6. - Deep Learning Frameworks

6.1 - Deep Learning Frameworks and TensorFlow 2.x

Rodolfo Sabino

10 Setembro, 2020

Departamento de de Computação

HUAWEI

Disclaimer

The following content is heavily based on HCIA-AI Course material by Huawei Technologies Co., Ltd., authored by Zhang Luxiao and Yuan Meng. Distribution is not allowed.

Index

- Deep Learning Frameworks
- TensorFlow Overview
- TensorFlow 2.x Basics
- TensorFlow 2.x Common Modules

Deep Learning Frameworks

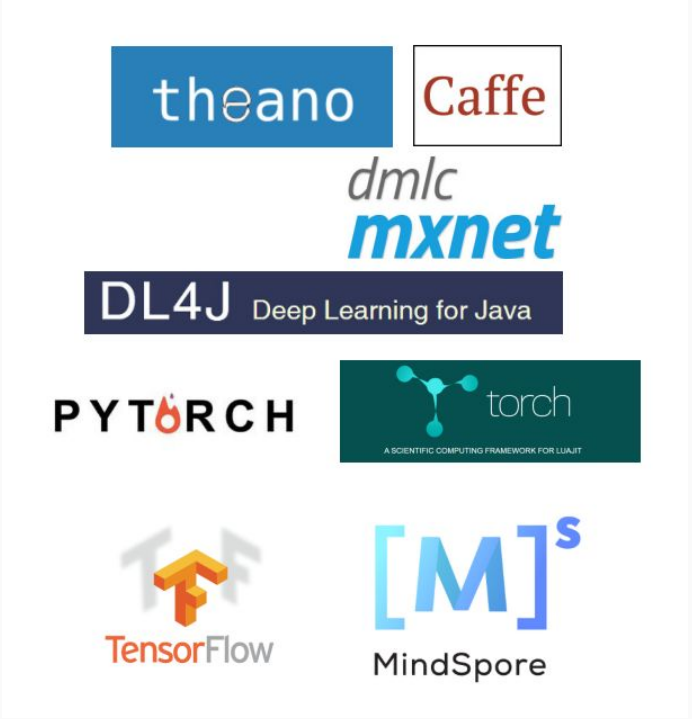
Deep Learning Frameworks

Convenient interface, library or tool

High-level, easy-to-use, quick development

Set of building blocks, i.e., algorithms and models: developers assemble suitable blocks, speeding up the process

Lowers requirements for developers, who can reuse optimizers, back-propagation, numerical methods, layers, etc. And focus their effort in customizing what they really need



PyTorch

Python-based framework developed by Facebook based on Torch, a flexible scientific computing framework, i.e., a tensor operation library similar to NumPy, but built in Lua

PyTorch is open source and was developed in python since its syntax is more powerful and popular than Lua to handle tensors

Many institutes and companies use PyTorch, including Twitter, GMU, and Salesforce

Image source: http://PyTorch123.com/FirstSection/PyTorchIntro/

PyTorch: Features

Python first: beyond binding Python to C++, provides fine grained direct Python support, as easy as using NumPy. Both lowes threshold for understanding and ensures code consistency with the native Python implementation.

Dynamic neural network: programs can dynamically build/adjust computation graphs at runtime. This feature is not supported by TensorFlow 1.x and many other mainstream frameworks, for example, so graphs must be fed and ran in TensorFlow 1.x.

Easy to debug: dynamic graphs can be generated during execution. Execution can be stopped in a debugger to inspect nodes and check outputs.

