COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

Deep Learning Introduction

Rodrigo Serna Pérez
January 2022

Hi guys! I'm Rodrigo

I also studied in ICAI!!

- Communications Engineering Degree
- Master in Communications Engineering

I worked two years as a researcher about Deep Learning and Natural Language Processing in Altran (now Capgimini)

Currently, **I am a data scientist in BBVA**. I work in Client Solutions Area. I work in projects related to areas such as marketing and customer relationship model.

Mail address: rodser@alu.icai.comillas.edu

Linkedin: www.linkedin.com/in/rodrigosernaperez/

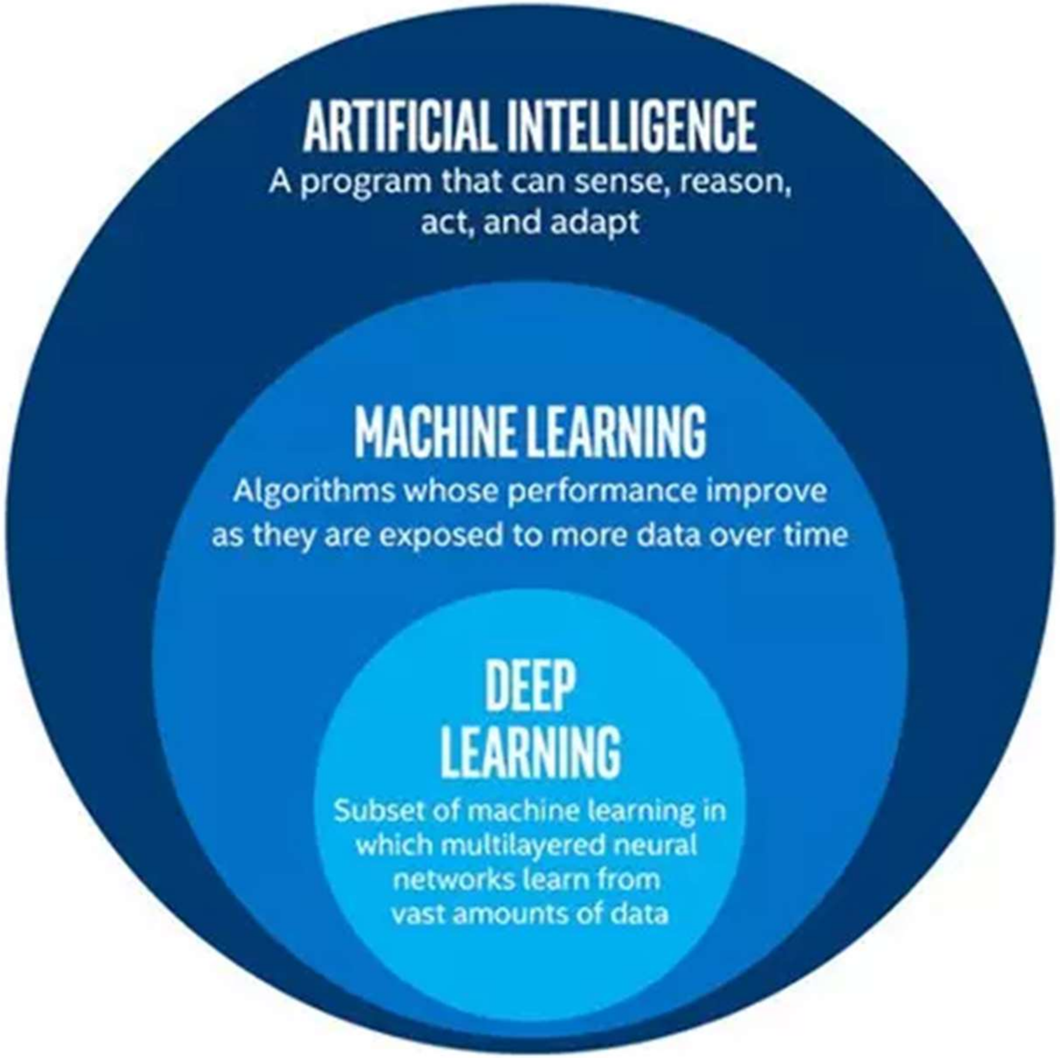
Today we'll talk about …

1. Introduction
2. Perceptron
3. Multilayer Perceptron
4. Learning Process
5. Modifications of Training Algorithms

1

Introduction

Deep Learning in AI



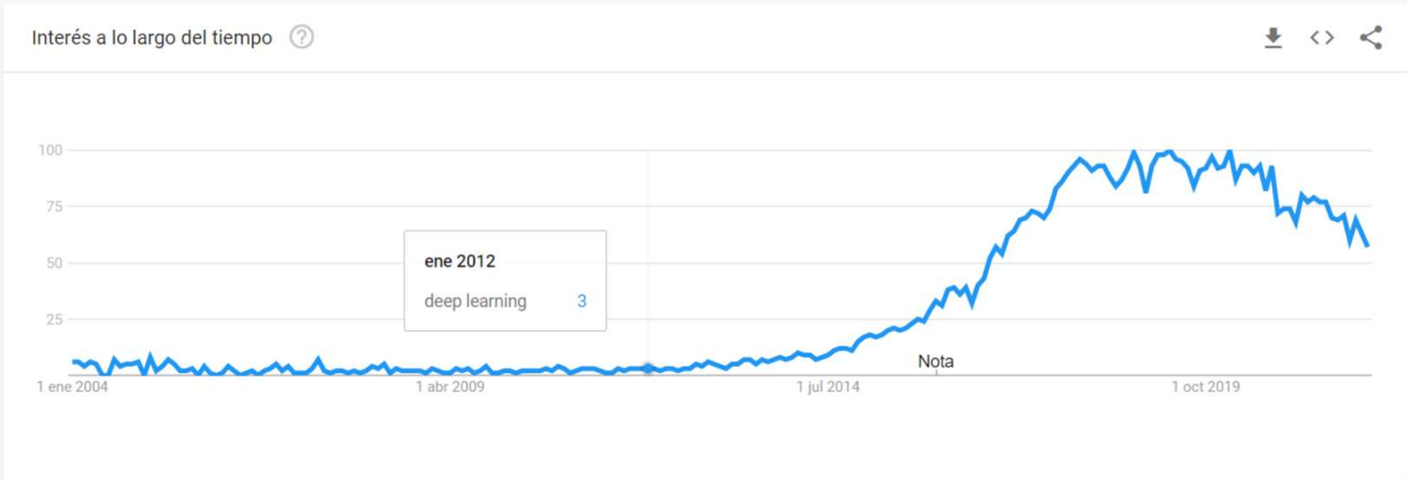
History

- Neural Networks have gained popularity during the last years, but they have a very log history: the proposal of the **Perceptron by Frank Rosenblatt comes from 1957**!!
- Researchers tried to emulate how humans learn, so they developed algorithms that “look like” human’s brain. This is why we call deep learning algorithm **Neural Networks**.
- 1998: **Gradient Descent** is proposed for training neural networks.
- Although a lot of interest was put into neural networks research, they didn’t find their place mainly because of the **hardware limitations**. Support Vector Machines (SVMs) became much more popular during years.

History

- 2012: **AlexNet wins ImageNet competition**. GPUs had enough power to train huge neural networks that could solve difficult problems such as Image Classification.

Google Searches for “Deep Learning”



Why Deep Learning?

- Very powerful when dealing with very **high dimensionality problems**: image processing, text processing …
- They are algorithms that **can be very easily parallelized**: GPUs and TPUs are a great help.
- **Extremely versatile**: can deal with continuous and categorical variables, temporal and non temporal, words, images, classification, regression, multitarget, unsupervised learning, generative models …
- **Transfer Learning**: once we train a model on a task, we can reuse that knowledge in similar ones.
- Can be trained on **huge datasets** without strong hardware limitations.
- It is the **state of the art for all AI fields** at the same time!!!

Why Not Deep Learning?

