La introducción de esta parte está bien para empezar

<https://media.githubusercontent.com/media/rrgalvan/nnpde/main/presenta/nn-pde.pdf>

Funciones de activación:

# Definición:

The activation function is usually an abstraction representing the rate of action potential.

In its simplest form, this function is binary—that is, either the neuron is firing or not. Neurons also cannot fire faster than a certain rate, motivating sigmoid activation functions whose range is a finite interval.

Esto significa que si recibimos cierta entrada debemos de valorar si vamos a realizar una acción o no, en su forma más simple esto implica que es binaria, siendo por ejemplo 0 si no se toma acción y viceversa. Puede también ocurrir que queremos una probabilidad entre 0 y 1, cuanto más cercana sea a 0 más probable es no tomar acción, de ahí las sigmoides.

# Why do we need activation functions

Although the neural network becomes simpler, learning any complex task is impossible, and our model would be just a linear regression model.

Let’s suppose we have a neural network working without the activation functions.

In that case, every neuron will only be performing a linear transformation on the inputs using the weights and biases. It’s because it doesn’t matter how many hidden layers we attach in the neural network; all layers will behave in the same way because the composition of two linear functions is a linear function itself.

Loss functions

# Definition

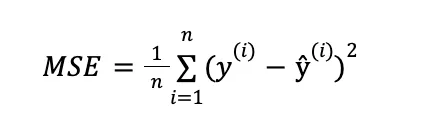
A loss function is a function that compares the target and predicted output values; measures how well the neural network models the training data. When training, we aim to minimize this loss between the predicted and target outputs.

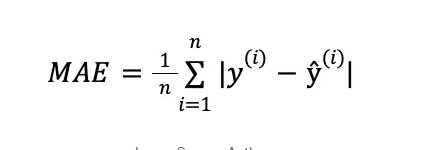
We can think of this akin to residuals, in statistics, which measure the distance of the actual y values from the regression line (predicted values) — the goal being to minimize the net distance.

# Types of Loss Functions

In supervised learning, there are two main types of loss functions — these correlate to the 2 major types of neural networks: regression and classification loss functions

1. Regression Loss Functions — used in regression neural networks; given an input value, the model predicts a corresponding output value (rather than pre-selected labels); Ex. Mean Squared Error, Mean Absolute Error
2. Classification Loss Functions — used in classification neural networks; given an input, the neural network produces a vector of probabilities of the input belonging to various pre-set categories — can then select the category with the highest probability of belonging; Ex. Binary Cross-Entropy, Categorical Cross-Entropy





Optimizers

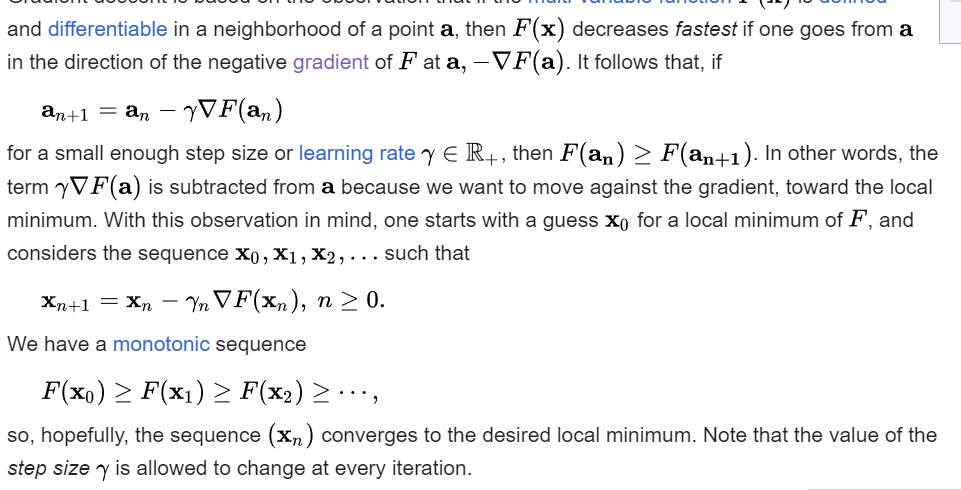
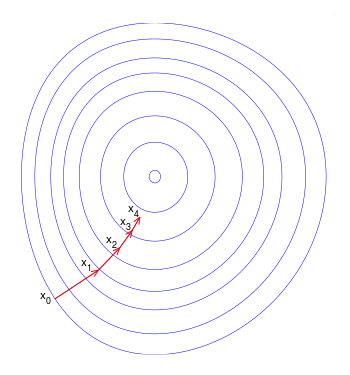
Optimizers are algorithms or methods used to minimize an error function(loss function)or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on model’s learnable parameters i.e Weights & Biases. Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

The most popular one is Adam, but to understand how a optimizer work we will explain gradient descent.