Both CPU and GPU support for computing acceleration

TensorFlow Overview

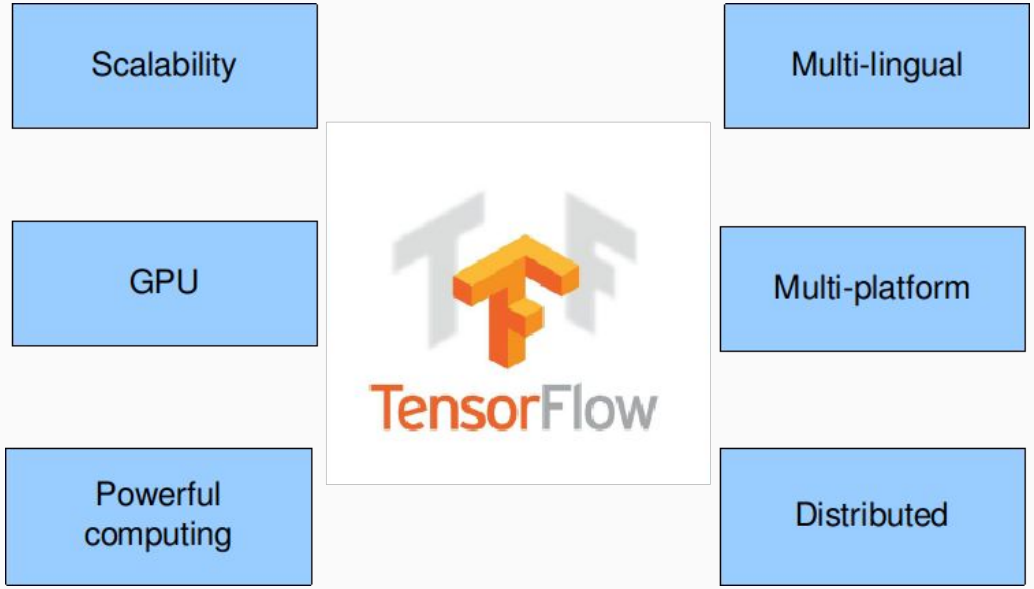
TensorFlow

TensorFlow is an open source library for digital computing used for developing machine learning algorithms.

Google Brain 2nd generation system, after DistBelief.

TensorFlow supports various deep learning algorithms and multiple computing platforms, ensuring high system stability.

TensorFlow 2.x Features



TensorFlow - Distributed Computing

Can run from smartphones and embedded devices to computer clusters

Current native distributed deep learning frameworks include TensorFlow, PyTorch, CNTK, Deeplearning4J, and MXNet

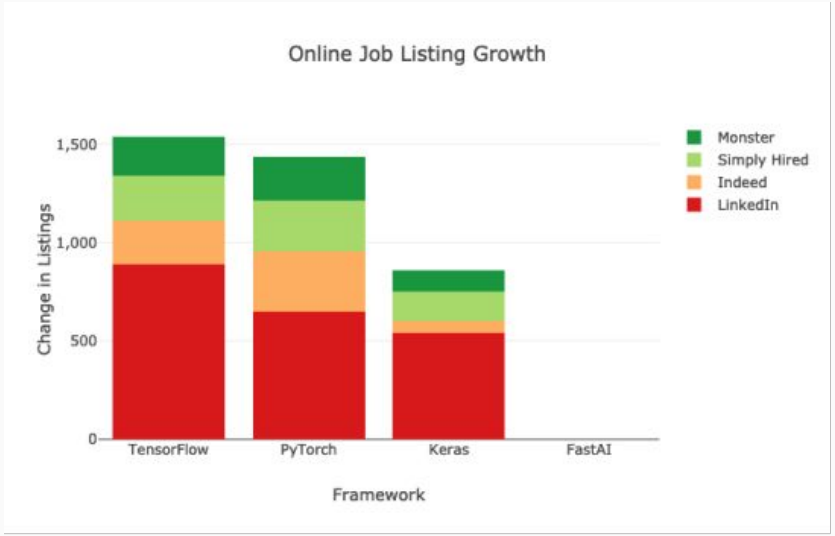
Most frameworks rely on on cuDNN for single GPU usage, so a similar training speed is obtained, provided that the hardware computing capabilities or allocated memories slightly differ.

However, for large-scale deep learning, massive data makes it difficult for the single GPU to complete training in a limited time. To handle such cases, TensorFlow enables distributed training.

Why TensorFlow?

TensorFlow is considered as one of the best libraries for neural networks, and can reduce difficulty in deep learning development. In addition, as it is open-source, it can be conveniently maintained and updated, thus the efficiency of development can be improved.