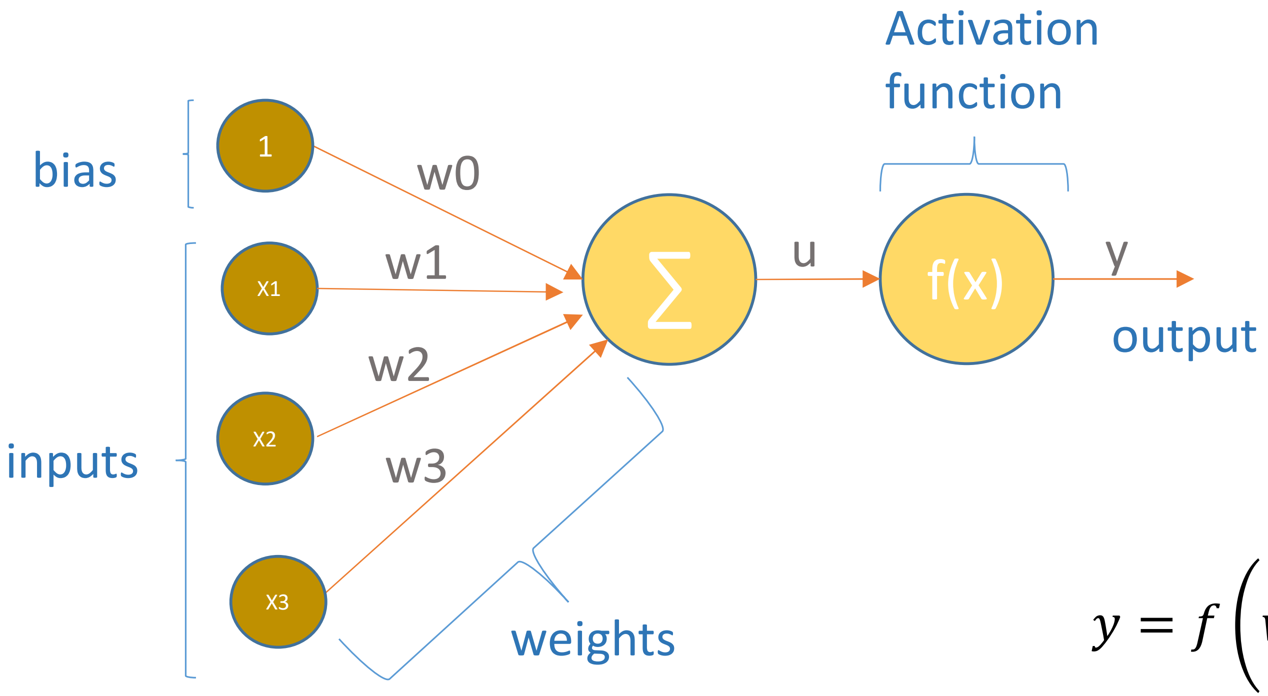
- **Lack of interpretability**
  - Getting insights could be too difficult (For example, “which are the most important input variables?”)
  - Fairness problems such as sex and race bias are complex to resolve.
- Deep Learning algorithms typically **need more data that classical machine learning algorithms.**
- **GPUs / TPUs are expensive hardware**, we do not always have access to them.

