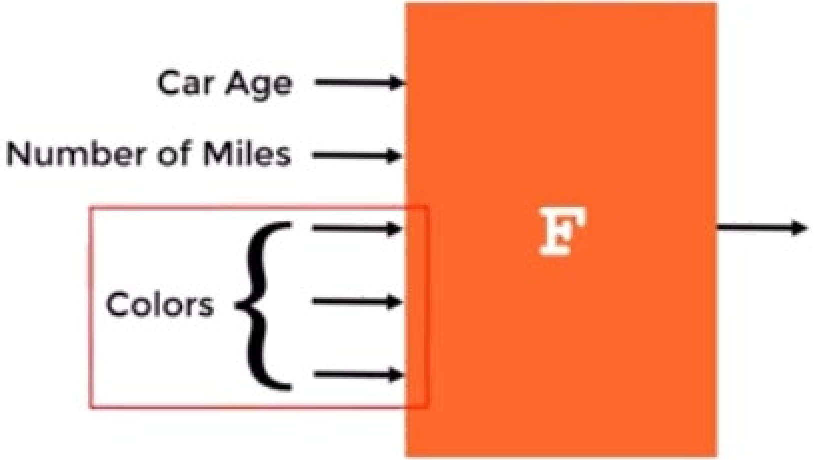
Deep Learning

SILVERIO GARCIA CORTES

UNIVERSIDAD DE OVIEDO

Problem statement, Feature matrix and Normalization

Used car value prediction problem:



AGE	MILES	COLOR
3	100000	Red
5	60000	Blue
7	120000	White

PRICE
5000
4500
3000

Feature matrix

Ground truth

Categorical feature codification: ONE SHOT

- Preserves equality distance between different classes (as opossed to 0,1,2)

Feature matrix

Ground truth

AGE	MILES	COLOR
3	100000	1,0,0
5	60000	0,0,1
7	120000	1,1,1

PRICE
5000
4500
3000

Numerical feature Normalization

- **Avoids one feature dominates over others becuase its magnitude**

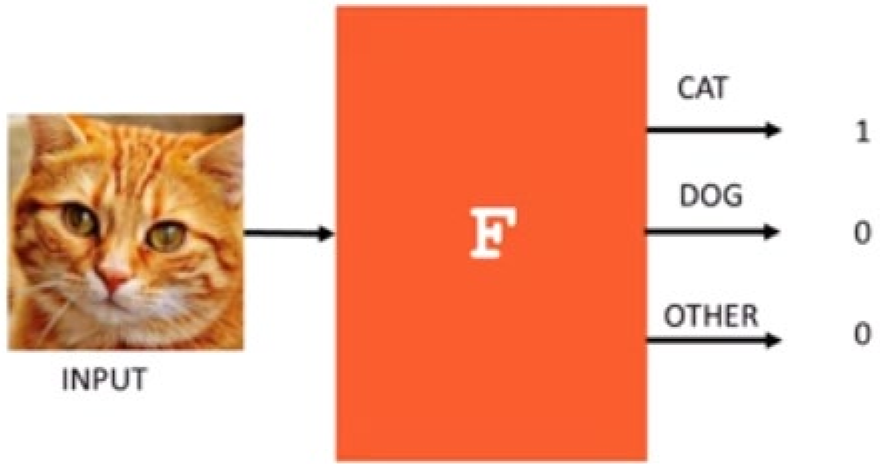
$$x_1^{scaled} = \frac{x_1 - min_{x_1}}{max_{x_1} - min_{x_1}}$$

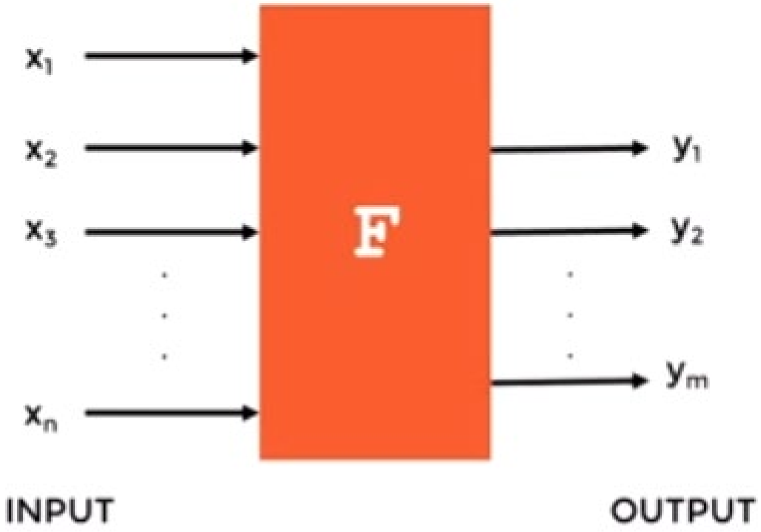
$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