Keras, ranking third in the number of stars on GitHub, is packaged into an advanced API of TensorFlow 2.0, which makes TensorFlow 2.x more flexible, and easier to debug.



Demand on the recruitment market

TensorFlow 2.x versus TensorFlow 1.x

TensorFlow 1.0 basic concepts:

- Tensors: sets of scalar values formatted into a multidimensional array, similar to NumPy
- Graph: a formal specification of processing as a flow, operations are the graph nodes and tensors are the edges. Tensors flow through the graph as its operations are carried out,
- Operations: usually mathematical, operate over tensors.
- Session: encapsulates the execution environment and underlying resources
- Variables: maintain state across multiple calls, but demand explicit initialization to a random number, a constant, or a computation from other variables

TensorFlow 2.x versus TensorFlow 1.x

After a tensor is created in TensorFlow 1.0, the result cannot be returned directly. To obtain the result, the session mechanism needs to be created, which includes the concept of graph, and code cannot run without session.run. This style is more like the hardware programming language VHDL.

Compared with some simple frameworks such as PyTorch, TensorFlow 1.0 adds the session and graph concepts, which are inconvenient for users.

TensorFlow 1.0 is complex to debug, and its APIs are disordered, making it difficult for beginners. Learners will come across many difficulties in using TensorFlow 1.0 even after gaining the basic knowledge. As a result, many researchers have turned to PyTorch

TensorFlow 2.x versus TensorFlow 1.x

A Session instantiates a graph in an environment which the graph operations are to be executed, allocating the necessary hardware resources to execute the graph.

Prior to TensorFlow 2.0, a session needed to be explicitly deallocated when its no longer in use.

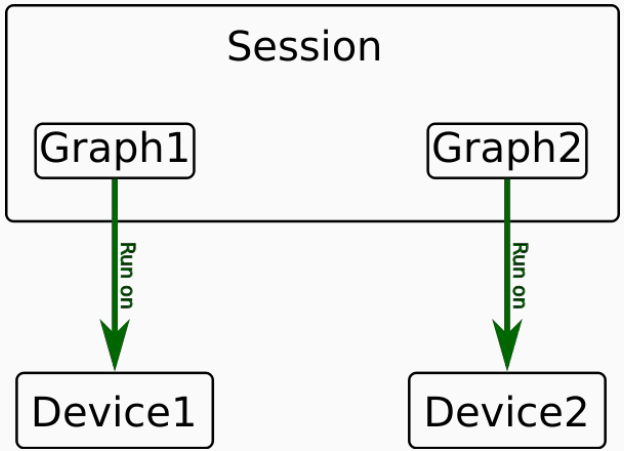


Image source

TensorFlow 2.x versus TensorFlow 1.x

Easy to use Keras API, *what you see is what you get*, just like Python and PyTorch: graph and session mechanisms are removed in TensorFlow 2.x

Major Improvements

Introduction of the **dynamic graph mechanism called Eager Execution**. It allows users to compile and debug models like ordinary programs, making TensorFlow easier to learn and use.

Multiple platforms and programming languages are supported, and compatibility between components can be improved via standardization on exchange formats and alignment of APIs.

Deprecated APIs are deleted and duplicate APIs are reduced to avoid confusion.

Compatibility and continuity: TensorFlow 2.x provides a module enabling compatibility with TensorFlow 1.x.

The tf.contrib module is removed. Maintained modules are moved to separate repositories. Unused and unmaintained modules are removed.

TensorFlow 2.x Basics

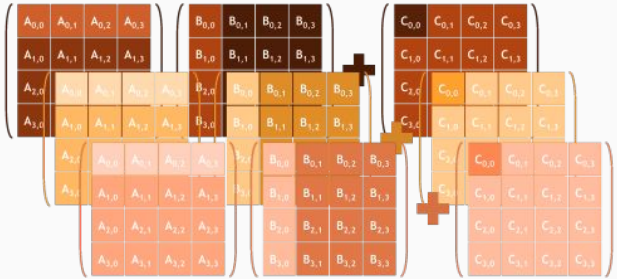
TensorFlow 2.x Basics

The most recent version is TensorFlow 2.3 released in July 2020.

