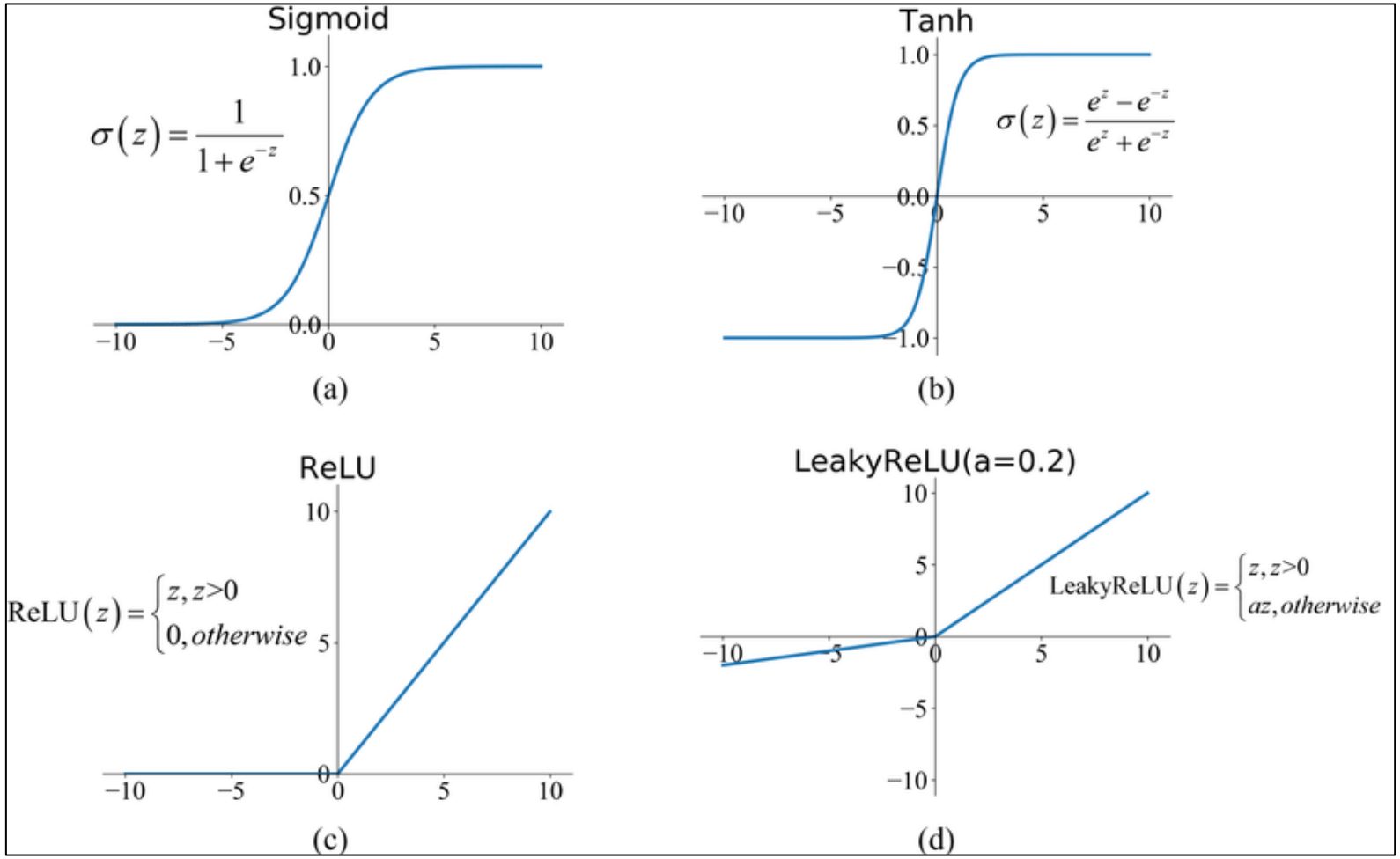
a) $\frac{d\,\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

b) $\frac{d\,\sigma(x)}{dx} = 1-\sigma(x)^2$

c) $\frac{d\,\sigma(x)}{dx} = \begin{cases} 0 & if\ z<0 \\ 1 & if\ z>0 \\ undefine\ if\ z=0^* \end{cases}$

d) $\frac{d\,\sigma(x)}{dx} = \begin{cases} a & if\ z<0 \\ 1 & if\ z>0 \\ undefine\ if\ z=0^* \end{cases}$

