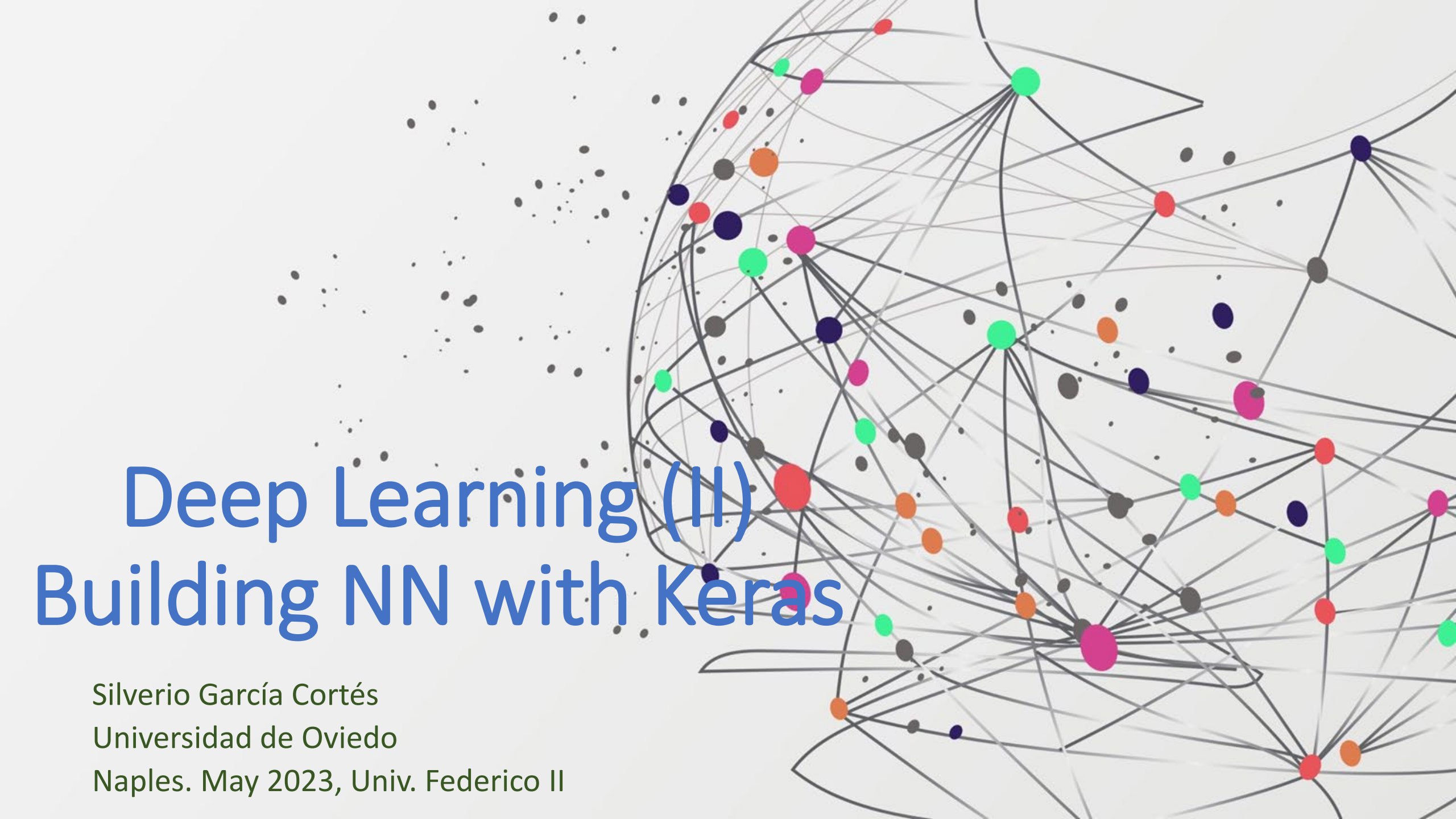


Deep Learning (II)
Building NN with Keras

Silverio García Cortés
Universidad de Oviedo
Naples. May 2023, Univ. Federico II

Building a Neural Network with Keras(I): Model

Sequential

No Sequential

- The most simple model in Keras is the so called *"Sequential"* model
- A *"Sequential"* model is built upon several layers of neurons stacked linearly one on top of other without any derivation or bifurcation. To créate a *"Sequential"* model :
  - Import *"Sequential"* class
  - Instance it (i.e. créate an object)

```
from keras.models import Sequential
modelo = Sequential()
```

  - There are different kinds of layers in keras. They have distinct inputs and outputs depending on the layers they are connected to.

```
from keras.layers import Dense
```

- A "*Dense*" layer is a fully connected layer of neurons (like the ones used on MLP)

```
capa1 = Dense(10, input_shape = (10,))
```

For the first layer of the model you must indicate how many inputs are expected from your data

- Another important parameter for the neuron layers is the activation function. If you does not set this parameter the activation will be *"Linear"*

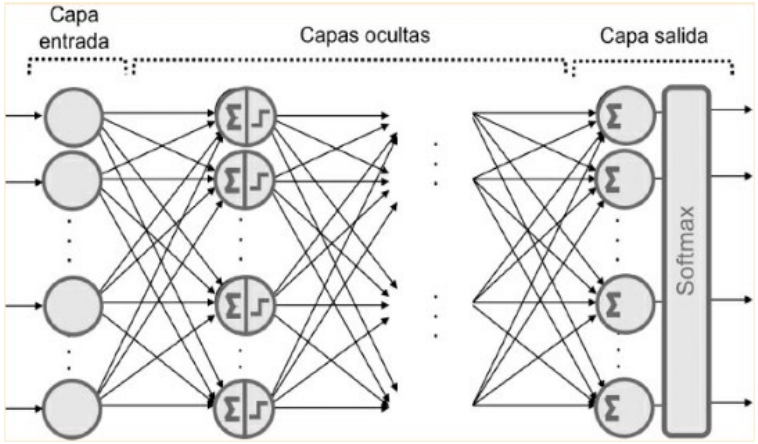
```
modelo.add(Dense(1, activation = 'sigmoid'))
```

- You can add the layer to the model and create more layers if you want.

```
modelo.add(capa1)
```

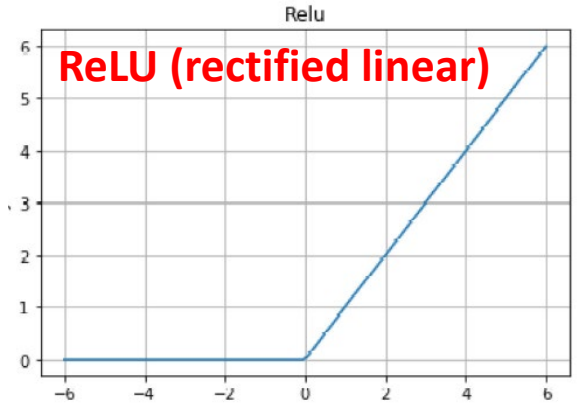
Building a Neural Network with Keras(II): activation Function

The actication function for the last layer of the model Will be chosen depending the problem to solve.



