**1. Tensorflow**

**- What is tensor?**

텐서(tensor)란 매우 수학적인 개념\*인데, 간단히 말하자면 데이터의 배열이라고 생각하면 편함. 텐서는 기본적으로 Rank, Shapes,Type으로 구성되어 있다.

텐서의 Rank는 간단히 말해서 몇 차원 배열이냐를 의미한다. 0차원의 경우 Scalar (magnitude only)라고 불리우며 1차원은 Vector (magnitude and direction)이라 하며, 2차원은 Matrix(table of numbers), 3차원은 3-Tensor(cube of number), 마지막으로 N차원은 n-Tensor(n 차원 배열)이라고 한다.

텐서의 Shape은 각 축이 몇 개의 Element들로 구성되어 있는지를 나타내는 값이다. Rank가 1인 텐서인 경우 1차원 배열인 vector가 shape이다. 즉, shape=3는 Element가 3개인 벡터를 의미한다. Shape이 [2,3]이면 Rank가 2인 텐서, 즉 2\*3 행렬을 의미한다.

텐서의 type은 tf.flaot32, tf.int32 와 같이 나타나는 것을 말한다.

\*3차원 공간에 있어서 9개의 성분을 가지며, 좌표 변환에 의해 좌표 성분의 곱과 같은 형의 변환을 받는 양. 예를 들면, 물체의 관성 모멘트나 변형은 이것으로 표시됨. 선형대수학과 물리학에서 텐서는 선형관계를 나타내는 미분기하학의 대상이며 스칼라곱과 선형변환이 기본적인 텐서의 예이다. 스칼라와 벡터 또한 해당된다. 물리학에서 자연 현상의 설명하기 위하여 도입된 개념이며 도입된 자표계와 무관하게 유일무이하게 자연현상을 기술하기 위해서 도입되었다.

* **Python** is the most convenient client language for working with TensorFlow. However, there are also experimental interfaces available in JavaScript, C ++, Java and Go, C # and Julia.
* TF takes into account not only powerful computing clusters but also the ability to run models on mobile platforms like **iOS and Android**. From Linux to Android.
* **TF needs a lot of coding**. It will not give you powerful AI overnight, it’s just a tool for deep learning research that will hopefully make it a bit less cumbersome. You need to think carefully about the architecture of the neural network, correctly assess the dimension and volume of input and output data.
* TF operates with a **static computation graph**. That is, we first define the graph, then we run the calculations and, if we need to make changes to the architecture, we re-train the model. Such an approach was chosen for the sake of efficiency, but many modern neural network tools are able to take into account refinements in the learning process without a significant loss in learning speed. In this regard, the main competitor of TensorFlow is the **PyTorch**.

**What Is It Good For?**

* It’s handy for creating and experimenting with deep learning architectures, and its formulation is convenient for data integration such as inputting graphs, SQL tables, and images together.
* it is backed by Google which guarantees it will stay around for a while, hence it makes sense to invest time and resources to learn it.
* Tensorflow is clearly great any way you look at it, but what about other instruments? Let’s compare and see if TF will be in the lead.
* System architecture is flexible, so it is possible to perform computations on CUPs or GPUs
* The main concept of tensorflow : Flow graphs usage. Nodes of the graph reflect mathematical operations, while the edges represent multidimensional data arrays (tensors) communicated between them / Intermediate-level developers and for developing production models that need to quickly process vast data sets
* NLP application, text classification, summarization, speech recognition, tagging and so on
* TFLearn : As Tensorflow is a low-level API, many high-level APIs were created to run on top of it to make the user experience faster and more understandable. TFLearn is one of these tools that runs on CPU and GPU. It has a special graph visualization tool with details about weights, gradients, activations, and so on. The library is already used for sentiment analysis, text generation, and named entity recognition. It lets users work with convolutional neural networks and recurrent neural networks (LSTM).
* Strengths : Flexibility, Contains several ready-to-use ML models and ready-to-run application packages, Scalability with hardware and software, Large online community
* Weaknesses : Supports only NVIDIA GPUs, A slightly steep learning Curve. There are numerous components that go into making TensorFlow. The two standout ones are:

1. **TensorBoard:** Helps in effective data visualization using data flow graphs
2. **TensorFlow:** Useful for rapid deployment of new algorithms/experiments

The flexible architecture of TensorFlow enables us to deploy our deep learning models on one or more CPUs (as well as GPUs). Below are a few popular use cases of TensorFlow:

1. Text-based applications: Language detection, text summarization
2. Image recognition: Image captioning, face recognition, object detection
3. Sound recognition
4. Time series analysis
5. Video analysis

* It provides both high and low level APIs. And it has high performance with large datasets. But debugging is so difficult.
* 먼저 그래프 모델을 구축하고 나서 나중에 값을 넣는 방식으로 모델이 구축되기 때문에 그래프 모델을 제대로 이해해야 제대로 구축이 가능함. 이러한 이유로 매우 어렵게 느껴짐

**2. PyTorch**

**-What is Torch?**

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

A summary of core features:

* a powerful N-dimensional array
* lots of routines for indexing, slicing, transposing, …
* amazing interface to C, via LuaJIT
* linear algebra routines
* neural network, and energy-based models
* numeric optimization routines
* Fast and efficient GPU support
* Embeddable, with ports to iOS and Android backends

## **Why Torch?**

The goal of Torch is to have maximum flexibility and speed in building your scientific algorithms while making the process extremely simple. Torch comes with a [**large ecosystem of community-driven packages**](https://github.com/torch/torch7/wiki/Cheatsheet) in machine learning, computer vision, signal processing, parallel processing, image, video, audio and networking among others, and builds on top of the Lua community.

