Neural Networks: Tensorflow

Dr. Amjad Hawash

What is TensorFlow?

- Currently, the most famous deep learning library in the world is Google's TensorFlow.
- Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

What is TensorFlow?

- To give a concrete example, Google users can experience a faster and more refined the search with AI.
- If the user types a keyword a the search bar, Google provides a recommendation about what could be the next word.

What is TensorFlow?

- Google wants to use machine learning to take advantage of their massive datasets to give users the best experience.
- Three different groups use machine learning:
  - Researchers.
  - Data Scientists.
  - Programmers.
- They can all use the same toolset to collaborate with each other and improve their efficiency.

What is TensorFlow?

- Google wants to use machine learning to take advantage of their massive datasets to give users the best experience.
- Three different groups use machine learning:
  - Researchers.
  - Data Scientists.
  - Programmers.
- They can all use the same toolset to collaborate with each other and improve their efficiency.

What is TensorFlow?

- TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research.
- It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java.

History of TensorFlow

- A couple of years ago, deep learning started to outperform all other machine learning algorithms when giving a massive amount of data.
- Google saw it could use these deep neural networks to improve its services:
  - Gmail.
  - Photo.
  - Google search engine.

History of TensorFlow

- They build a framework called Tensorflow to let researchers and developers work together on an AI model.
- Once developed and scaled, it allows lots of people to use it.
- It was first made public in late 2015, while the first stable version appeared in 2017.
- It is open source under Apache Open Source license.
- You can use it, modify it and redistribute the modified version for a fee without paying anything to Google.

TensorFlow Architecture

- Tensorflow architecture works in three parts:
  - Preprocessing the data.
  - Build the model.
  - Train and estimate the model.
- It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**.
- You can construct a sort of flowchart of operations (called a **Graph**) that you want to perform on that input.

TensorFlow Architecture

- The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.
- This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

Where can Tensorflow run?

- TensorFlow hardware, and software requirements can be classified into:
  - Development Phase:
    - This is when you train the mode.
    - Training is usually done on your Desktop or laptop.
  - Run Phase or Inference Phase:
    - Once training is done Tensorflow can be run on many different platforms.
    - You can run it on:
      - Desktop running Windows, macOS or Linux
      - Cloud as a web service
      - Mobile devices like iOS and Android.

Where can Tensorflow run?

- You can train it on multiple machines then you can run it on a different machine, once you have the trained model.
- The model can be trained and used on GPUs as well as CPUs.
- GPUs were initially designed for video games.
- In late 2010, Stanford researchers found that GPU was also very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations.

Where can Tensorflow run?

- Deep learning relies on a lot of matrix multiplication.
- TensorFlow is very fast at computing the matrix multiplication because it is written in C++.
- Although it is implemented in C++, TensorFlow can be accessed and controlled by other languages mainly, Python.
- Finally, a significant feature of TensorFlow is the TensorBoard.
- The TensorBoard enables to monitor graphically and visually what TensorFlow is doing.

Introduction to Components of TensorFlow: Tensor

- Tensorflow's name is directly derived from its core framework: Tensor.
- In Tensorflow, all the computations involve tensors.
- A tensor is a vector or matrix of n-dimensions that represents all types of data.
- All values in a tensor hold identical data type with a known (or partially known) shape.
- The shape of the data is the dimensionality of the matrix or array.

Introduction to Components of TensorFlow: Tensor

- A tensor can be originated from the input data or the result of a computation.
- In TensorFlow, all the operations are conducted inside a graph.
- The graph is a set of computation that takes place successively.
- Each operation is called an op node and are connected to each other.

Introduction to Components of TensorFlow: Tensor

- The graph outlines the ops and connections between the nodes.
- However, it does not display the values.
- The edge of the nodes is the tensor, i.e., a way to populate the operation with data.

Introduction to Components of TensorFlow: Graphs

- TensorFlow makes use of a graph framework.
- The graph gathers and describes all the series computations done during the training.
- The graph has lots of advantages:
  - It was done to run on multiple CPUs or GPUs and even mobile operating system
  - The portability of the graph allows to preserve the computations for immediate or later use.
  - The graph can be saved to be executed in the future.
  - All the computations in the graph are done by connecting tensors together.