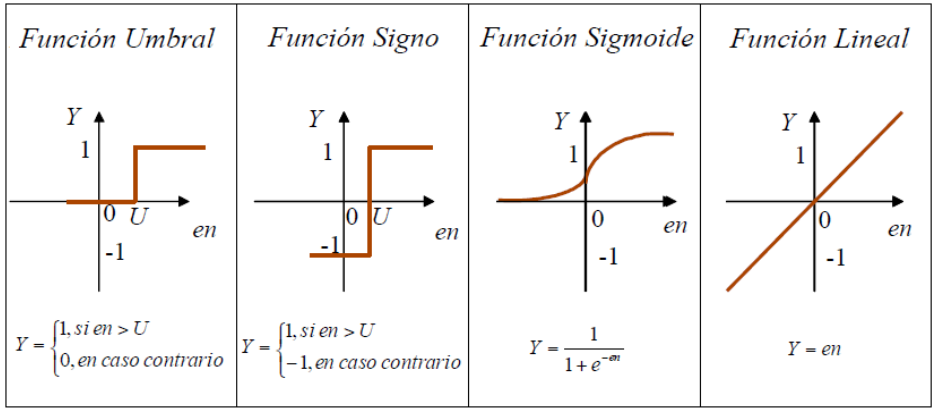
- **Linear** activation Function
  - It is used for the output layer in a **REGRESSION** model
  - This Linear act.funct. Is not used for hidden layers becuase, if used, the model cannot map non-linear relationships

- **Softmax** activation Function :
  - Used for **MULTICLASS** output layer. There must exist as many ouputs as classes
  - The summ of all outputs will be 1



- **ReLU (rectified linear unit)** activation Function: y = max(0, x)
  - ReLU: Most used act.funct. In **Hidden layers.**
  - example: Dense(10, activation = 'relu')

- **Logistic (Sigmoid)** activation Function:
  - Used in the output layer of the **BINARY CLASSIFICATION**



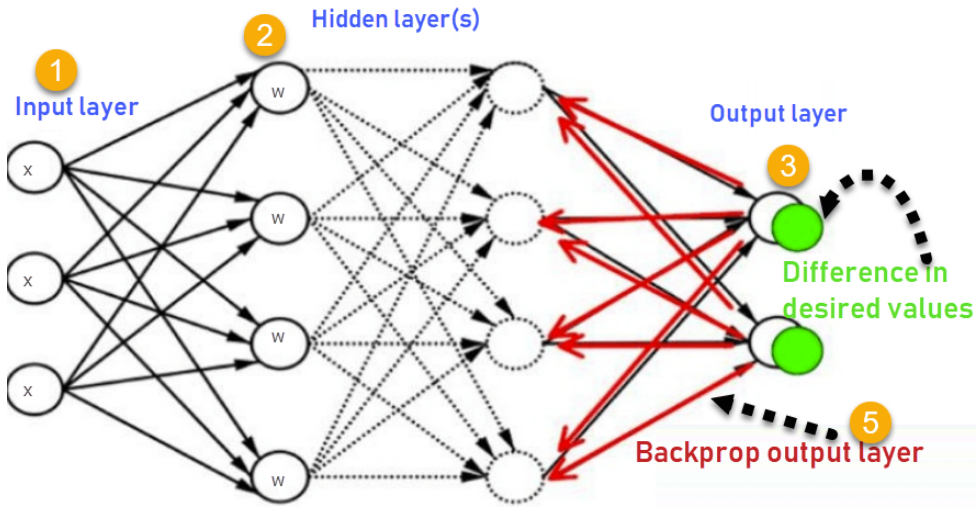
- Other advanced activation funct. are added as another layer (e.g.):

*from keras.layers import **LeakyReLU***
modelo.add(Dense(10))
modelo.add(LeakyReLU(alpha=0.02)

Building a Neural Network (III): Learning

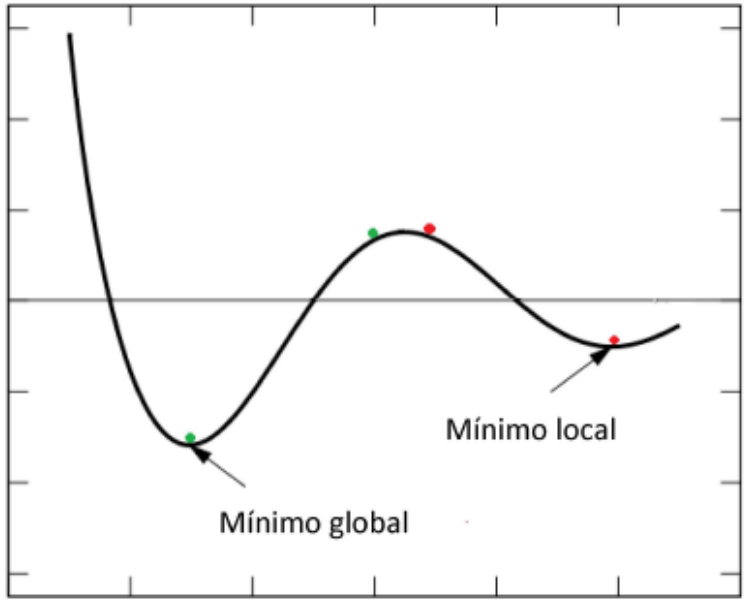
NN learning process:

- The goal is to find a global mínimum for the Error function (cost or los function).
- To achieve that we need to find right weights and bias for each neuron.
- Initial values of weights and bias can be random (there are more options for initialization)
- We find right weighs and values iteratively passing all the training data through the NN and calculating the error committed between predicted values and real values for our training samples for each iteration
- Error estimation is done through the **loss function**



BackPropagation

- From the measured error in the model output (we begin with random weight and bias values in first epoch)
- We calculate backward corrections for each weight and bias for all the neurons in the model using chain rule to calculate the gradient of the Loss function with respect each weight and bias.
- Retropropagation algorithms got 2 main parameters: learning rate and epoch number (number of times the training samples are presented forwrd to the model)
- It is not guaranteed that the process converge to global minimum (a local one can be reached instead)