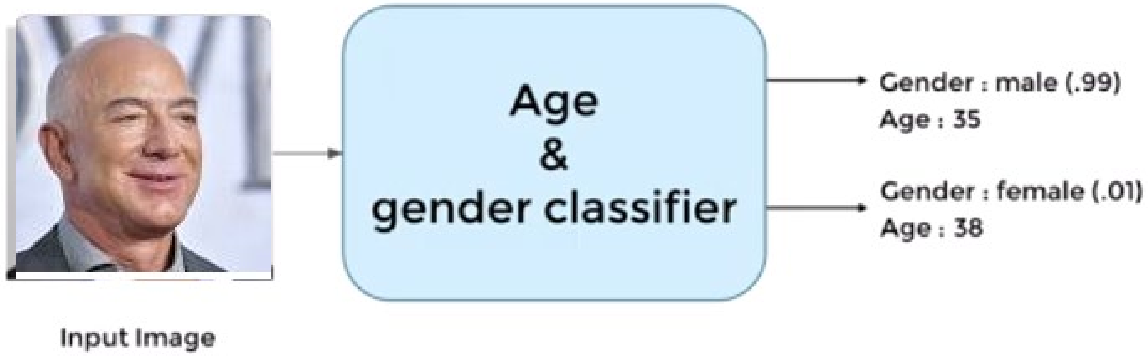
Problem statement:

- Input data should be converted to feature vectors (matrix)
- If the problem is a classification the outptut value will be also one hot coded

- Categorical features should be one got encoded
- Numerical features must me normalized
- Also problems with several outputs are posible

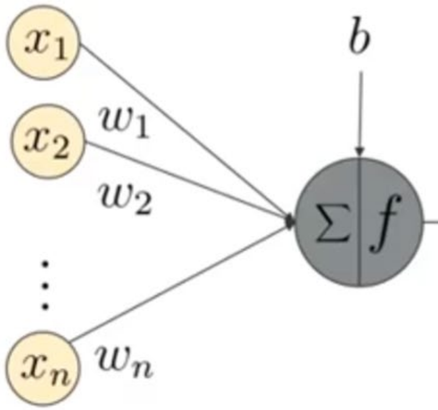




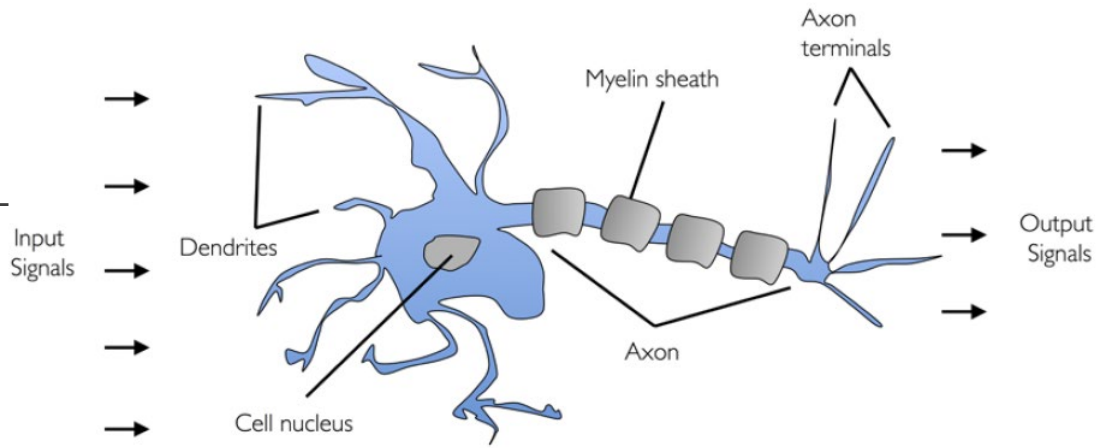


Neural Network basic component: Artificial Neuron. Perceptron

Perceptron Model



$$f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$



Input

- An artificial neuron artificial is a model of Machine Learning inspired in biological neural networks.
- It can model relationships between inputs and outputs
- The Perceptron is the unit of neural networks and was initially built as a binary classifier similar to a bio neuror