At the heart of Torch are the popular neural network and optimization libraries which are simple to use, while having maximum flexibility in implementing complex neural network topologies. You can build arbitrary graphs of neural networks, and parallelize them over CPUs and GPUs in an efficient manner.

## **Using Torch**

Start with our [**Getting Started**](http://torch.ch/docs/getting-started.html) guide to download and try Torch yourself. Torch is open-source, so you can also start with the code on the [**GitHub repo**](https://github.com/torch/torch7).

Torch is constantly evolving: it is already used within Facebook, Google, Twitter, NYU, IDIAP, Purdue and several other companies and research labs.

The primary software tool for deep learning after Tensorflow is PyTorch. The PyTorch framework was developed for Facebook services but is already used for its own tasks by companies like Twitter and Salesforce.

* Unlike TensorFlow, the PyTorch library operates with a **dynamically updated graph**. This means that it allows you to make changes to the architecture **in the process.**
* In PyTorch, you can use standard **debuggers**, for example, pdb or PyCharm.

#### What Is It Good For?

* The process of training a neural network is simple and clear. At the same time, PyTorch supports the data parallelism and distributed learning model, and also contains many pre-trained models.
* PyTorch is much better suited for small projects and prototyping. When it comes to cross-platform solutions, TensorFlow looks like a more suitable choice. However, it is worth noting that for the same tasks, the Caffe2 mobile framework introduced in 2017 can be used.
* Fast and flexible neural network framework with an imperative paradigm.
* It builds networks on a tape-based auto-grad system and provides tensor computation with strong GPU acceleration. Recurrent neural networks are mostly used in PyTorch for machine translation, classification, text generation, tagging, and other NLP tasks.
* People predict it will soon become a go-to alternative to many other frameworks

Strengths:

* Coding is easy, so it has a flatter learning curve
* Supports dynamic graphs so you can adjust on-the-go.
* Supports GPU acceleration

Weaknesses:

* Quite new, so it has a smaller community and fewer resources available online
* Pytorch is being lauded particularly by beginners, mostly due to its easy-to-write code – but the framework is basically a blend of both high and low-level APIs.
* It features a number of pre-trained models and when coding in Pytorch, **you don’t need to categorize numbers into ‘int’, ‘short’, or ‘double’ data types**, like other coding languages. This makes the performance of operations and functions on this framework more intuitive compared to other options (In tensorflow, about 20% of error is data type error)
* The highlight of this framework, though, is that it offers developers **the ability to use dynamic graphs**. **Tensorflow**, Theano, and their derivatives allow you to create only static graphs, so **you have to define the whole graph for the model before you can run it**. However**, in Pytorch,** you can **define or adjust your graph during runtime**, so it’s more flexible and allows you to use variable length inputs, especially in your RNNs. The framework also provides strong support for GPU acceleration, so you get both efficiency and speed.
* PyTorch is a port to the Torch deep learning framework which can be sued for building deep neural networks and executing tensor computations. Torch is a Lua-based framework whereas PyTorch runs on Python.
* PyTorch is a python package which provides tensor computations. Auto-grad package of PyTorch builds computation graphs from tensors and automatically computers gradients.

You can work on all sorts of deep learning challenges using PyTorch, including:

* Images (Detection, Classification, etc.)
* Text (NLP)
* Reinforcement Learning
* And the installation steps vary depending on operating system.
* Pytorch, on the other hand, is a **lower-level API** focused on direct work with array expressions. It has gained immense interest in the last year, becoming a preferred solution for academic research, and applications of deep learning requiring optimizing custom expressions.
* **Pytorch** on the other hand has **better debugging capabilities**
* PyTorch are used for **high performance** models and **large** **datasets** that require fast execution.

**3. Keras**

Keras is a machine learning framework that might be your new best friend if you have a lot of data and/or you’re after the state-of-the-art in AI: deep learning. Plus, it’s the most minimalist approach to using TensorFlow, Theano, or CNTK is the high-level Keras shell.

* Keras is usable as a high-level API on top of other popular lower level libraries such as Theano and CNTK in addition to Tensorflow.
* Prototyping here is facilitated to the limit. Creating massive models of deep learning in Keras is reduced to single-line functions. But this strategy makes Keras a less configurable environment than low-level frameworks.
* Based python API that runs on CPU or GPU.