* No relavant in pratical scenarios



Matrix – Partial Derivatives

The derivative of a vector function (a vector whose components are functions) $\boldsymbol{y} = [y_1 \;\; y_2 \; \ldots \;\; y_m]^T$, with respect to an input vector, $\boldsymbol{x} = [x_1 \;\; x_2 \;\; \ldots \;\; x_n]^T$ is written as

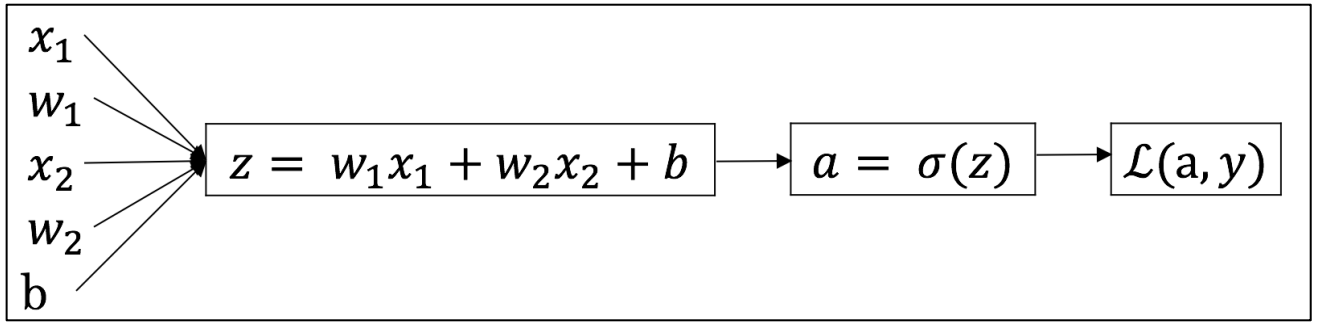
$$\frac{d\boldsymbol{y}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{dy_m}{dx_1} & \cdots & \frac{dy_m}{dx_n} \end{bmatrix}$$

The derivative of a vector function $\boldsymbol{y}$ with respect to an input vector, $\boldsymbol{x}$ whose components represent a space is known as the Jacobian Matrix.

Computing Gradients

Logistic Regression

$z = w^T x + b$
$a = \sigma(z)$
$Cost(\mathrm{a}, \mathrm{y}) = \mathcal{L}(a, y) = -(y ln(a) + (1 - y)\ln(1 - a))$



Derivatives:

$$\frac{d\mathcal{L}}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{da}\frac{da}{dz} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right)(a(1-a)) = a - y$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{dw_1} = (a-y)x_1$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{dw_2} = (a-y)x_2$$

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{da}\frac{da}{dz}\frac{dz}{db} = (a-y)$$

Gradient Descent Algorithm for Logistic Regression:

$$w_1 := w_1 - \alpha \frac{d\ J(w,b)}{dw_1}$$
$$w_2 := w_2 - \alpha \frac{d\ J(w,b)}{dw_2}$$
$$b := b - \alpha \frac{d\ J(w,b)}{db}$$