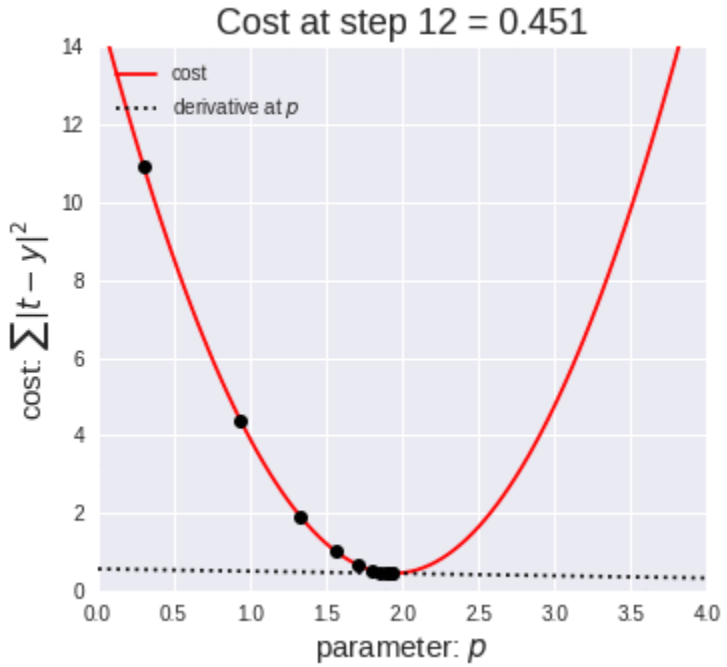
Iteration methods:

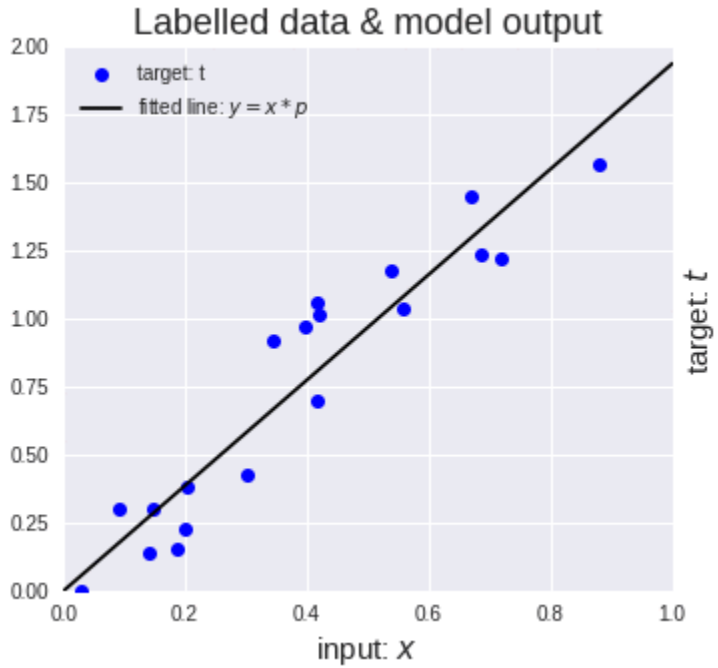
- **Gradient Descent:** Classic method. One iteration per epoch with all the training samples
- **Stotacstic Gradient Descent (SGD):** One iteration per sample. Sample order is random.
- **Minibatch:** Most common used. One iteration with a subset of the training samples (batch) in each iteration.

The optimizar is chosen when we select the iteration method.

Learning: Epochs, batch size, iterations

We need terminologies like epochs, batch size, iterations only when the data is too big which happens all the time in deep learning and we can't pass all the data to the computer at once. So, to overcome this problem we need to divide the data into smaller sizes





Epoch is when an **ENTIRE dataset** is passed forward and backward through the neural network only **ONCE**.

BATCH SIZE: Total number of training examples present in a single batch (batch: smaller set of data in which is splitted training set).

ITERATION: number of batches needed to complete one epoch. (The number of batches is equal to number of iterations for one epoch.) (e.g.: We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.)

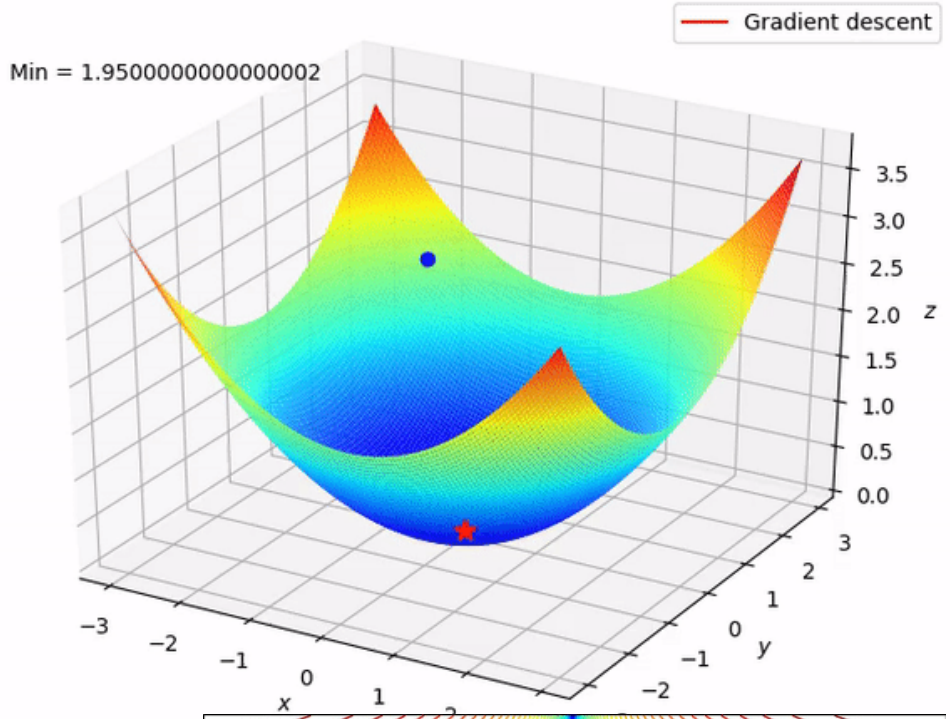
Building a Neural Network (IV): Loss functions and optimizers

Loss function

- Error estimation is performed through a **loss (cost) function**

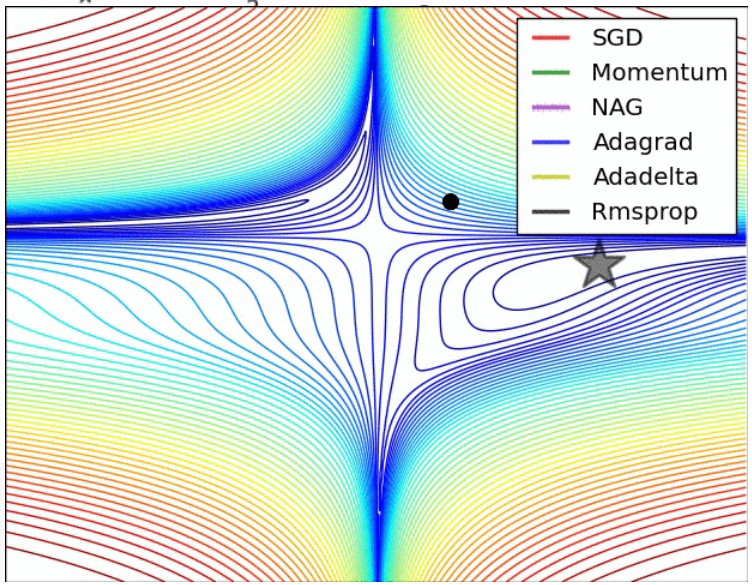
  This function is chosen depending on the problema to solve:

  - MSE (Mean Square Error): Regression problems
  - Binary Cross Entropy: Binary classification
  - Categorical Cross Entropy: Multiclass classification problems
  - Loss function is declared when compiling the model after its def



Optimizer

- Also defined when compiling the model. Some posible optimizers are:
  - SGD, Adam, RMSprop, Adadelta, Adagrad, Adamax, Frtl,… and others
- https://keras.io/api/optimizers/




```
adam = Adam(lr=0.005)
mlp.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
```

Optimizers

- **Stochastic Gradient Descent**

instead of taking the whole dataset for each iteration, we randomly select the batches of data. That means we only take a few samples from the dataset.

- **Stochastic Gradient Descent With Momentum**

Introduces momentum helps to help in faster convergence of the loss function by adding a fraction of the previous update to the current update. the learning rate should be decreased with a high momentum term

- **Mini Batch Gradient Descent**

using a batch of data instead of taking the whole dataset.Nedds to tune the hyperparameter that is "mini-batch-size"

- **Adagrad (Adaptive Gradient Descent)**

it uses different learning rates for each iteration.depends upon the difference in the parameters during training. The more the parameters get changed, the more minor the learning rate changes.

abolishes the need to modify the learning rate manually

- **RMS Prop (Root Mean Square)**

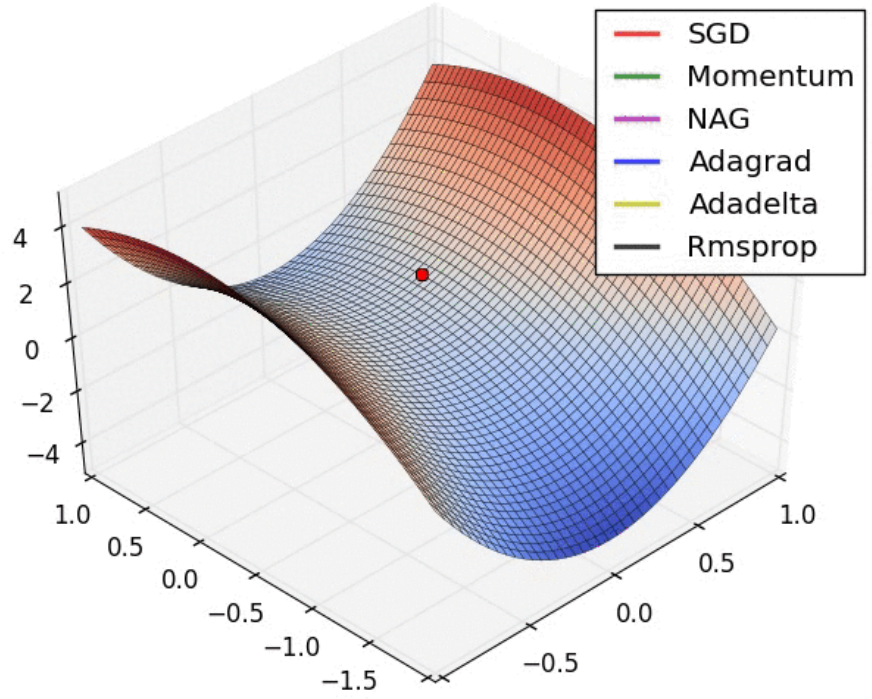
The problem with the gradients is that some of them were small while others may be huge. So, defining a single learning rate might not be the best idea. RPPROP uses the gradient sign, adapting the step size individually for each weight.

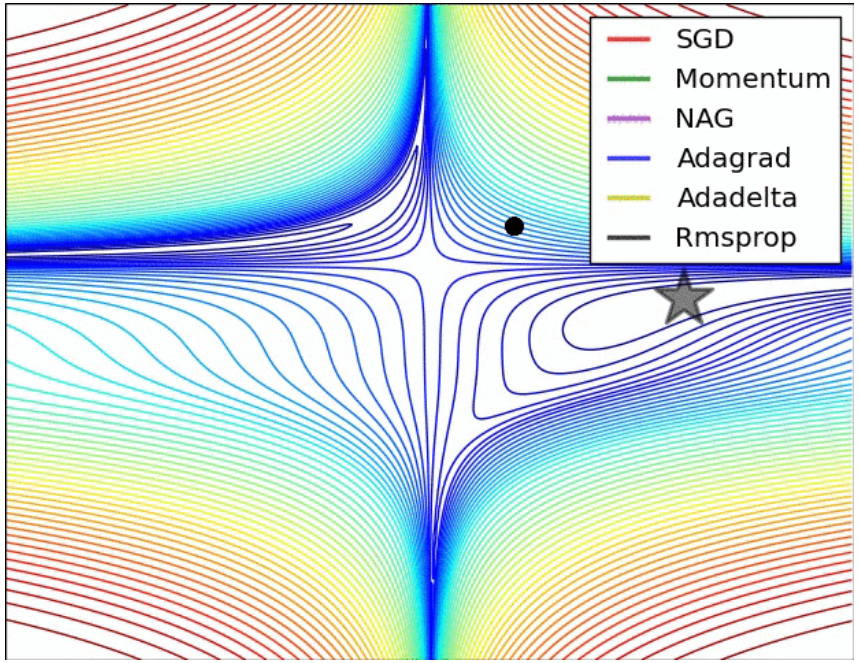
- **Adam**

adaptive moment estimation. This optimization algorithm is a further extension of stochastic gradient descent to update network weights during training. Unlike maintaining a single learning rate through training in SGD,

Adam optimizer updates the learning rate for each network weight individually.