List of Prominent Algorithms supported by TensorFlow

- Currently, TensorFlow 1.10 has a built-in API for:
  - Linear regression: tf.estimator.LinearRegressor
  - Classification:tf.estimator.LinearClassifier
  - Deep learning classification: tf.estimator.DNNClassifier
  - Deep learning wipe and deep: tf.estimator.DNNLinearCombinedClassifier
  - Booster tree regression: tf.estimator.BoostedTreesRegressor
  - Boosted tree classification: tf.estimator.BoostedTreesClassifier

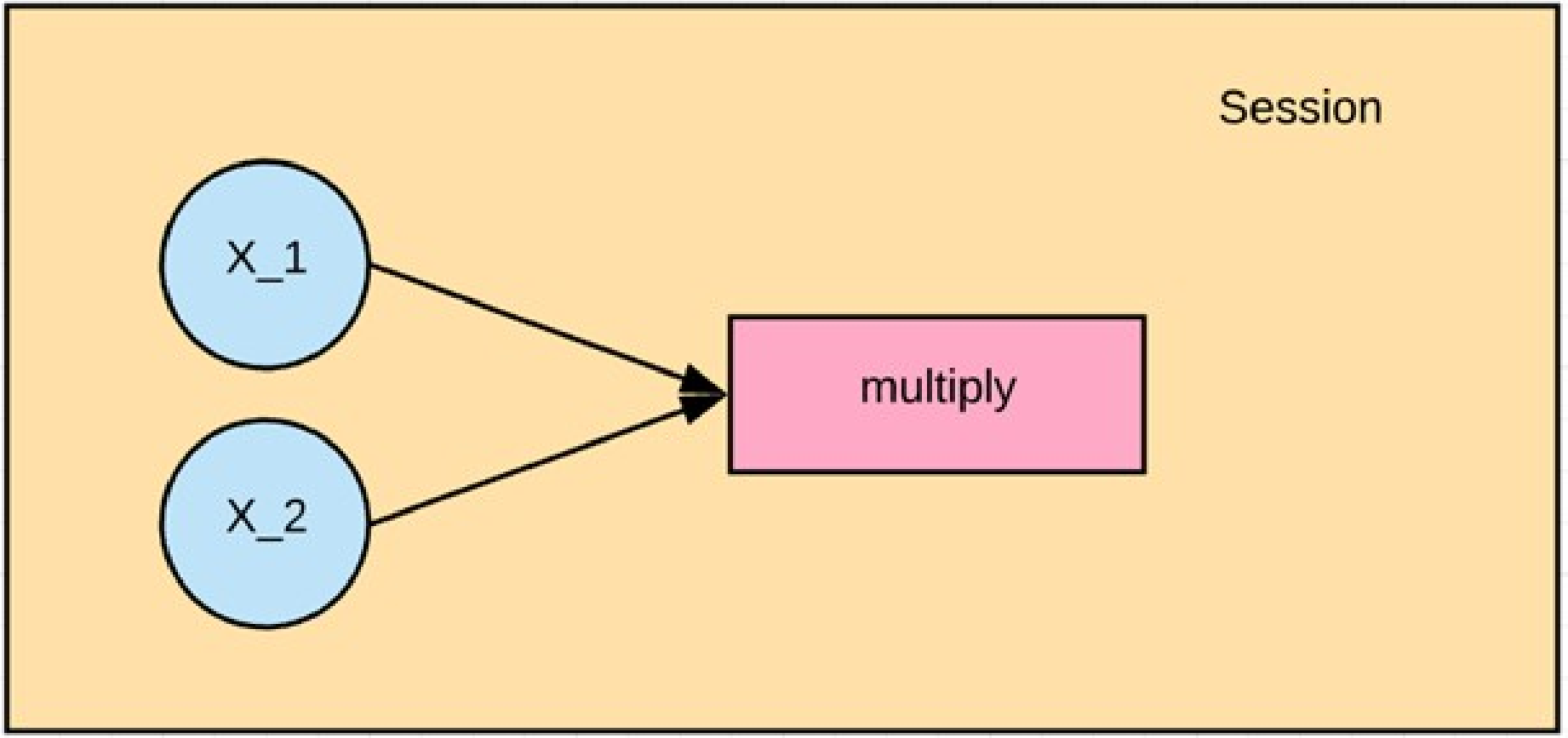
Simple TensorFlow Example

- **import numpy as np**
- **import tensorflow as tf**
- In the first two line of code, we have imported tensorflow as tf.
- Let 's practice the elementary workflow of Tensorflow with a simple example.
- Let 's create a computational graph that multiplies two numbers together.

Simple TensorFlow Example

- During the example, we will multiply X_1 and X_2 together.
- Tensorflow will create a node to connect the operation.
- In our example, it is called multiply.
- When the graph is determined, Tensorflow computational engines will multiply together X_1 and X_2.

Simple TensorFlow Example



Simple TensorFlow Example

- Finally, we will run a TensorFlow session that will run the computational graph with the values of X_1 and X_2 and print the result of the multiplication.
- Let 's define the X_1 and X_2 input nodes.
- When we create a node in Tensorflow, we have to choose what kind of node to create.
- The X1 and X2 nodes will be a placeholder node.
- The placeholder assigns a new value each time we make a calculation.
- We will create them as a TF dot placeholder node.