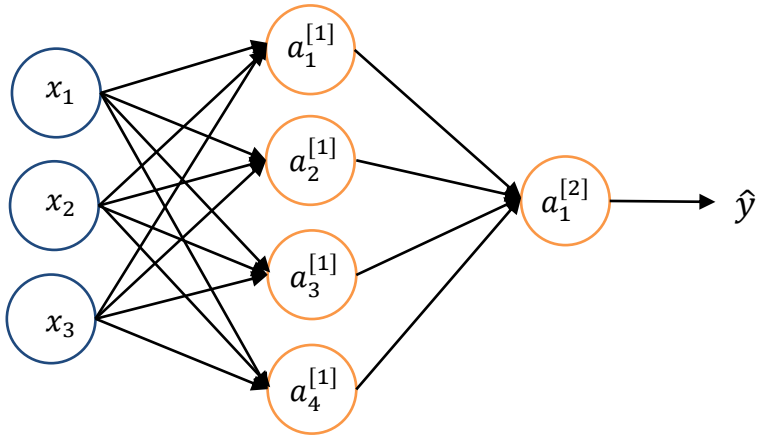
Neural Network Gradients

Let's suppose to design a Neural Network for Binary Classification

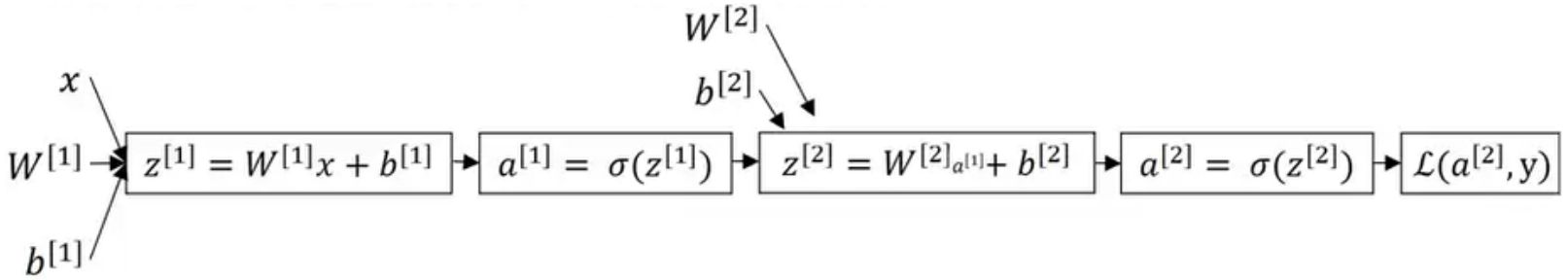
Parameters: $W^{[1]}, b^{[1]},\ W^{[2]}, b^{[2]}$

Cost Function: $\mathrm{J}\left(W^{[1]}, b^{[1]} W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$

$\mathcal{L}(\hat{y}, y) = \mathcal{L}\left(a^{[2]}, y\right) = -(y ln\left(a^{[2]}\right) + (1-y)\ln\left(1-a^{[2]}\right))$

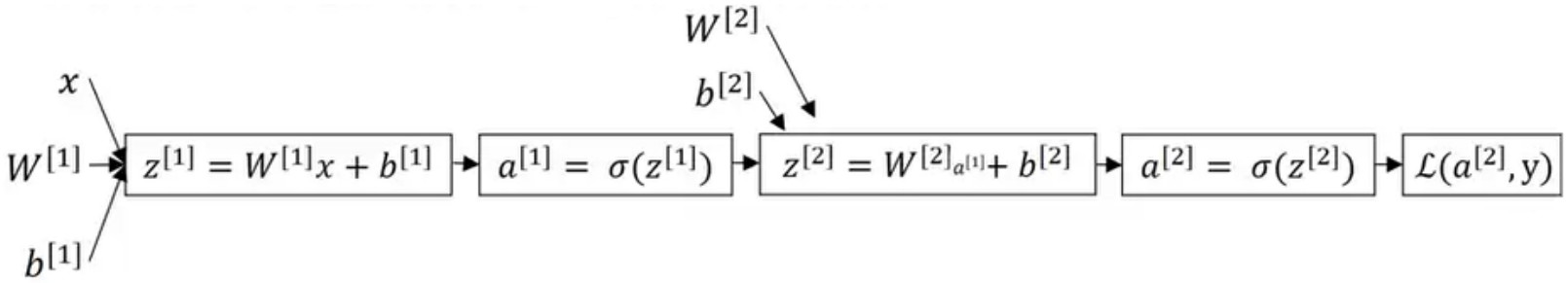


Let's start with one sample:



Remember: Learning Algorithm - Forward and Back propagation steps.