$$z = w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_m \cdot x_m = \sum_{j=0}^{m} \mathbf{x}_j \cdot \mathbf{w}_j$$

$$= \mathbf{w}^T \cdot \mathbf{x}$$

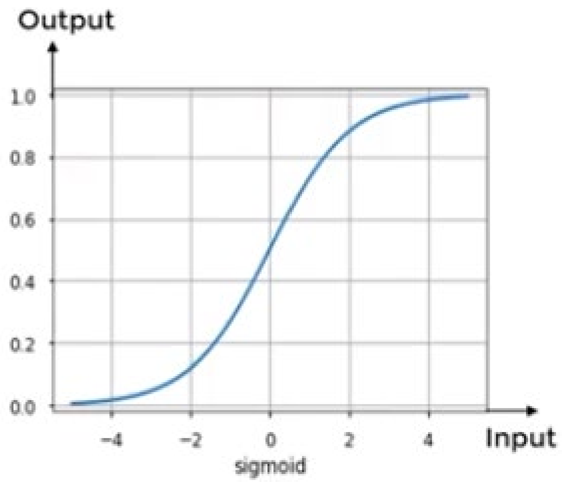
Activation Function Example

$$\phi(\mathbf{z}) = \begin{cases} 1 & \mathbf{z} \geq 0 \\ -1 & \mathbf{z} < 0 \end{cases}$$

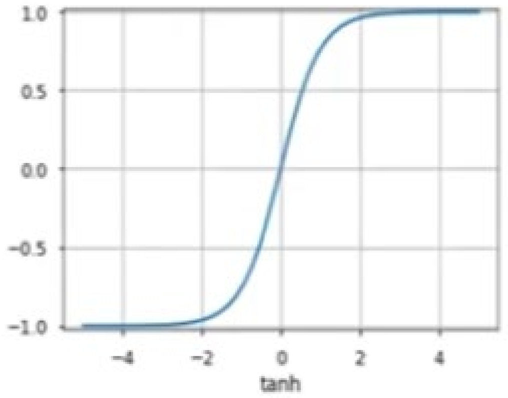
- A perceptron receives a number of inputs (x_i) which are combined with weights and bias (one per neuron) to obtain the “net input” z.
- Finally an activation function is applied to the “z” value to obtain the final output $\phi(z)$.

Activation fuctions

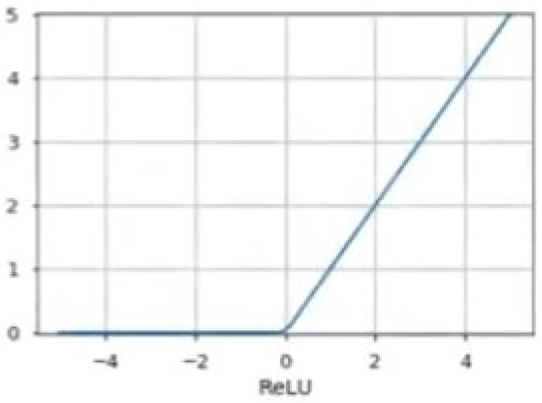
Commonly activation functions are non-linear. In this way the NN can model non linear relations between inputs and oputputs. Some common activation functions are:



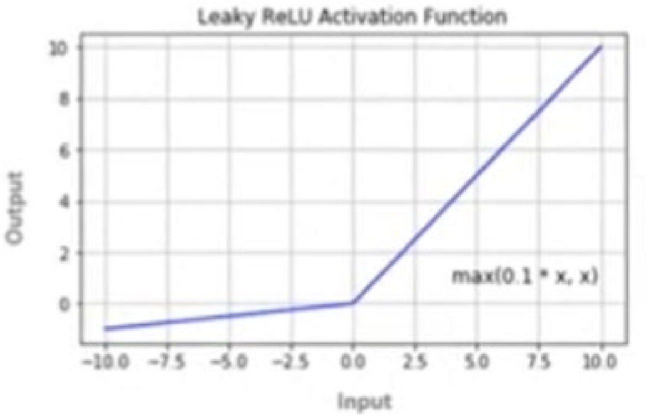
Sigmoid: Logistic Regression:
[-inf,inf]-> [0,1]



tanh: Logistic Regression:
[-inf,inf]-> [-1,1]



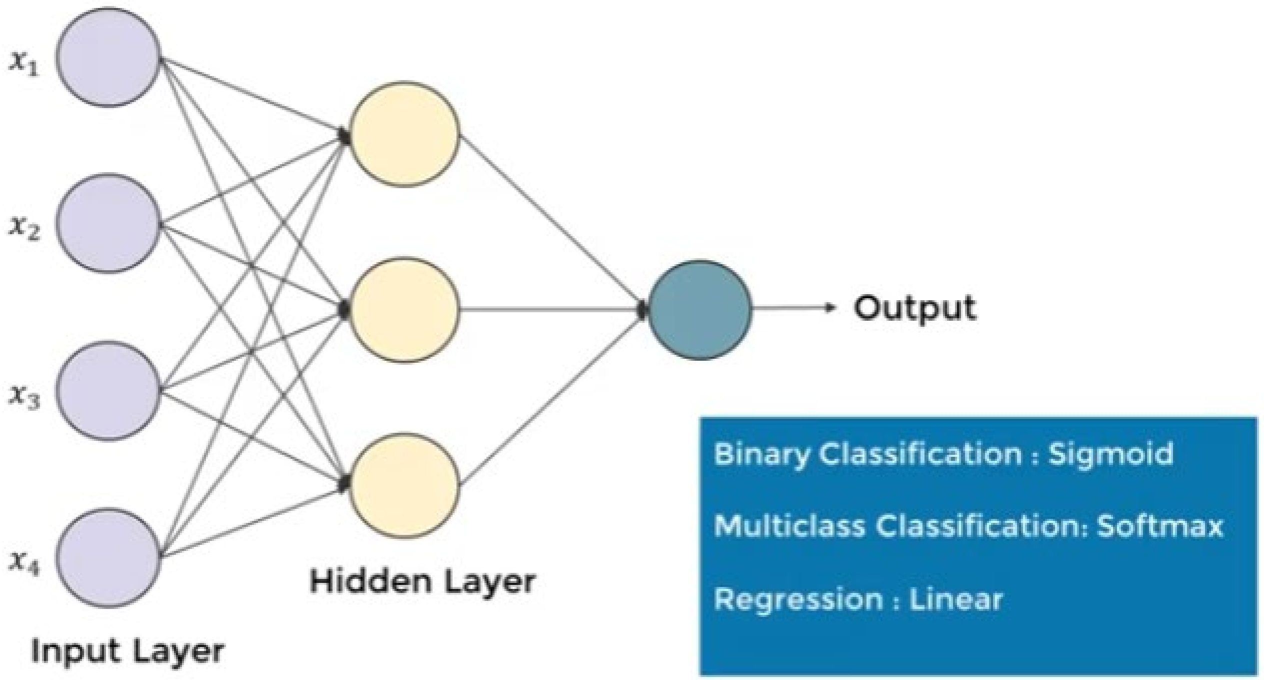
ReLU: Rectified linear unit:
[-inf,inf]-> [0,1]



Leaky ReLU:
Piecewise
Avoids dying gradient

From Perceptron to Neural Networks (NN)

