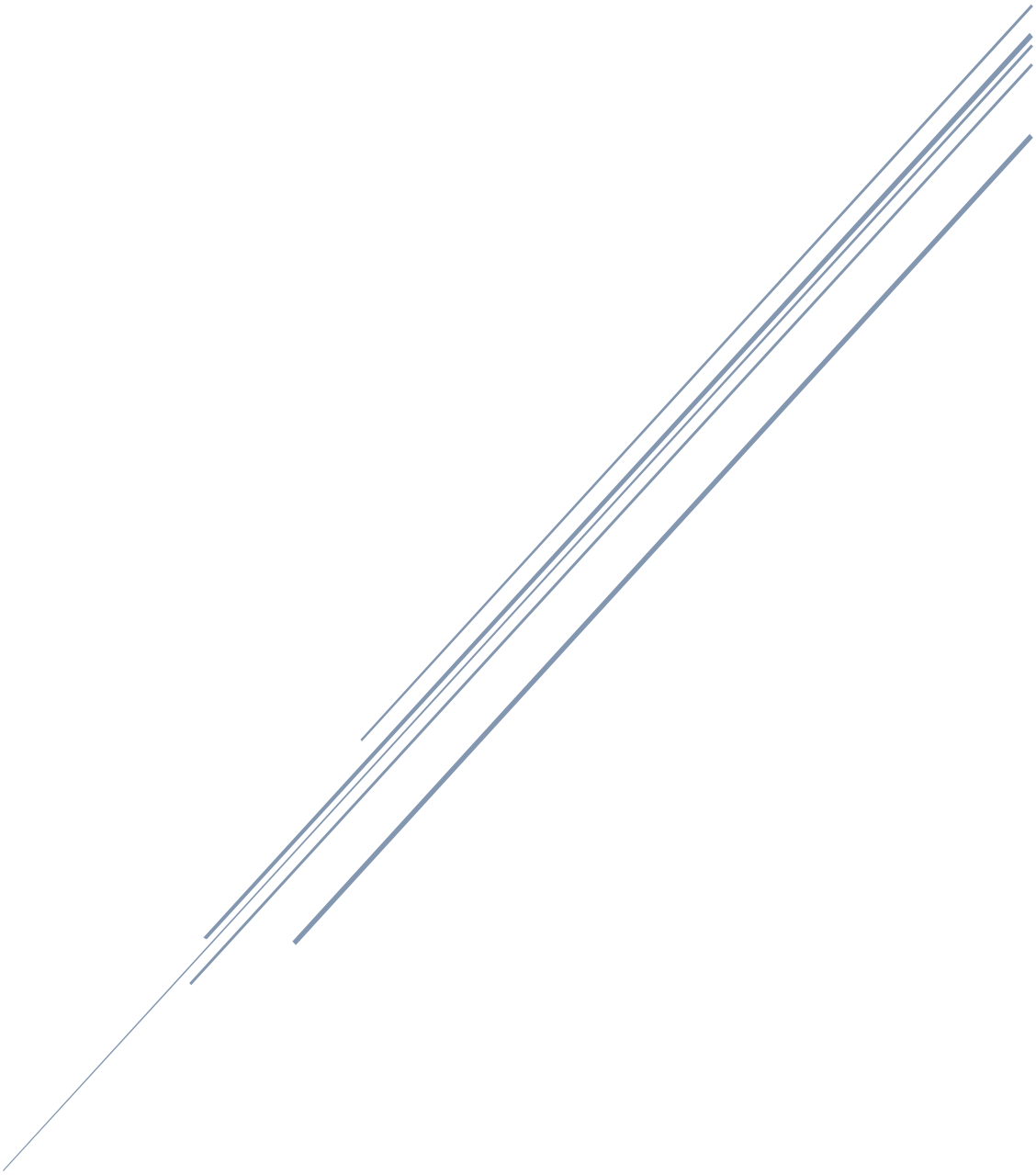
INTRO TO TENSORFLOW

Module 1



v-cardona

Deep Learning with Tensorflow

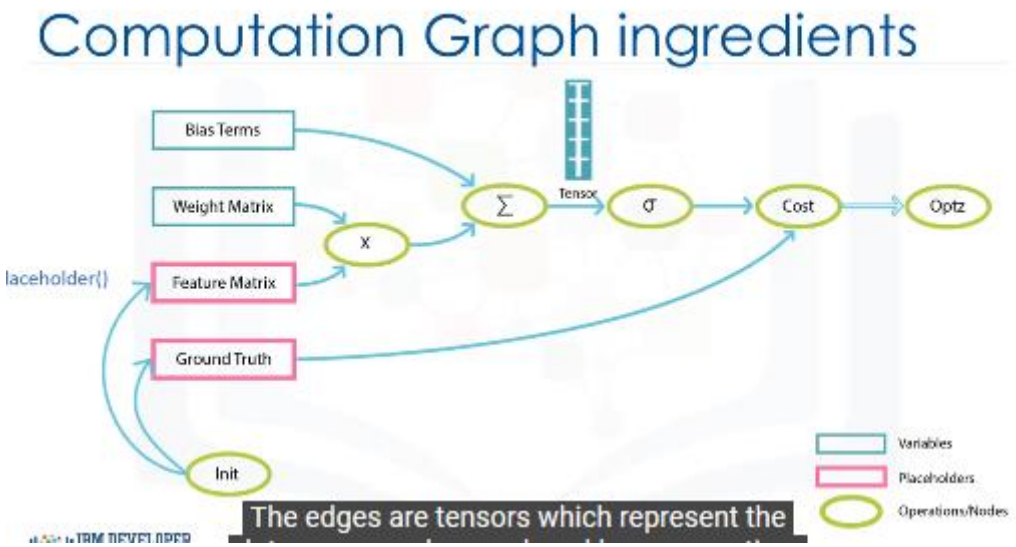
Contenido

Intro to TensorFlow

TensorFlow defines computations as Graphs, and these are made with operations (also know as “ops”). So, when we work with TensorFlow, it is the same as defining a series of operations in a Graph.

TensorFlow’s structure is based on the execution of a data flow graph. A data flow graph, has two basic units. The nodes that represent a mathematical operation, and the edges, which represent the multi-dimensional arrays, known as a tensors. This high-level abstraction reveals how the data flows between operations.

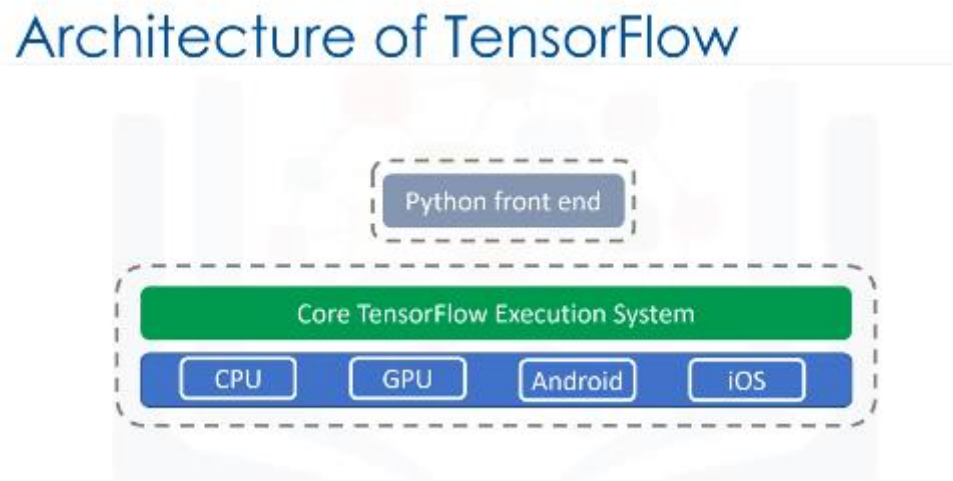
Tensor is the data that is passed between the operations, a multidimensional array.



The nodes are called operations, which represent units of computation. The edges are tensors which represent the data consumed or produced by an operation. In this graph, Feature matrix is a placeholder. Placeholders can be seen as "holes" in your model -- "holes" through which you can pass the data from outside of the graph. Placeholders allow us to create our operations in the graph, without needing the data. When we want to execute the graph, we have to feed placeholders with our input data. This is why we need to initialize placeholders before using them. TensorFlow variables, are used to share and persist some values, that are manipulated by the program.