Neural Network Gradients



Forward propagation step:

$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

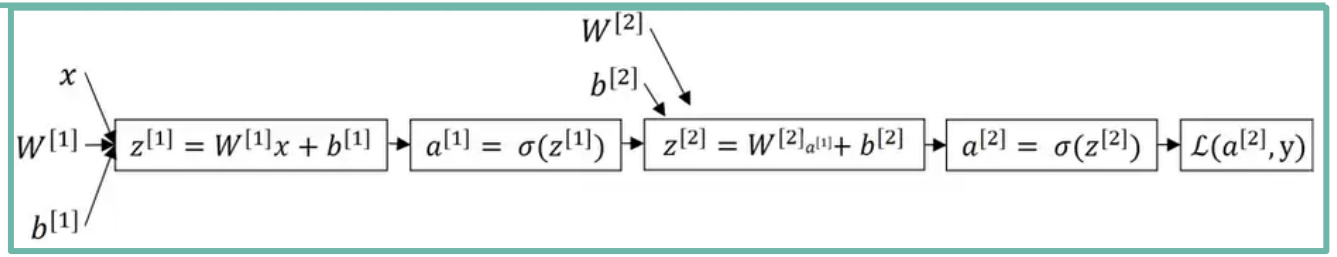
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = g^{[2]}(z^{[2]})$

Neural Network Gradients

Loss:

$$\mathcal{L}(\hat{y}, y) = \mathcal{L}(a^{[2]}, y) = -(yln(a^{[2]}) + (1-y)\ln(1-a^{[2]}))$$



Back propagation step:

$$\frac{d\mathcal{L}}{da^{[2]}} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$$

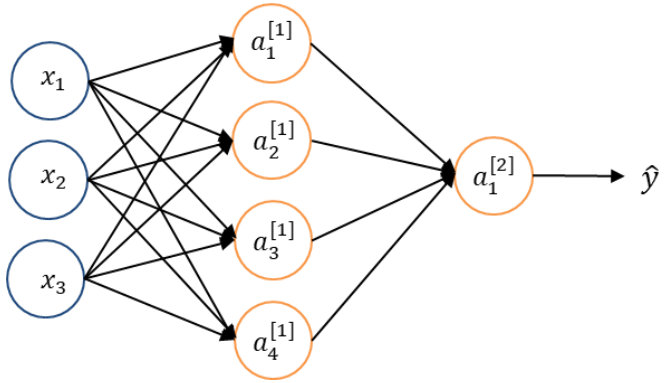
$$\frac{d\mathcal{L}}{dz^{[2]}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}} = \left(-\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}\right)\left(a^{[2]}(1-a^{[2]})\right) = a^{[2]} - y$$

$$\boldsymbol{\frac{d\mathcal{L}}{d\,W^{[2]}}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{d\,W^{[2]}} = \boldsymbol{(a^{[2]} - y)a^{[1]T}}$$

$$\boldsymbol{\frac{d\mathcal{L}}{d\,b^{[2]}}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{d\,b^{[2]}} = \boldsymbol{(a^{[2]} - y)}$$

$$\frac{d\mathcal{L}}{da^{[1]}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}} = (a^{[2]} - y)W^{[2]T} = W^{[2]T}(a^{[2]} - y)$$

$$\frac{d\mathcal{L}}{d\,z^{[1]}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}} = W^{[2]T}(a^{[2]} - y) * (a^{[1]} * (1-a^{[1]}))$$