2

Perceptron

Perceptron

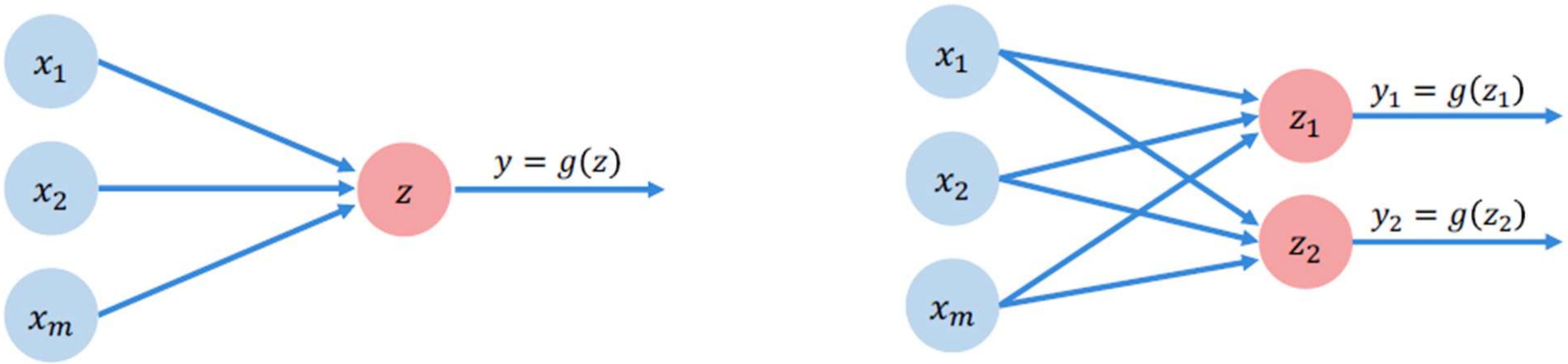
A perceptron is a linear supervised learning algorithm.



$$y = f\left(w_0 + \sum_i x_i * w_i\right)$$

Perceptron

Perceptron can be generalized to a multitarget problem.



Activation functions

The **activation function** will be chosen according to the target we are looking for.

Activation Function	Equation	Target
Linear	$f(x) = x$	Regression (Equivalent to linear regression)
Sigmoid	$f(x) = \dfrac{1}{1 + e^{-x}}$	Binary classification (Equivalent to linear separation of two classes)

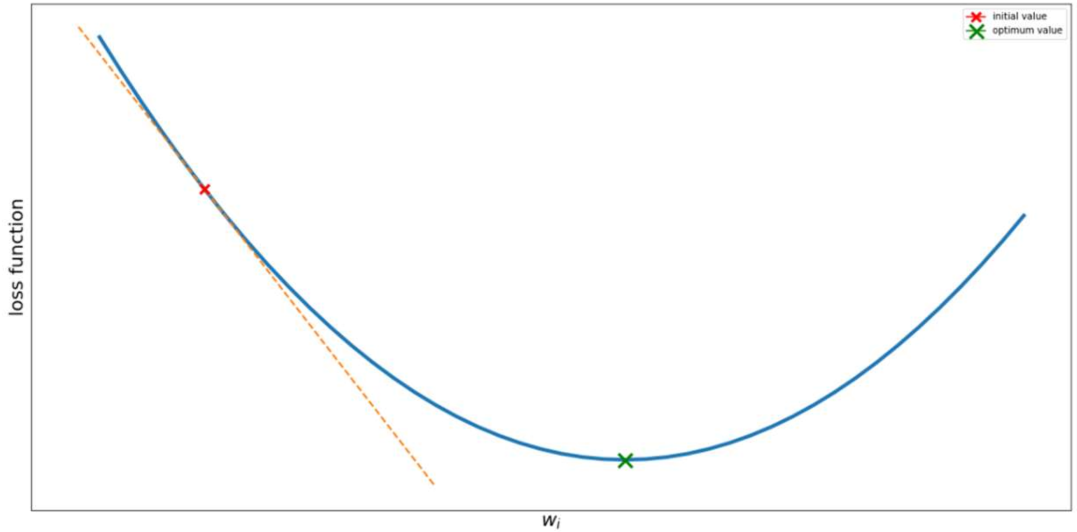
Learning weights

- The weights for the neural network are initially set at random.
- Using training data, we compute the prediction with the given inputs and then **compare it with the expected target**.
- Weights will be updated in order to minimize a **lossing function**.

Loss function	Equation	Target
Mean Squared Error	$loss(y, y_{pred}) = (y - y_{pred})^2$	Regression
Binary Entropy	$loss(y, y_{pred}) = -(y * \log(y_{pred}) + (1 - y) * \log(1 - y_{pred})$	Binary classification

Learning algorithm

- We are looking for the value for w_i that minimizes the loss function.



- **If the slope is negative**, we must “add something” to the current weight.
- **If the slope is positive**, we must “subtract something” to the current weight.