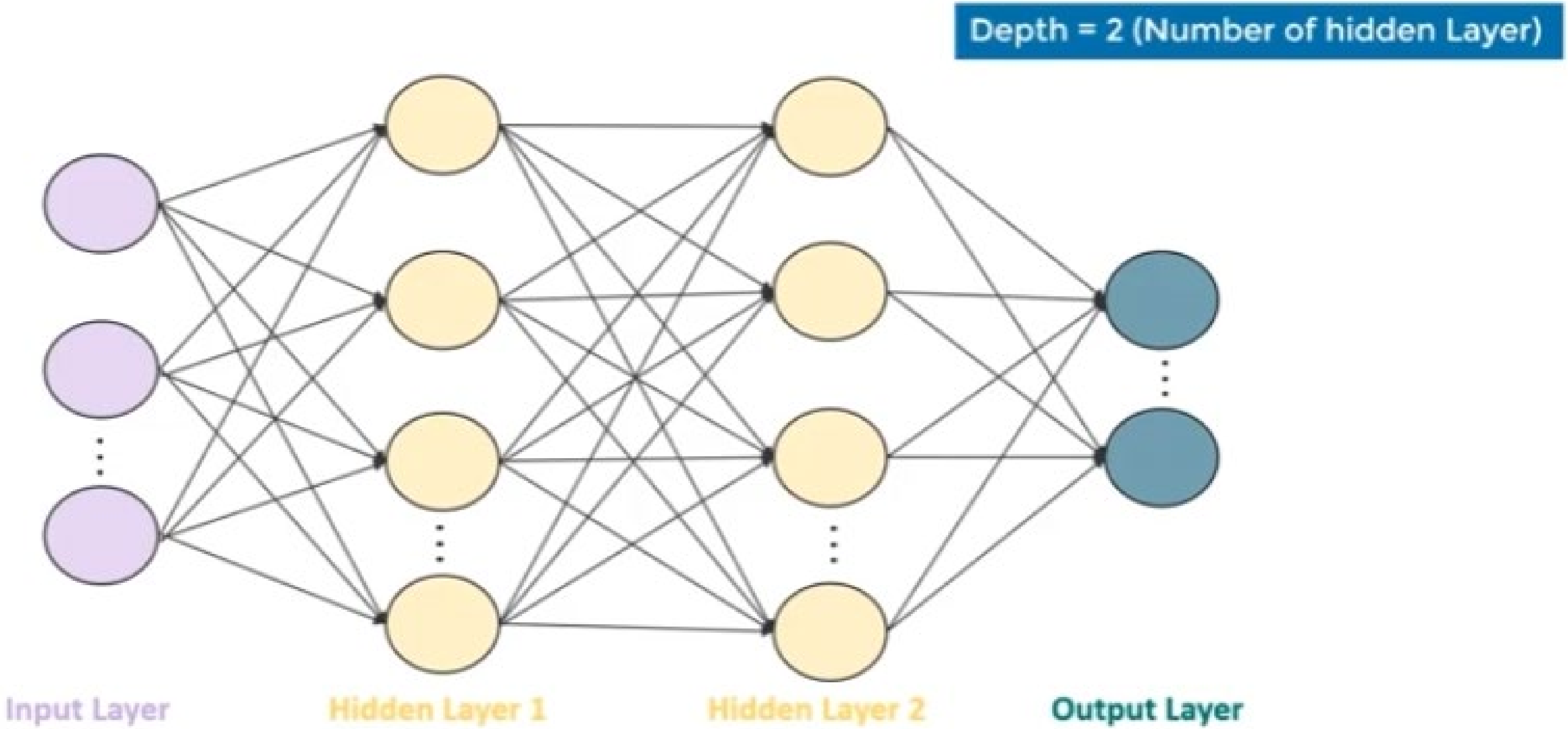
- A perceptron is combined in layers.
- Each neuron receives all the inputs (features)
- And has its own weights and bias
- The goal of the NN is to output the correct value for the input features (x1,x2,x3,..)



Different problems need different activation functions in the output layer

NN depth

A network can model more complex relationships the deeper it is.



How does a NN learns? Error and Cost function

- At the ouput of the NN the error between the predicted value and the ground truth value is calculated
- The total mean error is calculated. For example Means Square error (no outliers) or Meand absolute error (tolerate better outliers)