Given in input X:

$z^{[1]} = W^{[1]}x + b^{[1]}$

(4,1) (4,3) (3,1) (4,1) - Dimensionality

$a^{[1]} = g^{[1]}(z^{[1]})$

(4,1) (4,1) - Dimensionality

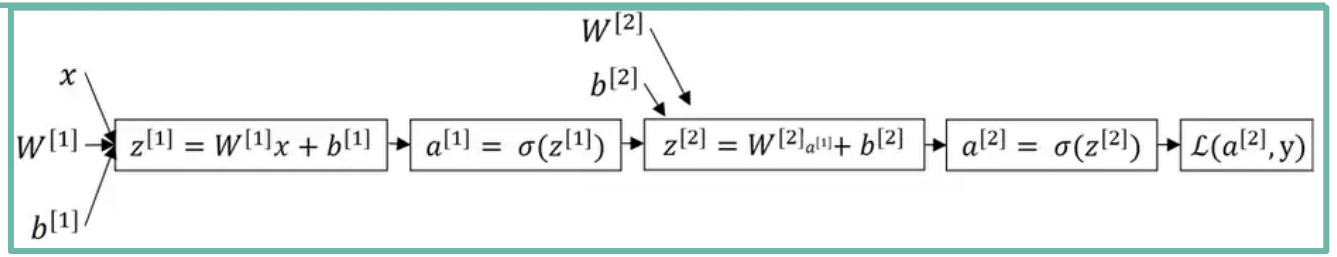
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

(1,1) (1,4) (4,1) (1,1) - Dimensionality

$a^{[2]} = g^{[2]}(z^{[2]})$

(1,1) (1,1) - Dimensionality

Neural Network Gradients



Back propagation step:

$$\frac{d\mathcal{L}}{da^{[2]}} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$$

$$\frac{d\mathcal{L}}{dz^{[2]}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}} = \left(-\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}\right)\left(a^{[2]}(1-a^{[2]})\right) = a^{[2]} - y$$

$$\boldsymbol{\frac{d\mathcal{L}}{d\,W^{[2]}}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{d\,W^{[2]}} = \boldsymbol{(a^{[2]} - y)a^{[1]T}}$$

$$\boldsymbol{\frac{d\mathcal{L}}{d\,b^{[2]}}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{d\,b^{[2]}} = \boldsymbol{(a^{[2]} - y)}$$

$$\frac{d\mathcal{L}}{da^{[1]}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}} = (a^{[2]} - y)W^{[2]T} = W^{[2]T}(a^{[2]} - y)$$

$$\frac{d\mathcal{L}}{d\,z^{[1]}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}} = W^{[2]T}(a^{[2]} - y) * (a^{[1]} * (1 - a^{[1]}))$$

$$\boldsymbol{\frac{d\mathcal{L}}{d\,W^{[1]}}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}}\frac{dz^{[1]}}{dW^{[1]}} = \frac{d\mathcal{L}}{d\,z^{[1]}}x^T$$

$$\boldsymbol{\frac{d\mathcal{L}}{d\,b^{[1]}}} = \frac{d\mathcal{L}}{da^{[2]}}\frac{da^{[2]}}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}}\frac{dz^{[1]}}{db^{[1]}} = \frac{d\mathcal{L}}{d\,z^{[1]}}$$

Gradient Descent Algorithm :

$$W^{[1]} := W^{[1]} - \alpha\frac{d\,\mathrm{J}(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]})}{dW^{[1]}}$$

….