TensorFlow API is available through various languages, including Python, which is the language we will be using in this course.

The main elements of TensorFlow 2.x are

- Tensors
- Operations



TensorFlow 2.x API

Tensorflow computation model is described as data flow graphs.

Edges of the graph are Tensors representing I/O data sets and dependencies,

Nodes, in their turn, represent mathematical operations.

The API provides an abstraction to these structures.

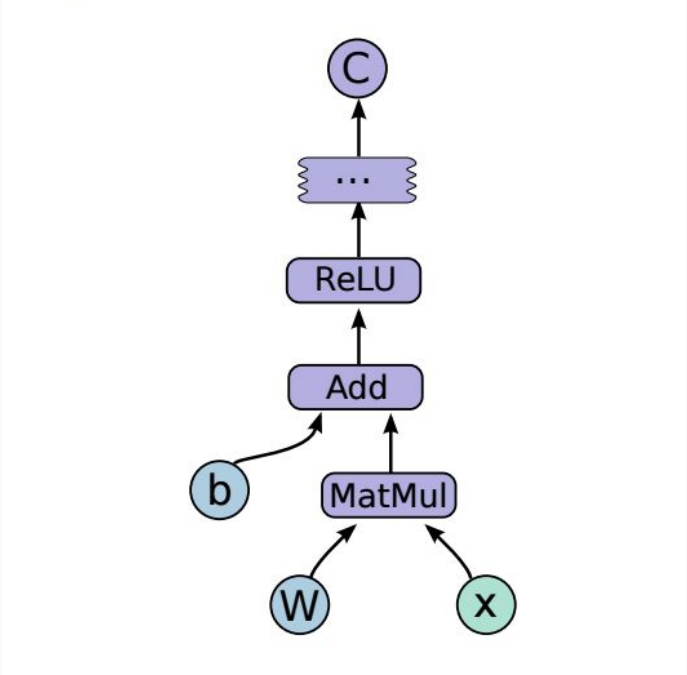


Image source

Computation Graphs

Graphs are used in TensorFlow to represent a function computation with nodes representing operations.