Step 1: Define the variable

- **X_1 = tf.placeholder(tf.float32, name = "X_1")**
- **X_2 = tf.placeholder(tf.float32, name = "X_2")**
- When we create a placeholder node, we have to pass in the data type will be adding numbers here so we can use a floating-point data type, let's use tf.float32.
- We also need to give this node a name.
- This name will show up when we look at the graphical visualizations of our model.
- Let's name this node X_1 by passing in a parameter called name with a value of X_1 and now let's define X_2 the same way. X_2.

Step 2: Define the computation

- **multiply = tf.multiply(X_1, X_2, name = "multiply")**
- Now we can define the node that does the multiplication operation.
- In Tensorflow we can do that by creating a tf.multiply node.
- We will pass in the X_1 and X_2 nodes to the multiplication node.
- It tells tensorflow to link those nodes in the computational graph, so we are asking it to pull the values from x and y and multiply the result.
- Let's also give the multiplication node the name multiply. It is the entire definition for our simple computational graph.

Step 3: Execute the operation

- To execute operations in the graph, we have to create a session.
- In Tensorflow, it is done by tf.Session().
- Now that we have a session we can ask the session to run operations on our computational graph by calling session.
- To run the computation, we need to use run.

Step 3: Execute the operation

- When the addition operation runs, it is going to see that it needs to grab the values of the X_1 and X_2 nodes, so we also need to feed in values for X_1 and X_2.
- We can do that by supplying a parameter called feed_dict.
- We pass the value 1,2,3 for X_1 and 4,5,6 for X_2.

Step 3: Execute the operation

- We print the results with print(result).
- We should see 4, 10 and 18 for 1x4, 2x5 and 3x6

Step 3: Execute the operation

- X_1 = tf.placeholder(tf.float32, name = "X_1")
- X_2 = tf.placeholder(tf.float32, name = "X_2")
- multiply = tf.multiply(X_1, X_2, name = "multiply")
- with tf.Session() as session:
- result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:[4,5,6]})
- print(result)

Options to Load Data into TensorFlow

- The first step before training a machine learning algorithm is to load the data.
- There is two commons way to load data:
  - Load data into memory:
    - It is the simplest method.
    - You load all your data into memory as a single array. You can write a Python code.
    - This lines of code are unrelated to Tensorflow.
  - Tensorflow data pipeline:
    - Tensorflow has built-in API that helps you to load the data, perform the operation and feed the machine learning algorithm easily.
    - This method works very well especially when you have a large dataset.
    - For instance, image records are known to be enormous and do not fit into memory.
    - The data pipeline manages the memory by itself

Load data in memory

- If your dataset is not too big, i.e., less than 10 gigabytes, you can use the first method.
- The data can fit into the memory.
- You can use a famous library called Pandas to import CSV files.
- You will learn more about pandas in the next tutorial.

Load data with Tensorflow pipeline

- The second method works best if you have a large dataset.
- For instance, if you have a dataset of 50 gigabytes, and your computer has only 16 gigabytes of memory then the machine will crash.
- In this situation, you need to build a Tensorflow pipeline.
- The pipeline will load the data in batch, or small chunk.

Load data with Tensorflow pipeline

- Each batch will be pushed to the pipeline and be ready for the training.
- Building a pipeline is an excellent solution because it allows you to use parallel computing.
- It means Tensorflow will train the model across multiple CPUs.
- It fosters the computation and permits for training powerful neural network.

Create Tensorflow pipeline

- In the example before, we manually add three values for X_1 and X_2.
- Now we will see how to load data to Tensorflow.

Step 1: Create the data

- First of all, let's use numpy library to generate two random values.
- **import numpy as np**
- **x_input = np.random.sample((1,2))**
- **print(x_input)**

Step 2: Create the placeholder

- Like in the previous example, we create a placeholder with the name X.
- We need to specify the shape of the tensor explicitly.
- In case, we will load an array with only two values. We can write the shape as shape=[1,2]
- **# using a placeholder**
- **x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')**

Step 3: Define the dataset method

- next, we need to define the Dataset where we can populate the value of the placeholder x.
- We need to use the method tf.data.Dataset.from_tensor_slices
- **dataset = tf.data.Dataset.from_tensor_slices(x)**

Step 4: Create the pipeline

- In this step, we need to initialize the pipeline where the data will flow.
- We need to create an iterator with make_initializable_iterator.
- We name it iterator. Then we need to call this iterator to feed the next batch of data, get_next. We name this step get_next.
- Note that in our example, there is only one batch of data with only two values.

Step 4: Create the pipeline

- iterator = dataset.make_initializable_iterator()
- get_next = iterator.get_next()

Step 5: Execute the operation

- The last step is similar to the previous example.
- We initiate a session, and we run the operation iterator. We feed the feed_dict with the value generated by numpy.
- These two value will populate the placeholder x.
- Then we run get_next to print the result.

Step 5: Execute the operation

- with tf.Session() as sess:
- # feed the placeholder with data
- sess.run(iterator.initializer, feed_dict={ x: x_input })
- print(sess.run(get_next))