Neural Network Gradients with *m* examples

$$\mathrm{J}\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(a^{[2](i)}, y^{(i)}\right)$$

Back propagation step with m examples:

$$\frac{d\,J\left(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]}\right)}{dW^{[2]}} = \frac{1}{m}\sum_{i=1}^{m}\frac{d\,\mathcal{L}\left(a^{[2](i)}, y^{(i)}\right)}{d\,W^{[2]}}$$

$$\frac{d\,J\left(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]}\right)}{db^{[2]}} = \frac{1}{m}\sum_{i=1}^{m}\frac{d\,\mathcal{L}\left(a^{[2](i)}, y^{(i)}\right)}{d\,b^{[2]}}$$

$$\frac{d\,J\left(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]}\right)}{dW^{[1]}} = \frac{1}{m}\sum_{i=1}^{m}\frac{d\,\mathcal{L}\left(a^{[2](i)}, y^{(i)}\right)}{d\,W^{[1]}}$$

$$\frac{d\,J\left(W^{[1]}, b^{[1]}W^{[2]}, b^{[2]}\right)}{db^{[1]}} = \frac{1}{m}\sum_{i=1}^{m}\frac{d\,\mathcal{L}\left(a^{[2](i)}, y^{(i)}\right)}{d\,b^{[1]}}$$

Gradient Descent Algorithm:

$$W^{[2]} := W^{[2]} - \alpha\frac{d\,\mathrm{J}\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right)}{dW^{[2]}}$$

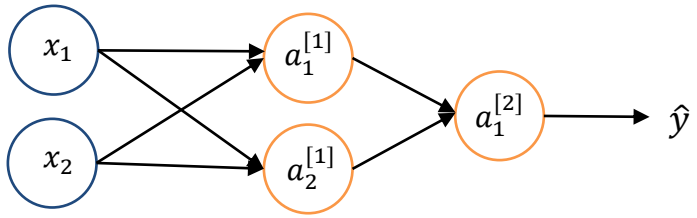
$$b^{[2]} := b^{[2]} - \alpha\frac{d\,\mathrm{J}\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right)}{db^{[2]}}$$

...

Random Initialization

What happens if you initialize weights to zero?

Let's start with this example:



Derivatives from previous slides:

$$\frac{d\mathcal{L}}{d\,W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

$$\frac{d\mathcal{L}}{d\,z^{[1]}} = W^{[2]T}(a^{[2]} - y) * (a^{[1]} * (1 - a^{[1]}))$$

$$\frac{d\mathcal{L}}{d\,W^{[1]}} = \frac{d\mathcal{L}}{d\,z^{[1]}}x^T$$

If $W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and, $W^{[2]} = [0 \quad 0]$ for any input sample in forward propagation $a_1^{[1]} = a_2^{[1]} \rightarrow \frac{d\mathcal{L}}{dW_1^{[2]}} = \frac{d\mathcal{L}}{dW_2^{[2]}} \rightarrow \frac{d\mathcal{L}}{dz_1^{[1]}} = \frac{d\mathcal{L}}{dz_2^{[1]}}$

Let's suppose $\frac{d\mathcal{L}}{dz_1^{[1]}} = \frac{d\mathcal{L}}{dz_2^{[1]}} = v$, when we compute the $\frac{d\mathcal{L}}{d\,W^{[1]}}$ we can see that:

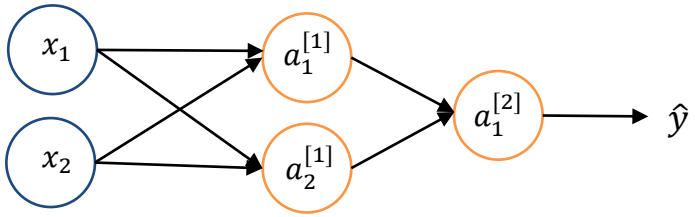
$$\frac{d\mathcal{L}}{d\,W^{[1]}} = \frac{d\mathcal{L}}{d\,z^{[1]}}x^T = \begin{bmatrix} v \\ v \end{bmatrix}[x_1 \quad x_2] = \begin{bmatrix} vx_1 & vx_2 \\ vx_1 & vx_2 \end{bmatrix}$$