When you define a place- holder or variable, TensorFlow adds an operation to your graph. In our graph, “Weight matrix” and “Feature matrix” should be multiplied using a multiplication operation. After that, Add operation is called, which adds the result of the previous operation with bias term. The output of each operation is a tensor. The resulting tensors of each operation crosses the next one until the end where it's possible to get the desired result.

To execute these operations as computations, we must launch the Graph into a Session. The session translates and passes the operations represented into the graphs to the device you want to execute them on, be it a GPU or CPU. In fact, TensorFlow's capability to execute the code on different devices such as CPUs and GPUs is a consequence of it's specific structure.



- Create graph:
  graph1 = tf.Graph()

- Calling tf.constant([2], name = 'constant_a') adds a single tf.Operation to the default graph. This operation produces the value 2, and returns a tf.Tensor that represents the value of the constant. Notice: tf.constant([2], name="constant_a") creates a new tf.Operation named "constant_a" and returns a tf.Tensor named "constant_a:0"

- Create a session and run it:
  sess = tf.Session(graph = graph1)
  result = sess.run(a)
  print(result)
  sess.close()

- To avoid having to close sessions every time, we can define them in a **with** block, so after running the **with** block the session will close automatically:
  with tf.Session(graph = graph1) as sess:
  result = sess.run(c)
  print(result)

Dimensions:

Dimension	Physical Representation	Mathematical Object	In Code
Zero	Point	Scalar (Single Number)	[1]
One	Line	Vector (Series of Numbers)	[1,2,3,4,...]
Two	Surface	Matrix (Table of Numbers)	[[1,2,3,4,...], [1,2,3,4,...], [1,2,3,4,...],...]
Three	Volume	Tensor (Cube of Numbers)	[[[1,2,...], [1,2,...], [1,2,...],...], [[1,2,...], [1,2,...], [1,2,...],...], [[1,2,...], [1,2,...], [1,2,...] ,...]...]

- Define arrays:
  Scalar = tf.constant(2)
  Vector = tf.constant([5,6,2])
  Matrix = tf.constant([[1,2,3],[2,3,4],[3,4,5]])
  Tensor = tf.constant([[[1,2,3],[2,3,4],[3,4,5]] , [[4,5,6],[5,6,7],[6,7,8]] , [[7,8,9],[8,9,10],[9,10,11]]])

- Shape of our data structure:
  Scalar.shape
  Tensor.shape

- Add matrix:
  Tf.add(matix_one, matrix_two) or matrix_one + matrix_two.

- Multiply matrix:
  Tf.matmult(matrix_one, matrix_two)

TensorFlow adds a tf.Operation to your graph. Then, this operation will store a writable tensor value that persists between tf.Session.run calls. So, you can update the value of a variable through each run, while you cannot update tensor (e.g a tensor created by tf.constant()) through multiple runs in a session.

To define variables we use the command tf.Variable(). To be able to use variables in a computation graph it is necessary to initialize them before running the graph in a session. This is done by running tf.global_variables_initializer().

- update = tf.assign(v, v+1); // it takes de value v, and update to v + 1, then it store it on update.
- **tf.placeholder(*datatype*)**, where ***datatype*** specifies the type of data (integers, floating points, strings, booleans) along with its precision (8, 16, 32, 64) bits.

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

Now we need to define and run the session, but since we created a "hole" in the model to pass the data, when we initialize the session we are obligated to pass an argument with the data, otherwise we would get an error.

To pass the data into the model we call the session with an extra argument feed_dict in which we should pass a dictionary with each placeholder name followed by its respective data, just like this:

- *a = tf.placeholder(tf.float32)*
  *b = a * 2*
  with tf.Session() as sess:
  result = sess.run(b,feed_dict={a:3.5})
  print (result)

- To find value of our loss:
  Tf.reduce_mean()

- Define an optimizer method:
  optimizer = tf.train.GradientDescentOptimizer(0.05)

- Define training method of our graph:
  train = optimizer.minimize(loss) -> **.minimize()** which will minimize the error function of our optimizer.

Why use Placeholders?