Objective Function : Cost Function

Error

$$e_i = y_i - \hat{y}_i$$

MEAN SQUARED ERROR

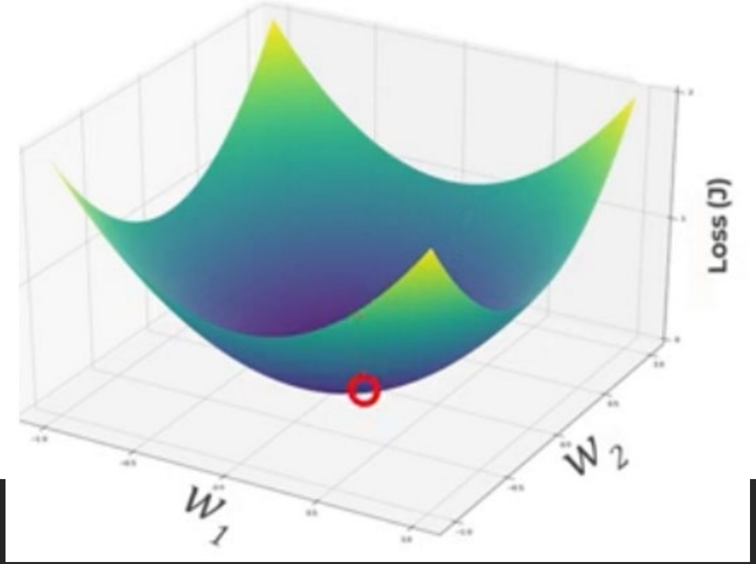
$$L = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{y}_i - \hat{\mathbf{y}}_i||^2$$

MEAN ABSOLUTE ERROR

$$L = \frac{1}{n}\sum_{i=1}^{n} |\mathbf{y}_i - \hat{\mathbf{y}}_i|$$

Average error per data point

Total mean error must be minimized
It matematically depends on weights and bias of each neuron



How does a NN learns? Gradient descent and Backpropagation

Corrections for weights can be calculated if we know the slope of the cost function

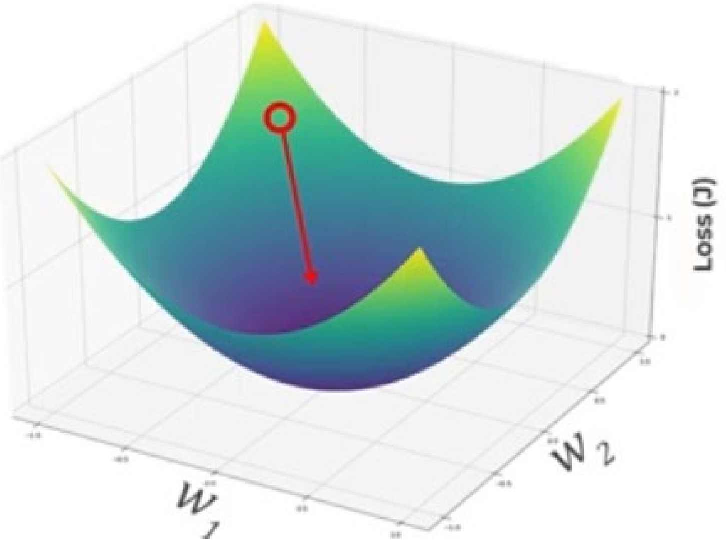
The slope of the cost function is the gradient.

We will proceed iteratively correcting the weights

We modulate the gradient with a factor called **learning rate** to avoid big steps

To calculate the gradients we Will use the chain rule:

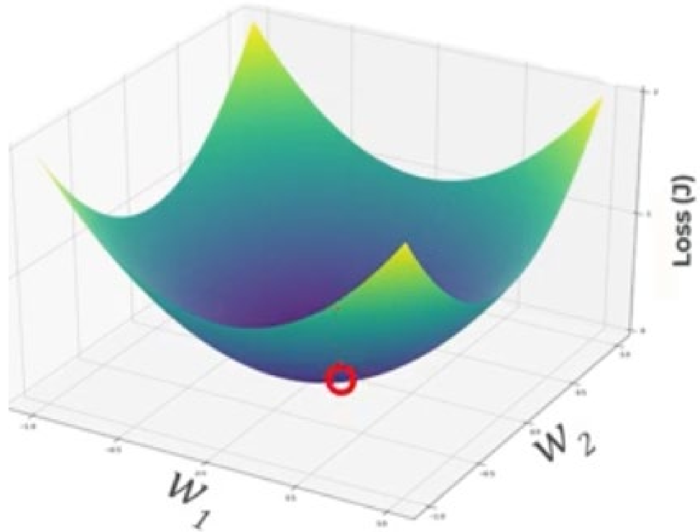
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \times \frac{\partial y}{\partial x}$$