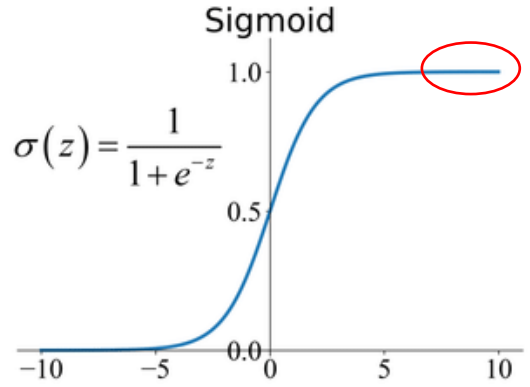
Thus, in the second forward interaction computing we will have $a_1^{[1]} = a_2^{[1]}$ and so on…. It's called "symmetry problem"

Random initialization is a solution! (Note: biases initialization to zero is not a problem)

Random Initialization

Let's start with this example:





$W^{[1]} = np.radom.randn\big((2,2)\big) * \gamma$

$b^{[1]} = np.zeros\big((2,1)\big)$

$W^{[2]} = np.radom.randn\big((1,2)\big) * \gamma$

$b^{[2]} = 0$

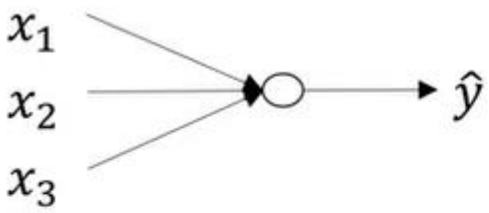
For example, $\gamma = 0.01$

The main idea is to try to keep values small!

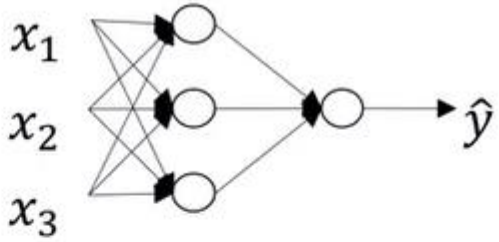
Otherwise, if you use **large weights**, for example with a **sigmoid function** in the last layer, your output values will go on the flat region of the function where **the slop or the gradient is very small**. **Gradient Descent will be very slow**.

What is a Deep Neural Network?

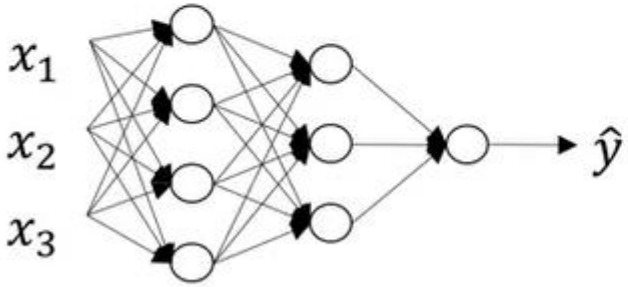
«Shallow Network»