[analyticsvidhya A-comprehensive-guide-on-deep-learning-optimizers]





Building a Neural Network (V): Compiling and fitting

Compile method:

- During compiling an optimizer , cost function and metrics are defined.
- Optimizer can also be instanciated before compiling. (Optimizers (keras.io))

```
mlp.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
```

```
from keras.optimizers import Adam
adam = Adam(lr=0.005)
model.compile(loss='binary_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])
```

```
# compiling the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

Fittting method:

- Fitting (training) is performed using the fit() method.
- Number of epochs and batch_size for training can be set in compiling
- At the end of the training an history object is returned. It contains metrics of the process

```
hist = mlp.fit(x_train,y_train, batch_size = 128, epochs = 10, verbose=1)
```

Model evaluation: Metrics

- We need to have some “metrics” to evaluate the performance of the trained models. In the case of Regression (a continumm variable prediction) the MSE (Mean Squared Error) or RMSE (roor mean squared error) can be used.
- In classification problems oother metrics are used: “accuracy”, “precision”, “recall” and “F1”. The confusion matrix can be useful to calculate that metrics.

Accuracy$= \frac{True\ Positive}{Total\ samples}$

Precisión (precision): Proportion of real success with respect to all the de positive hits assigned by the classificator (real or not)

$\boldsymbol{Precision} = \frac{TP}{TP + FP}$

Exhaustivity (Recall): Proportion of real success assigned by the classificator with respect to all the the real positive hits

Recall $= \frac{TP}{TP+FN}$

F1-score $= 2\frac{Recall * Precision}{Recall+Precision}$

Harmonic mean between precision and recall. It combines both measures in only oneas en un solo indicador

Confusion Matrix

Valores predichos \ Valores reales	1	0
1	VP	FP
0	FN	VN

sklearn.metrics.**accuracy_score**(*y_true, y_pred)*

sklearn.metrics.**precision_score**(*y_real, y_pred*)

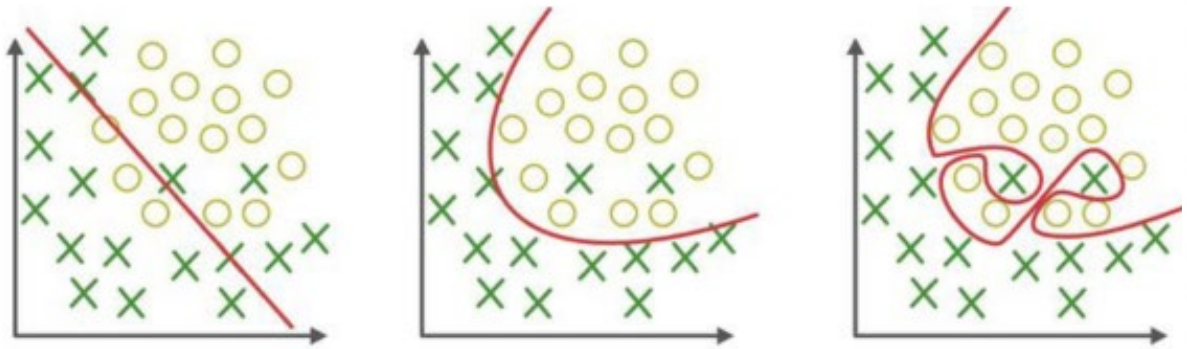
sklearn.metrics.**recall_score**(*y_real, y_pred*)

sklearn.metrics.**f1_score**(*y_real, y_pred*)

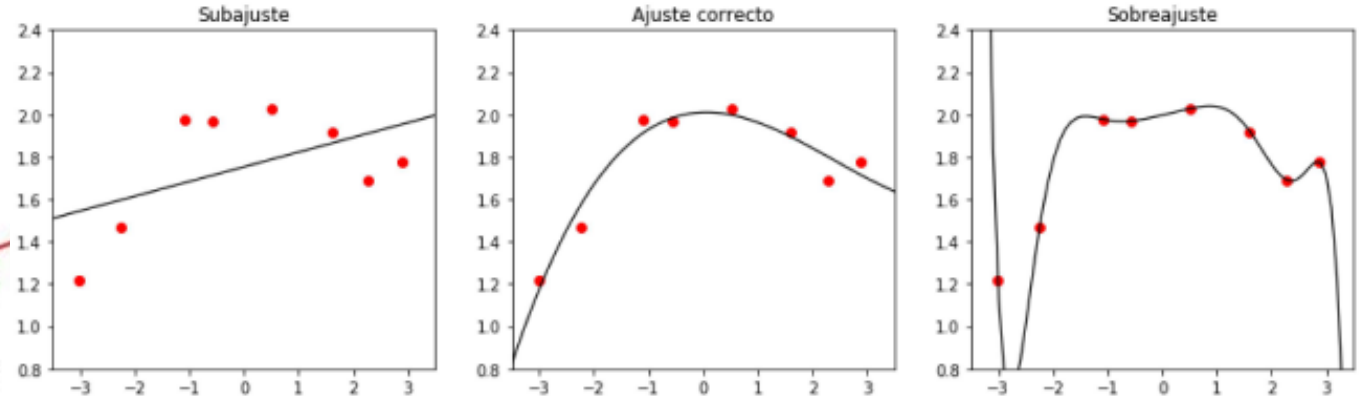
Overfitting and Underfitting in NN´s:

- Generalization is the expected result from our trained DL model. That mean that our model can give us as Good results with non seen data as the results with training dataset. If our model performs well for training data but not for not seen data that is called overfitting. If our model does not perform well even for the training data is named underfitting.
- **Overfitting:** It is very common in Deeep Learning (because of the large number of parameters that can exist in our models) and other causes. Possible reasons:
  - Too Little data: Dataset is very small or it is not enough representative of the real problema. Solution add more data.
  - Model too complex for the problem at hand. Solution use simpler model (less parameters).
  - Too much training (early stopping)
- **Underfitting:** It is the opposite case. Model is not able to learn the realtionship between the featuress and the outputs and the error does not reduce during training . Possible reasons for that are:
  - Wrong model: Maybe the model its too simple and cannot map relation between inputs and oputputs.
  - Not sufficiently descriptive features: Relations bewteen features and results are not “present” in the chosen data features

(Classification)



(Regression)



Regularization in NN´s (I):

- Avoiding overfitting.Regularisation techniques:

  Methods devised to reduce the error of models outside the training data set. The general criterion is to make the assumptions that the model makes on the data as simple as possible.

  - Some techniques against overfitting:
    - Elimination of layers or number of neurons in layers.
    - Adding more training samples
    - Other techniques :
  - Early stopping:

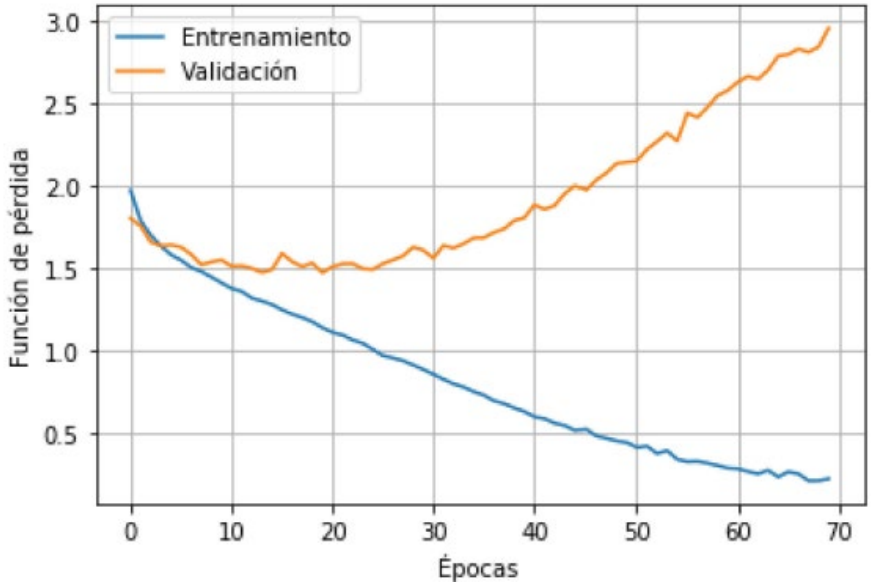
It may happen that from a training epoch onwards, the loss function is still decreasing but the model is memorising noise. This can be detected by monitoring the cost function in the validation (test) set. If it starts to grow, it is likely that overfitting is taking place.

Patience: Number of epochs where the loss function does not improve before stopping

- It is implemented as a “callback” in the **fit() method**
- It is necessary to pass to it the validation data