Stochastic Gradient Descent

- The weights will be updated in the direction in which the scope is decreasing.

STOCHASTIC GRADIENT DESCENT ALGORITHM

$$w_i^k = w_i^{k-1} - \alpha \frac{\partial loss}{\partial w_i}$$

$\alpha \equiv learning\ rate$

Non trainable parameter that must be manually chosen.

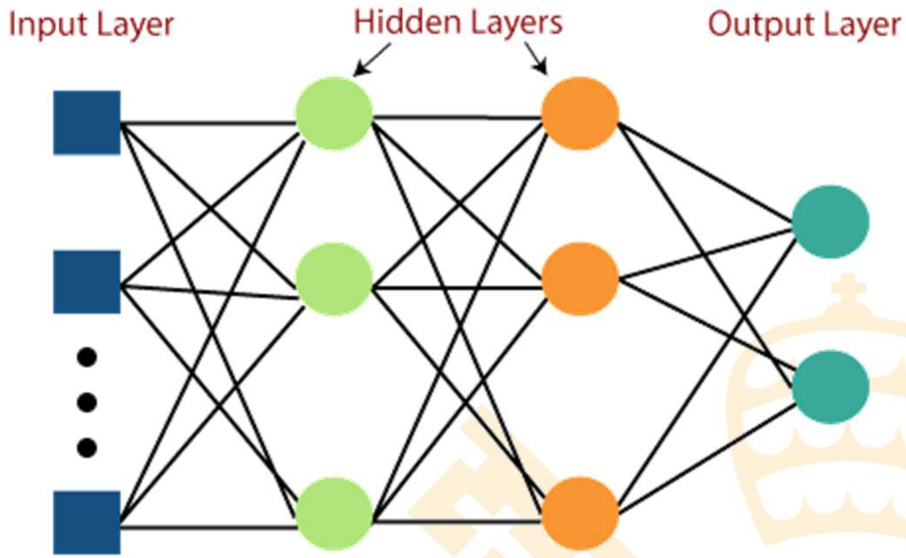
- All the weights in the network are updated once for each one of the samples in the training dataset.
- Example: Implementation in a sheet

3

Multilayer Perceptron

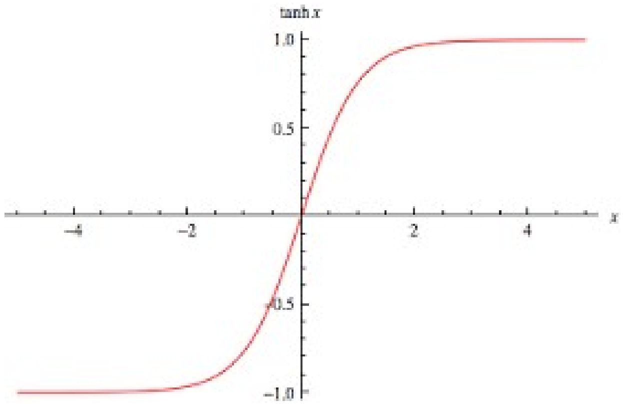
Multilayer Perceptron

- A simple perceptron is only capable of **learning linear functions**. If we want to estimate a more complex function, we need to add hidden layers.
- Instead of having one perceptron, there are N perceptrons that work in parallel. This is called a **hidden layer.**
- The output of a layer can be another hidden layer or a final layer.