Examples: *Algebraic, activation functions, synchronization, flow control, ...*

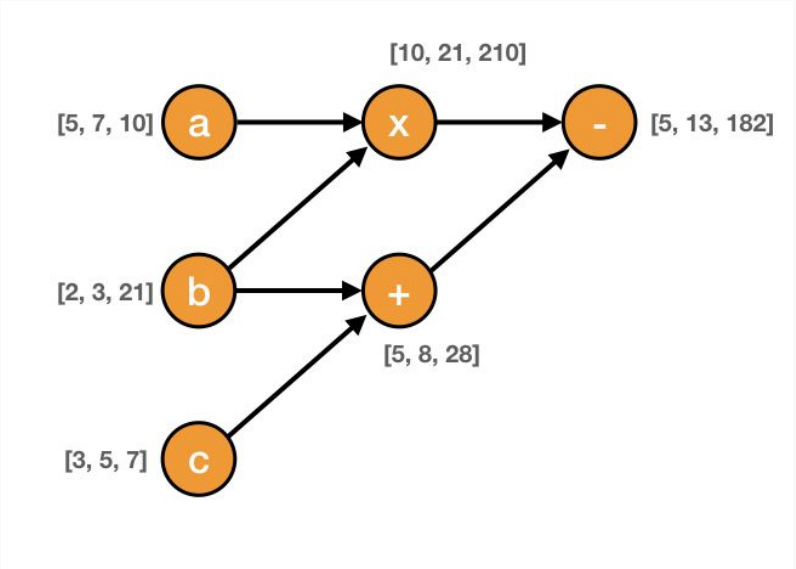


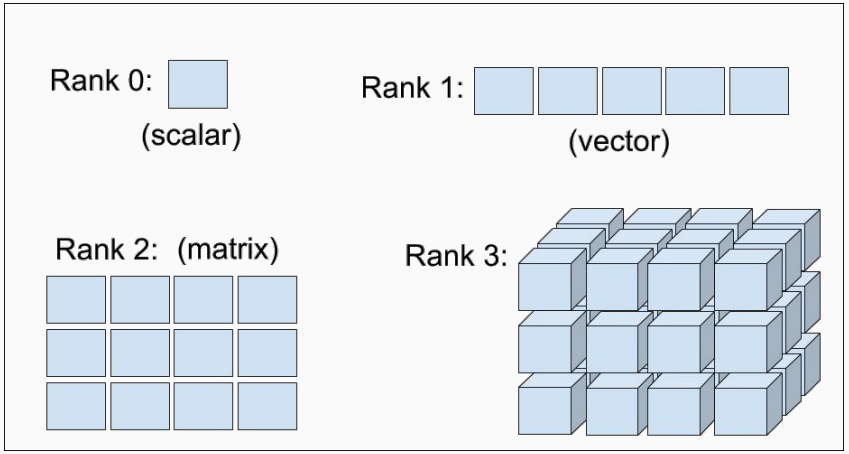
Image source

Tensors

The most basic data structures found in TensorFlow, which encapsulates all data

Tensors are multidimensional arrays with a name, a uniform data type, shape, and rank

- A scalar is a rank-0 tensor
- A vector is a rank-1 tensor
- A matrix is a rank-2 tensor
- A “cube” is a rank-3 tensor



In TensorFlow 2.x, tensors are classified into Constants and Variables

Basic Operations in TensorFlow 2.x

TensorFlow 2.x provides a number of APIs for performing operations over tensors, of which stand out:

- Methods for creating constants and variables
- Tensor slicing and indexing
- Dimension changes of tensors
- Arithmetic operations on tensors
- Tensor concatenation and splitting
- Tensor sorting

Eager Execution Mode

Static graph: used by TensorFlow 1.x (graph mode in 2.x) separates computation definition and execution by using computational graphs. This is a declarative programming model. In graph mode, developers need to build a computational graph, start a session, and then input data to obtain an execution result.

Static graphs are advantageous in distributed training, performance optimization, and deployment, but inconvenient for debugging. Executing a static graph is similar to invoking a compiled C language program, and internal debugging cannot be performed in this case. Therefore, eager execution based on dynamic computational graphs emerges.

Eager execution is a command-based, imperative programming model, which is the same as native Python. A result is returned immediately after an operation is performed.

AutoGraph

Eager Execution is enabled in TensorFlow 2.x by default. It is intuitive and flexible for users (easier and faster to run a one-time operation), but may compromise performance and deployability.

To achieve optimal performance and make a model deployable anywhere, you can run @tf.function to add a decorator to build a graph from a program, making Python code more efficient.

tf.function can build a TensorFlow operation in the function into a graph. In this way, this function can be executed in graph mode. Such practice can be considered as encapsulating the function as a TensorFlow operation of a graph.

HUAWEI

TensorFlow 2.x Common Modules

API Reference

The TensorFlow website provides a comprehensive list with functions with their use.

https://www.tensorflow.org/api_docs/

It also provides a series of tutorials and guides on how to use the API.

https://www.tensorflow.org/overview

TensorFlow 2.x Common Modules 1 of 2

tf: common arithmetic operations, such as **tf.abs** (calculating an absolute value), **tf.add** (adding elements one by one), and **tf.concat** (concatenating tensors). Most operations in this module can be performed by NumPy.

tf.errors: error type module of TensorFlow

tf.data: implements operations on datasets. Input pipes created by **tf.data** are used to read training, evaluation, and prediction data. In addition, data can be easily input from memories (such as NumPy).

tf.distributions: implements a myriad of statistical distributions (such as Bernoulli distribution, uniform distribution, and Gaussian distribution)

TensorFlow 2.x Common Modules 2 of 2

tf.io.gfile: unblocking file operations, such as file I/O, copying, and renaming.

tf.image: implements operations on images, including image processing functions. This module is similar to OpenCV, and provides functions related to image luminance, saturation, phase inversion, cropping, resizing, image format conversion (RGB to HSV, YUV, YIQ, or gray), rotation, and sobel edge detection.

tf.keras: a Python API for invoking Keras tools. This is a comprehensive module that enables various network operations, units, and layers.