```
from keras.callbacks import EarlyStopping
```

```
parada = EarlyStopping(patience=4)
hist = modelo.fit(x_train,y_train, batch_size = 128,
                  epochs = 20,
                  verbose = 1,
                  validation_data=(x_val, y_val),
                  callbacks=[parada])
```



Regularization in NN´s (II):

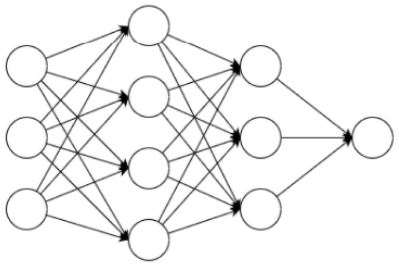
- **L2 regularisation:**

In overfitting, a network memorises too much certain patterns and this results in weights with very high values. Therefore, another way to regularise is to penalise the parameters with high values in the cost function by means of an additional term

$$L(\Theta) = \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n}(f(x_i,\Theta) - y_i)^2 + \alpha\Omega(\Theta)$$

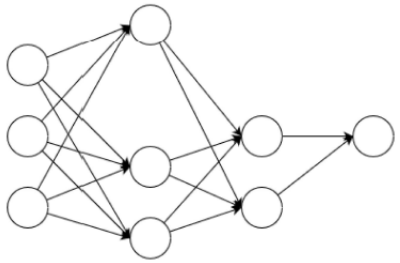
The term Ω(Θ) is the Euclidean norm of the vector of model weights. Minimising this function requires iteratively searching for values of the weights that minimise the error, but have the minimum possible value.