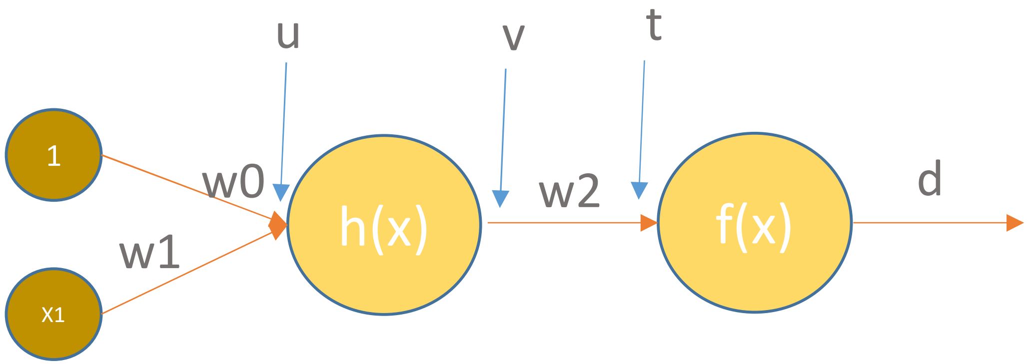
Multilayer Perceptron

- If the number of layers is high enough, a multilayer perceptron **can aproximate any function**.
- Number and size of hidden layers must be manually chosen (not optimized during training). They can be chosen through **cross validation.**
- Most typical used function as the activation function in hidden units is $y = \tanh(x) = \frac{\exp(-x) - \mathrm{ex}\ \ (x)}{\exp(-x) + \mathrm{ex}\ \ (x)}$



Multilayer Perceptron Training

- We can still use the scope to train the weights in the hidden layer. The **chain rule** is very useful in this step.



BACK PROPAGATION

$$\Delta w_2 = -\alpha * \frac{\partial e}{\partial w_2} = -\alpha * \frac{\partial e}{\partial d} * \frac{\partial d}{\partial t} * \frac{\partial t}{\partial w_2}$$

$$\Delta w_1 = -\alpha * \frac{\partial e}{\partial w_1} = -\alpha * \frac{\partial e}{\partial d} * \frac{\partial d}{\partial t} * \frac{\partial t}{\partial v} * \frac{\partial v}{\partial u} * \frac{\partial u}{\partial w_1}$$

Note that:

$$e = loss(d, y)$$

$$\frac{\partial v}{\partial u} = h'(v)$$

$$\frac{\partial d}{\partial t} = f'(v)$$

Vanishing gradient

- Weights in the first layers are more difficult to train that the ones in the last layers.
- Let's assume we have a neural network with N layers. Then the gradient of a weight in layer i will be like:

$$\Delta w = h'_{i+1}(x) * h'_{i+2}(x) * \cdots * h'_N(x) * other\ factors$$

- If all $h_k{}'(x)$ are between 0 and 1(happens with typical activation functions such as sigmoid and tanh), the gradient will tend to 0 in the first layers. This is called **Vanishing Gradient**.

Vanishing gradient - Mitigations

- Some Deep learning architectures have been developed in order to deal with the Vanishing Gradient problem (for example, LSTMs).
- Simple mitigation technique: use **RELU as hidden activation** function.

$$relu(x) = \begin{cases} x \ if \ x > 0 \\ 0 \ if \ x \leq 0 \end{cases}$$

- RELU could cause the opposite problem: **gradient explosion**. This happens when the gradient of any weight is too large causing an overflow in the digital number representation. Solution is **clipping on the gradient** for some value (for example, forcing the gradient to be always less tan 0.2).

4

Learning Process

Batch Training

- In order to increase the convergence speed, we will **pass several samples through the network at the same time**. A bucket of simples that are process simultaneously is called a **batch**.
- Gradients are computed for each sample and then average for computing Δw. This makes the estimation less sensible to noise.
- We will try to use a batch size as large as possible, although a too large value could cause **memory problems.**