1. This feature of TensorFlow allows us to create an algorithm that accepts data and knows something about the shape of the data without knowing the amount of data going in.
2. When we insert “batches” of data in training, we can easily adjust how many examples we train on in a single step without changing the entire algorithm.

Intro to Deep Learning

Deep Learning is a series of supervised, semi-supervised and unsupervised methods that try to solve some machine learning problems using deep neural networks. A deep neural network is a neural network that often has more than two layers, and uses specific mathematical modeling in each layer to process data. These networks are trying to automatically extract feature sets from the data, and this is why, they are mostly used in data types where the feature selection process is difficult, such as when analysing unstructured datasets, such as image data, videos, sound and text.

Deep neural networks

Convolutional neural network (CNN)

For image classification, for example.

This model is supposed to look at this particular sample set of images and learn from them, toward becoming trained. Later, given an unseen image of either a cat or a dog, we should be able to use this trained model to recognize the image as being one or the other.

First step in building such a model would be feature extraction, choose the best features from your images and then use those features in a classification algorithm, such as neural network. The better that you can define the feature sets, the more accurate and efficient your image-classification will be. However, the process of selecting and using the best features is tremendously time-consuming and is often ineffective.

Convolutional Neural Networks can automatically find those features and classify the images for you. So, we can say that a Convolutional Neural Network - or CNN for short - is a deep learning approach that learns directly from samples in a way that is much more effective than traditional Neural networks. Through multiple layers, a CNN learns multiple levels of feature sets at different levels of abstraction. And this leads to very effective classification.

Recurrent neural networks (RNN)

Tries to solved the problem of modelling sequential data, such as stock price in market, sentiment analysis, predict next word in a sentence, translation, recognise voice. Depends on the previous context.

Restricted Boltzmann machine (RBM)

RBM are used to find patterns in data in an unsupervised manner. They are shallow neural nets that learn to reconstruct data by themselves. They are very important models, because they can automatically extract meaningful features from a given input, without the need to label them.

Useful tasks: feature extraction, dimensionality reduction, patter recognition, recommender systems, handling missing values and topic modelling.

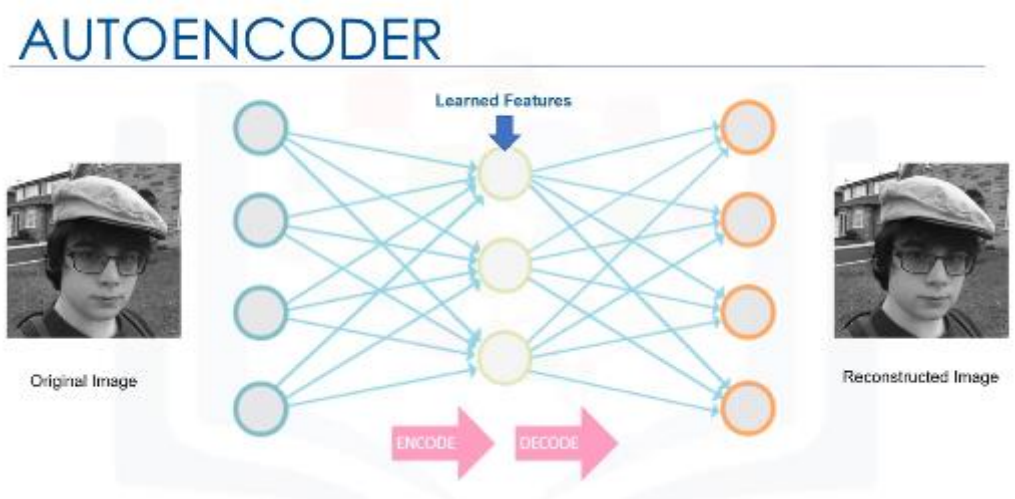
Deep belief network (DBN)

It was invented to solve the back-propagation problem, causes by local minima or vanishing gradients issues in the learning process. The applications are usually classification image recognition.

DBN is a very accurate discriminative classifier because it does not need a big set of labelled data to train the network.

Autoencoder

Autoencoders were invented to address the issue of extracting desirable features. Autoencoders take a set of unlabelled inputs, encodes them into short codes, and then uses those to reconstruct the original image, while extracting the most valuable information from the data.



Applications: dimensionality reduction tasks, feature extraction, image recognition.