$$w_1^{(1)} = w_1^{(0)} - \lambda \frac{\partial J}{\partial w_1}$$

$$w_2^{(1)} = w_2^{(0)} - \lambda \frac{\partial J}{\partial w_2}$$

Learning Rate



$$w_1^{(t+1)} = w_1^{(t)} - \lambda \frac{\partial J}{\partial w_1}$$

$$w_2^{(t+1)} = w_2^{(t)} - \lambda \frac{\partial J}{\partial w_2}$$

Backpropagation

1. Start with random weights.
2. Select a small batch of images from the traning set.
3. Using forward propagation, calculate loss for the current weights.
4. Calculate gradients using back propagation.
5. Perform Gradient Descent to update weights.
6. Repeat until convergence.

Multilayer Perceptron (MLP)

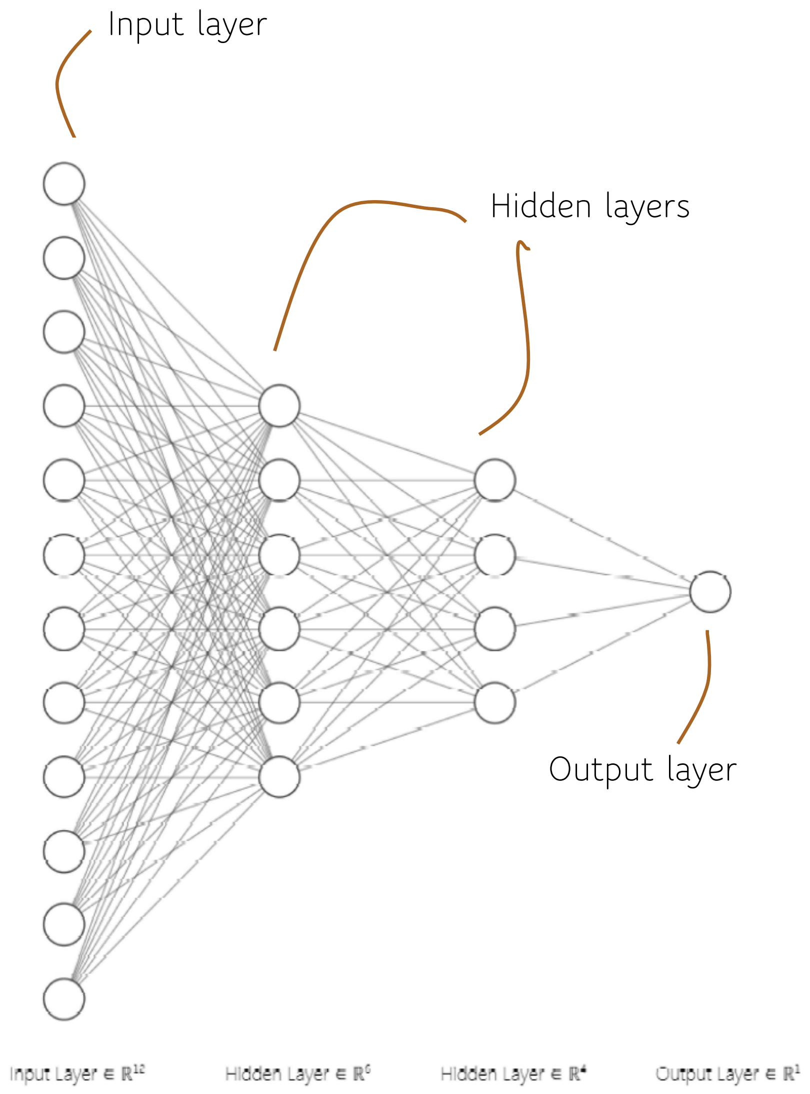
MLP (Multilayer Perceptron) is compose of dense layers where each neuron of one layer is connected with all the neurons from the previous one and the following one

The MLP needs to have at least one hidden layer different from the input and output layers

Activations functions introduce the capacity to model non linear relations in the NN.

Non linear activations are needed to model non linear relations between features and outputs

It has been shown that certain activation functions with at least one hidden layer can approximate any function if the number of neurons is high enough



NN´s are then Universal aproximators

TensorFlow Playground

- Tensorflow Playground