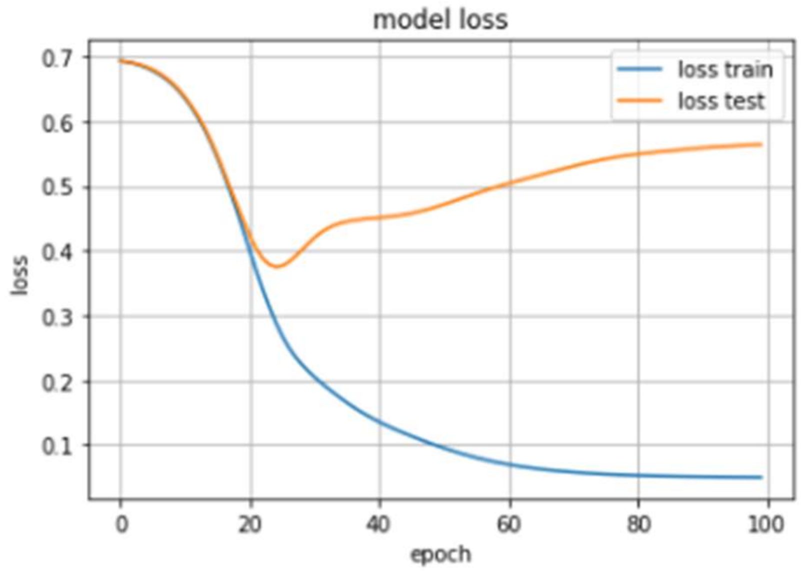
Epochs

- Normally, we will have a lack of data which will cause the **network cannot converge**.
- Solution: start again
- The whole dataset will be used to train the network several times. Each time is called an **epoch**.
- Using a too large number of epochs could cause the model learns how to model even the noise of the train data and thus cannot generalize to new data (**overfitting**).

Epochs

- In order to avoid overfitting, we use a **validation dataset** in which we can evaluate our model after each epoch.

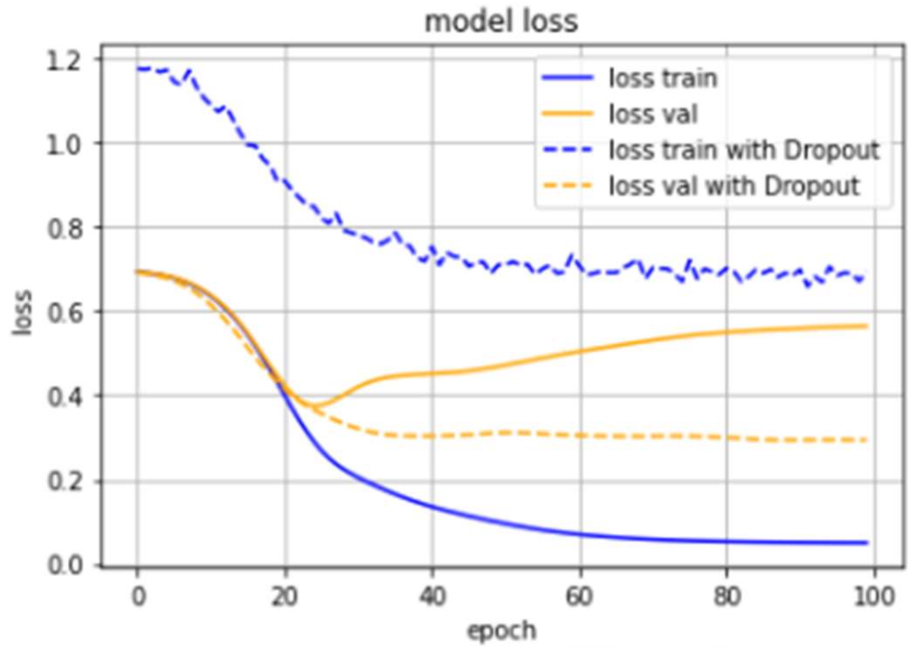
- We can stop the training if the validation loss has not decreased during a few epochs (**Early Stopping**).



Epochs - Dropout

- Another technique to avoid overfitting is **Dropout**. This technique randomly selects some neurons and **disable them** during the batch training process (different neurons are chosen in different batchs).

- Intuition: when a sample is passed twice through the neural network it will not activate the same neurons and the model **will not be able to exactly learn the signal**.



COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

5

Modifications of Training Algorithms

Training algorithms

- The learning rate α must be manually chosen (is not trainable). The selection could be problematic:
  - **If it is too small**, the convergence will be very slow.
  - **If it is too large**, the algorithm could diverge!!!

- Modifications of gradient descent have been developed in order to deal with this and other problems.

- One of the most used: **Adaptative Moment Estimation (ADAM)**. Adapts the learning rate during the training process.

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

comillas.edu