\*\*EJemplos de optimizadores: https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0

# Gradient descent

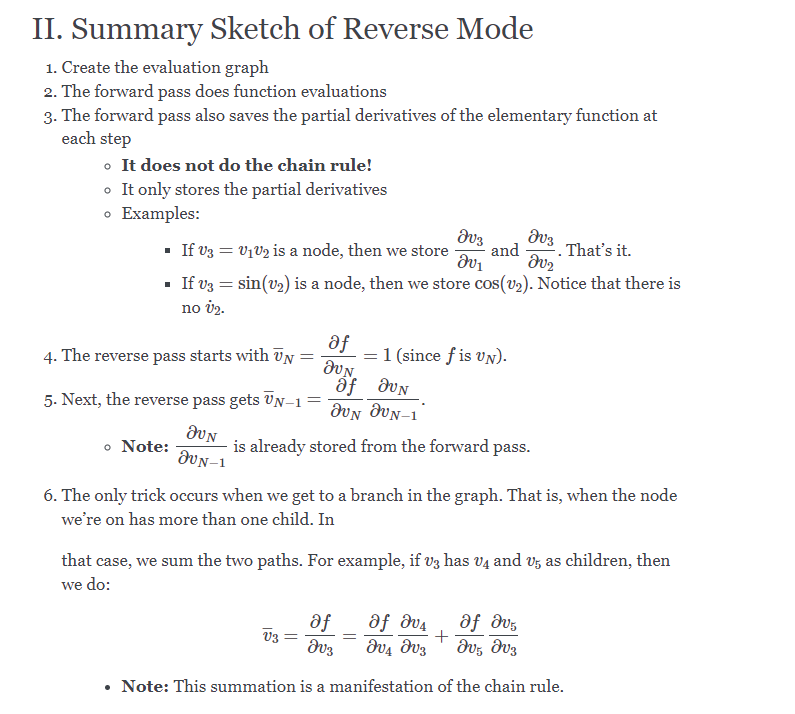
Given a stepsize lambda and a point x0 we do:



In our case x0 are the weights of the network in each layer.

How do we get the derivative of the weights

Importante: <https://colab.research.google.com/drive/1_a9HKnVccMtwRf9ewuL8iL-BUkKAlElg>



Hyperparámetros

# Definition

A hyperparameter is a constant parameter whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include learning rate, the number of hidden layers and batch size.

Usually if the model is not behaving well, we must tune tha hyperparameters making so the model learns slower (learning rate), make it more complex (adding more layers).

**Learning rate**

The learning rate defines the size of the corrective steps that the model takes to adjust for errors in each observation. A high learning rate shortens the training time, but with lower ultimate accuracy, while a lower learning rate takes longer, but with the potential for greater accuracy.

**Batch size**

batch size is the number of sub samples given to the network after which parameter update happens.

A good default for batch size might be 32. Also try 32, 64, 128, 256, and so on.

**Number of epochs**

Number of epochs is the number of times the whole training data is shown to the network while training.

In short, training an NN is an iterative process where each iteration is called epoch, on each epoch we give #batch\_size samples to the NN and calculate the predicted output using the current weights, then we use the loss function to get an estimation of how well the NN is performing. Based on that the optimizer will adjust the weights.

And repeat until #Number of epochs is reached.

Software to use on NN

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

PyTorch1​2​ es una biblioteca de aprendizaje automático3​ de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones como visión artificial y procesamiento de lenguajes naturales, principalmente desarrollado por el Laboratorio de Investigación de Inteligencia Artificial4​ de Facebook (FAIR).

\*\*Poner imágenes pa ambas

A desarrollar si quieres:

The cost of training a model and implementing it

parallelization for efficient training