```
from keras.regularizers import l2
mod3 = Sequential()
mod3.add(Dense(512, input_shape=(3072,), activation='relu', kernel_regularizer=l2(0.005)))
mod3.add(Dense(256, activation='relu', kernel_regularizer=l2(0.0025)))
mod3.add(Dense(128, activation='relu', kernel_regularizer=l2(0.002)))
mod3.add(Dense(10, activation='softmax'))
```



- **Dropout:** It consists of randomly removing neurons from the network during the training phase. This avoids that several parameters of several neurons are "cooperating" to model the same effect or noise.

It is implemented by a scalar value between 0 and 1 corresponding to the probability that neurons are no longer available. The maximum regularisation is given for 0.5 since it implies an uncertainty on the availability of neurons of 50%.

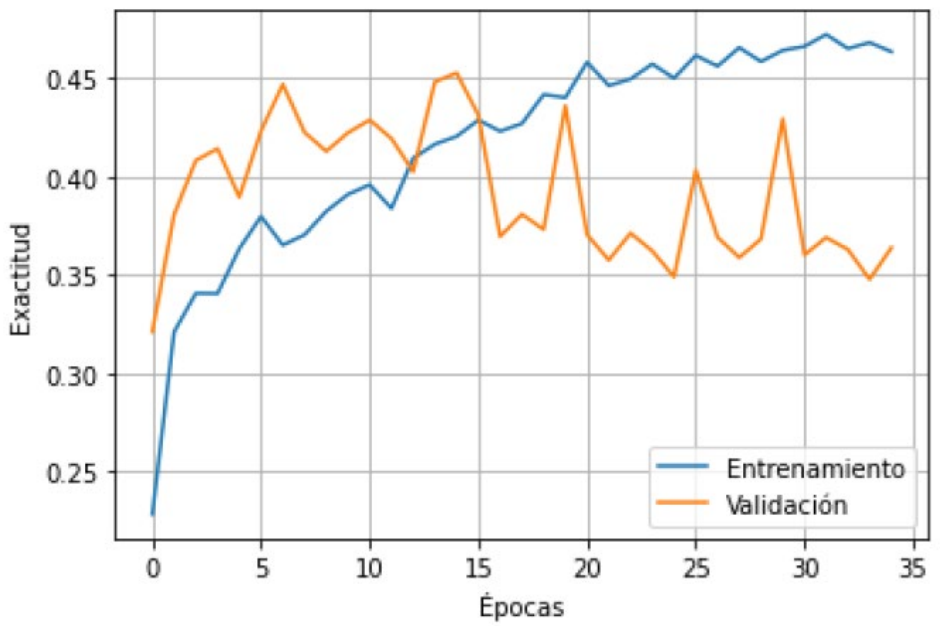


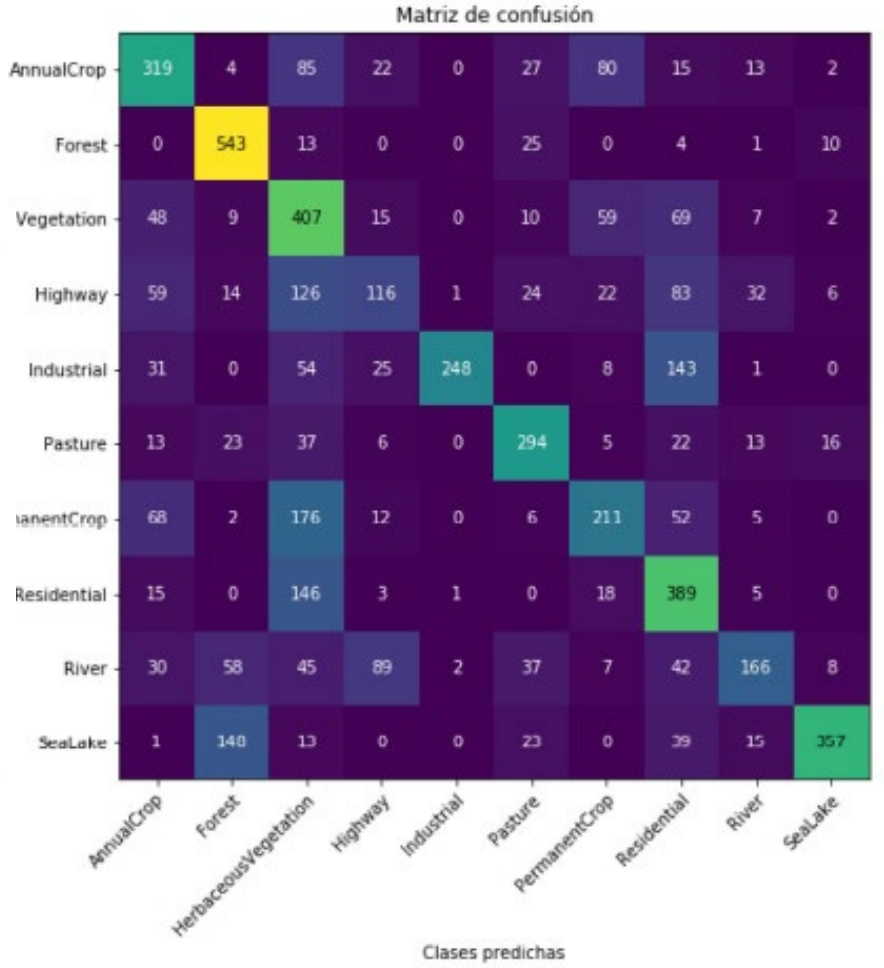
```
from keras.layers import Dropout
modelo = Sequential()
modelo.add(Dense(100, input_shape = (600,), activation='relu'))
modelo.add(Droput(0.25))
modelo.add(Dense(5, activation='softmax'))
```

Neural Network Hyperparameters

Hyperparameters of an NN are those parameters that are susceptible to change or choice by the operator and that can improve the performance of the NN. Some of them are:

- Number of layers and number of neurons in them
- Layer activation function
- Optimizer:
  - Type
  - Learning rate
  - Number of epochs

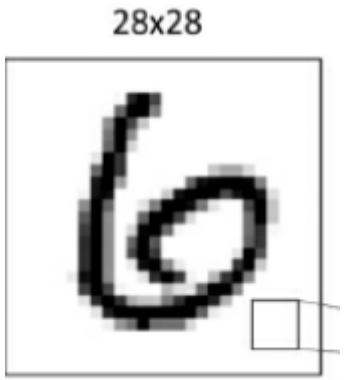




NN example: written characters recognizing

National Institute of Standards and Technology
U.S. Department of Commerce

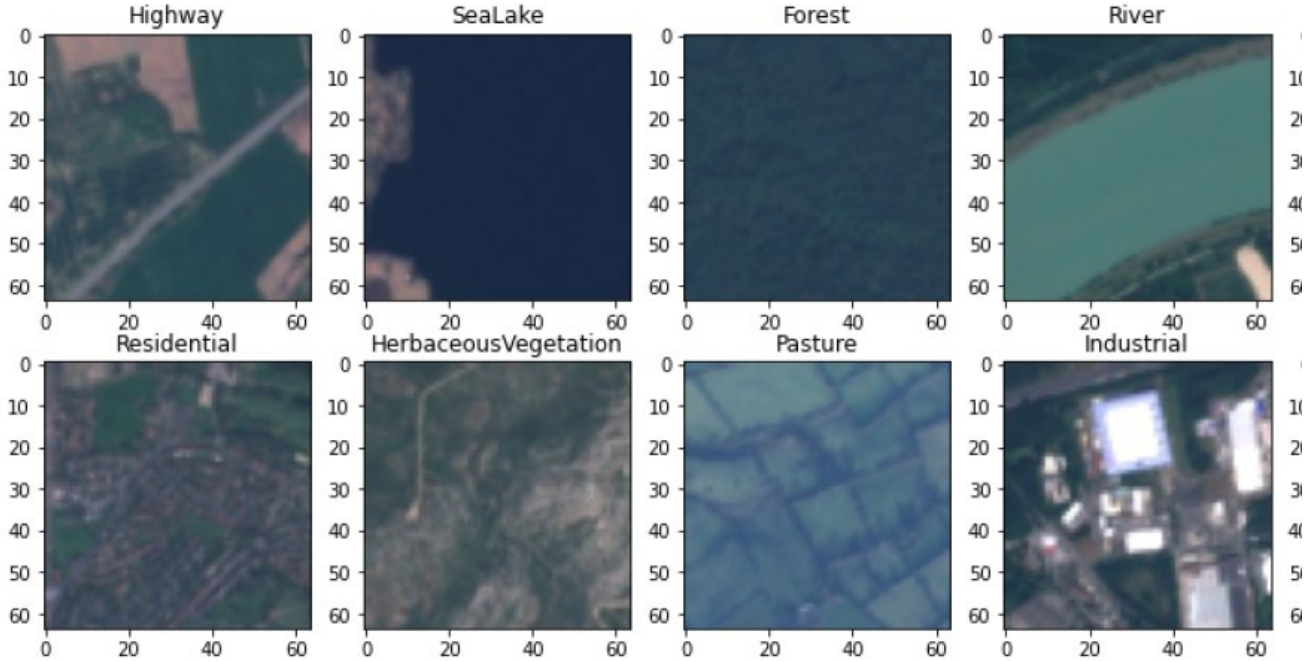
- The MNIST (National Institute of Standards and Technology) dataset is composed of 60000 images (8 bits and 28x28 píxels each) with numerical characters handwritten.
- Two image groups are defined one training set of 50K and another 10K set for testing
- The problema to solve is a multiclass classification. We want our NN recognize a given digit image

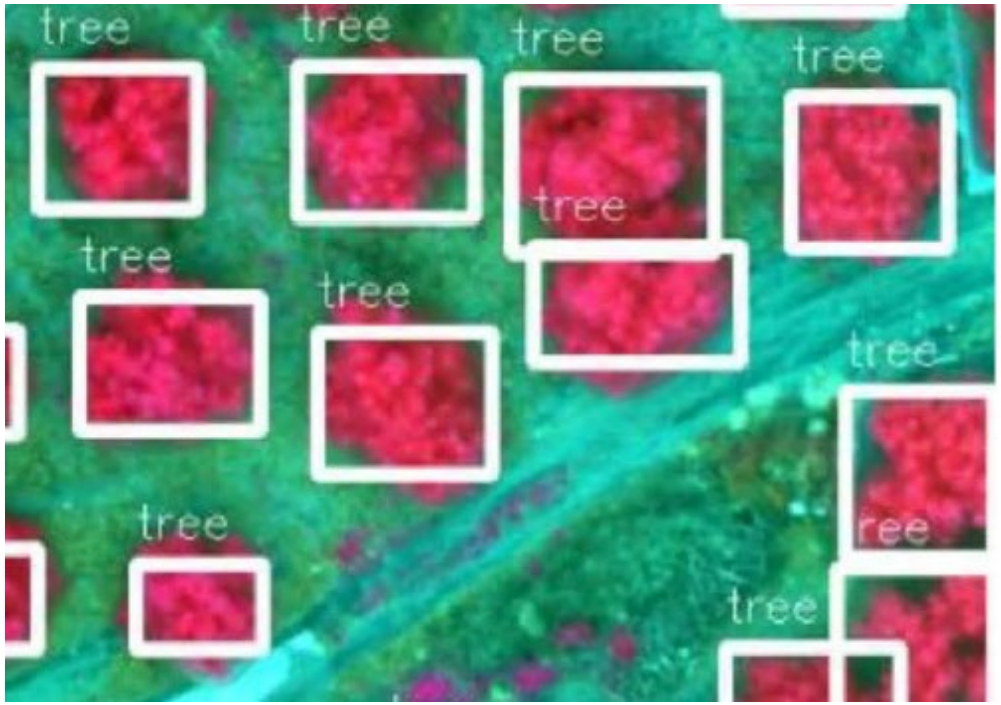


This is a six number !!!

Deep Learning applications (I):