logistic regression

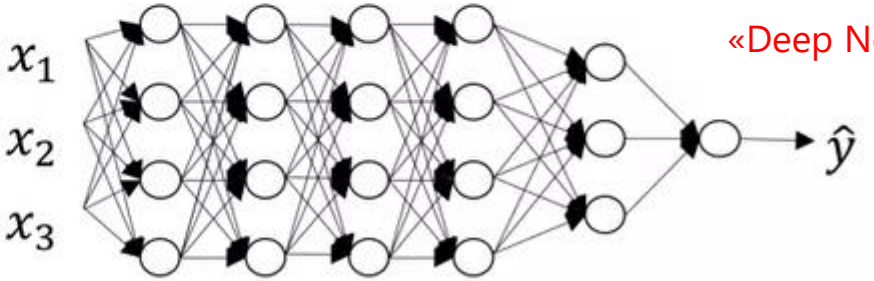


1 hidden layer



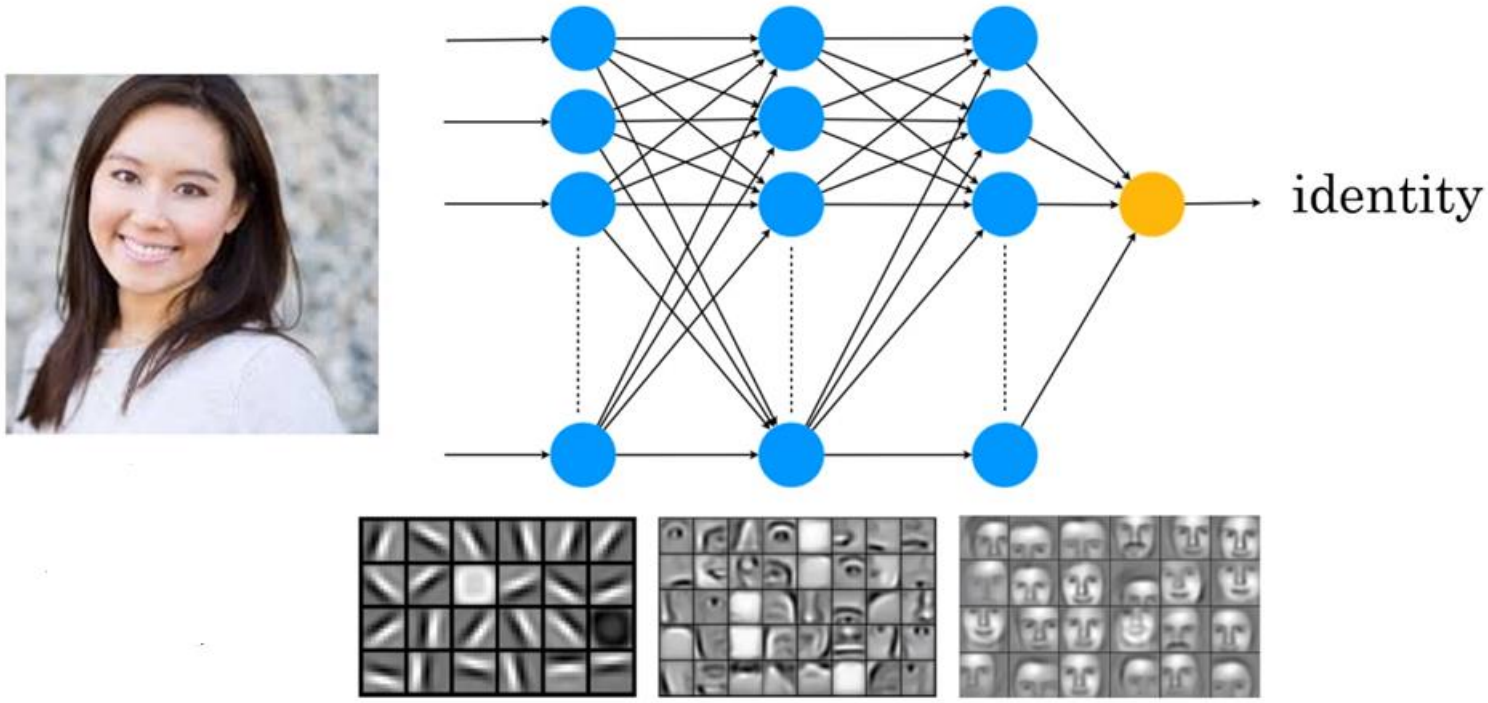
2 hidden layers

«Deep Network»



5 hidden layers

Intuition about deep representation



We will better understand this image when we are using Convolution Neural Networks.

The idea could be applied to audio context: phonemes… words… sentences

What are hyperparameters?

Parameters: $W^{[1]}, b^{[1]} W^{[2]}, b^{[2]}$....

Hyperparameters:

- Learning rate
- Number of interaction (epochs)
- Number of hidden layers
- Number of hidden units in each hidden layers
- Activation functions

Right now, this process is empirically.