Keras Interface

TensorFlow 2.x recommends Keras for network building, so tf.keras.layers include many common neural networks

Keras is a high-level API used to build and train deep learning models. It can be used for rapid prototype design, advanced research, and production. Keras has the following three advantages:

- Easy to use: provides simple and consistent GUIs optimized for common cases. It provides practical and clear feedback on user errors.
- Modular and composable: models are built by connecting configurable building blocks together, with little restriction.
- Easy to extend: building blocks can be customized to express new research ideas, create layers and loss functions, and develop advanced models.

Common Keras Methods and Interfaces

Many useful methods, interfaces, and submodules can be found the tf.keras module, of which stand out:

- Dataset processing: dataset manipulation and preprocessing, including tf.keras.datasets, which supports for automatic downloading and creation of well-known datasets
- Neural network model creation: Sequential, Model, Layers...
- Network learning configuration: compile, Loss functions, Metrics, and Optimizers
- Network training and evaluation: fit, fit_generator, and evaluate

Thank You!

Next: 6.2 Deep Learning using TensorFlow

Data Types

Data Types

A tensor is multi-dimensional array made from elements the same type.

***Examples**. float_, int_, bool, string, ...*

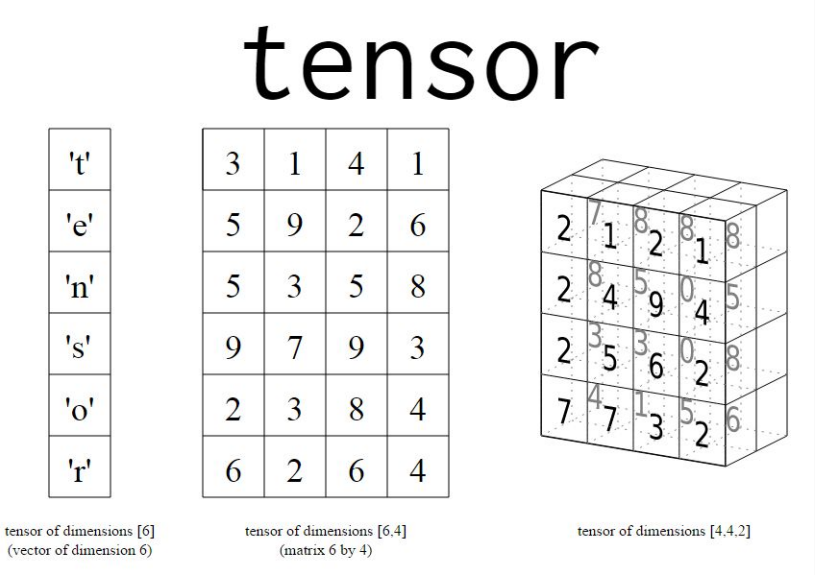


Image source

Operators and Functions

Operators and Functions

The feed operator connects a tensor to a node in the graph. This operation is used during graph execution. Feed is is executed in the form of "run" or "eval" commands.

Reference: https://www.tensorflow.org/api_docs/java/org/tensorflow/Session.Run

Operators and Functions

The fetch operator retrieves the result of a graph execution. This operation is used after graph execution. The fetched tensors are returned by the "run" command on the session object.

Reference: https://www.tensorflow.org/api_docs/java/org/tensorflow/Session.Run

Operators and Functions

A variable maintains state across multiple calls to run during graph execution. The variable can be set to a random number, constant or calculated based on initial values of other variables.

Reference: https://www.tensorflow.org/guide/variable

Operators and Functions

Modules

- **tf.nn:** Used to create Recurrent Neural Networks.
- **tf.estimator:** Used to create one or more input functions.
- **tf.Layers:** Encapsulates variables and operations.
- **tf.Contrib:** Misc. Functions for computing streaming metrics.

Reference: https://www.tensorflow.org/guide/intro_to_modules

TensorFlow Installation

TensorFlow Installation

TensorFlow can be installed locally via pip

```
pip3 install tensorflow
```

It can also run on the web via Google's collab with no setup required at all

https://blog.tensorflow.org/2018/05/colab-easy-way-to-learn-and-use-tensorflow.html

Next