- NN's are able to learn high level concepts (much different tan the Digital Numbers in the case of images) Complex patterns like shapes, image context , relative positions etc can learn by this technique.
- Convolutional Neural Networks (CNN's) are the NN´s which can efficiently process images in Deep Learning. The more important tasks of this CNN's are:
  - **Scene Clasification**: It´s the easisest because the position of the object or shape is not involved in the process. A complete subimage (tile) is associated to a category or class (e.g. dogs , cats)
  - **Object Detection**: The netowork detect the objects (e.g cars, trees, etc) and it place a bounding box around them). This task requires the labelling of the objects in the training images but it allow to count, or track the objects in the images.
  - **And …. (see next slide)**

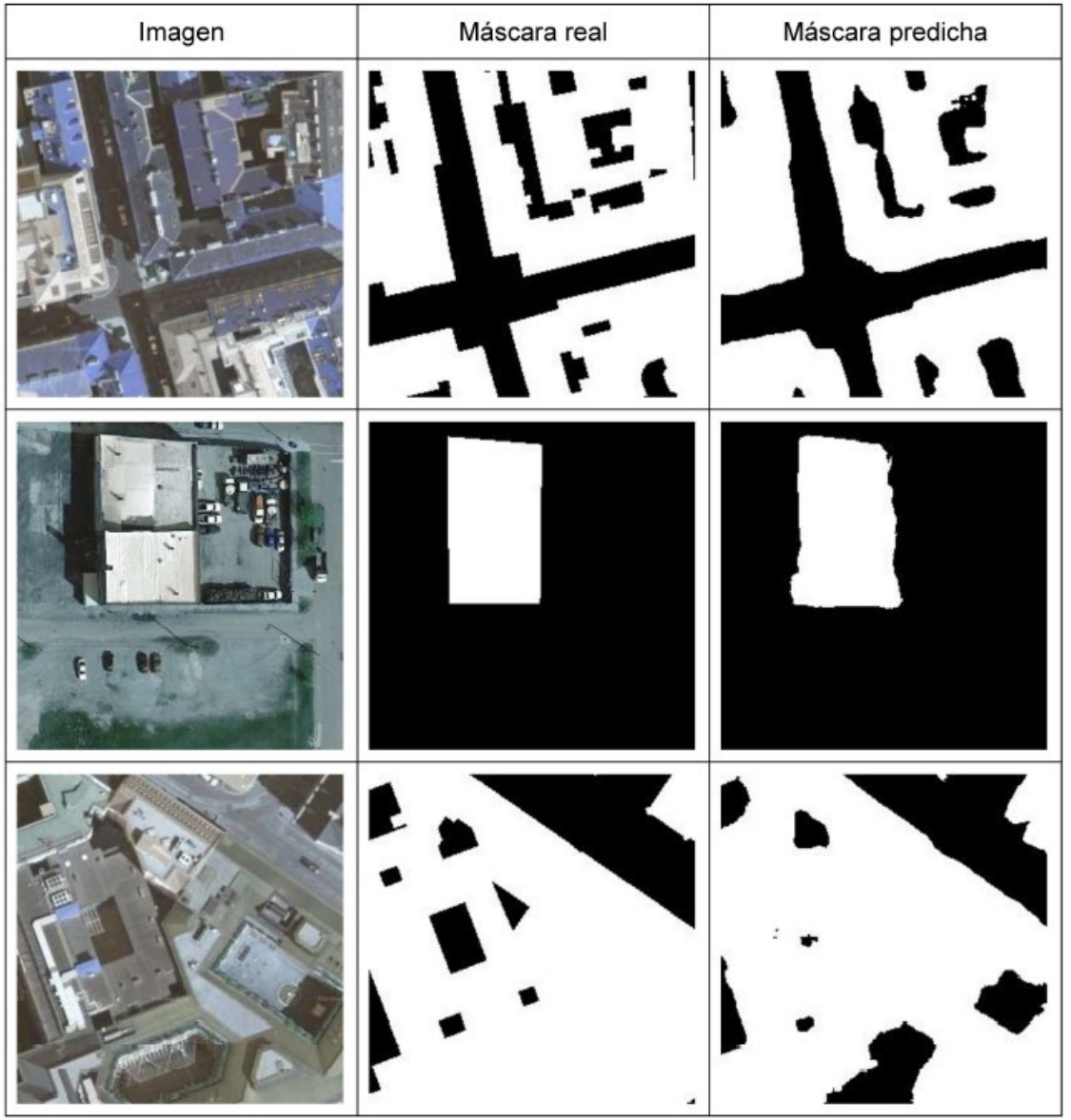




DL applications (II): Semantic segmentation

Semantic Segmentation

- **Each pixel in the image receives a label indicating which class it belongs to.**
- **The training data creation is expensive becuase it implies to creatre masks for each interest class (e.g buildings and no buildings in the binary classification example on the picture)**



References:

- **Raschka Sebastian, Mirjalili Vahid, 2017. Python Machine Learning. 2nd ed. Packt Birmingham-Mubai**
- **Brownlee Jason. (2019). Deep Learning with Python: Develop Learning Models on Theano and TensorFlow using Keras. (Machine Learning Mastery)**
- **Torres Jordi. 2020. Python Deep Learning: Introducción práctica con Keras y Tensorflow 2. 1ª ed. Marcombo.**
- **Chollet, Francois. 2020. Deep Learning con Python. Ed ANAYA**
- **The MNIST dataset. http://yann.lecun.com/exdb/mnist/ . Accessed Dec. 2020**
- **German Research Center for Artificial Intelligence. Multimedia análysis and Data Mining. Dowloads. Datasets for Machine Learning. Downloads — Multimedia Analysis and Data Mining Research Group (dfki.de) . Accessed Dec. 2020.**
- **Epoch vs Batch Size vs Iterations [https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9]**
- **A Comprehensive Guide on Optimizers in Deep Learning [https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/]**