#### What Is It Good For?

* Keras is the best Deep Learning framework for those who are just starting out. It’s ideal for learning and prototyping simple concepts, to understand the very essence of the various models and processes of their learning.
* Keras is a beautifully written API. The functional nature of the API helps you completely and gets out of your way for more exotic applications. Keras does not block access to lower level frameworks.
* Keras results in a much more readable and succinct code.
* Keras model Serialization/Deserialization APIs, callbacks, and data streaming using Python generators are very mature.

By the way, you cannot compare Keras and Tensorflow because they sit on different levels of abstraction.

PS: Tensorflow is on the **Lower Level**: This is where frameworks like MXNet, Theano, and PyTorch sit. This is the level where mathematical operations like Generalized Matrix-Matrix multiplication and Neural Network primitives like Convolutional operations are implemented.

Keras is on the **higher Level**. At this Level, the lower level primitives are used to implement Neural Network abstraction like Layers and models. Generally, at this level, other helpful APIs like model saving and model training are also implemented.

**4. Sci-Kit Learn**

Sci-kit Learn is a library that features a host of the classical machine learning algorithms like Support Vector Machines (SVMs), KNN Maps, K-Nearest Neighbors (KNN) classifiers, Random Forests, and regression algorithms. It includes options for both supervised and unsupervised learning. Thus, it’s ultimately an effective tool for statistical modeling. It has been built on many other Python libraries like SciPy, Numpy, and Matplotlib, and some of its core algorithms are also written using Cython.

Strengths:

* Great for beginners and for those looking to explore machine learning algorithms
* Good for data-mining and simple projects like predictions on small or labeled data sets

Weaknesses:

* Does not support ANNs
* Does not support GPU computing

What sets this framework apart from others is an easy-to-use interface for developers and a high level of abstraction that allows especially beginners in machine learning to get easily acquainted with the platform, without having to deal with the nitty-gritty of actual algorithms.

It’s easy to run and debug, and there are some nice and easy tutorials available to help understand the algorithms when you do have to work with them. However, Sci-kit Learn does have a couple of limitations.

Firstly, it does not support Artificial Neural Networks. Secondly, it’s only suitable for small projects with small datasets, and for tasks that are not particularly computationally intensive. This is mainly due to the fact that the framework does not support GPU computing. For more seasoned or hard-core developers, it can feel limiting to some extent, as the abstraction doesn’t allow for fine tuning the underlying algorithms.

**The other**

1. **Sonnet** : Sonnet deep learning framework built on top of TensorFlow. It is designed to create neural networks with a complex architecture by DeepMind.

* High-level object-oriented libraries that bring about abstraction when developing neural networks (NN) or other machine learning (ML) algorithms.
* The idea of Sonnet is to construct the primary Python objects corresponding to a specific part of the neural network. Further, these objects are independently connected to the computational TensorFlow graph. Separating the process of creating objects and associating them with a graph simplifies the design of high-level architectures. More information about these principles can be found in the framework documentation.

#### What Is It Good For?

* The main advantage of Sonnet, is you can use it to reproduce the research demonstrated in DeepMind’s papers with greater ease than Keras, since DeepMind will be using Sonnet themselves.

So all-in-all, it’s a flexible functional abstractions tool that is absolutely a worthy opponent for TF and PyTorch.

2. **MXNet :** MXNet is a highly scalable deep learning tool that can be used on a wide variety of devices. Although it does not appear to be as widely used as yet compared to TensorFlow, MXNet growth likely will be boosted by becoming an Apache project.

* The framework initially supports a large number of languages (C ++, Python, R, Julia, JavaScript, Scala, Go, and even Perl).
* The main emphasis is placed on the fact that the framework is very effectively parallel on multiple GPUs and many machines. This, in particular, has been demonstrated by his work on Amazon Web Services.

#### What Is It Good For?

* Support of multiple GPUs (with optimized computations and fast context switching)
* Clean and easily maintainable code (Python, R, Scala, and other APIs)
* Fast problem-solving ability (vital, for newbies in deep learning, like me)

Although it is not so popular as TF, MXNet has detailed documentation and is easy to use, with the ability to choose between imperative and symbolic programming styles, making it a great candidate for both beginners and experienced engineers.

3. **Gluon :** Glun is one more great Deep Learning framework that can be used to create simple as wells as sophisticated models.

* The specificity of the Gluon project is a flexible interface that simplifies prototyping, building and training deep learning models without sacrificing learning speed.
* Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
* Similar to PyTorch, the Gluon framework supports work with a **dynamic graph**, combining this with high-performance **MXNet**. From this perspective, Gluon looks like an extremely interesting alternative to Keras for distributed computing.

#### What Is It Good For?

* In Gluon, you can define neural networks using the simple, clear, and concise code.
* It brings together the training algorithm and neural network model, thus providing flexibility in the development process without sacrificing performance.
* Gluon enables to define neural network models that are dynamic, meaning they can be built on the fly, with any structure, and using any of Python’s native control flow.

4. **Swift** : If you are into programming, when you hear Swift, you will probably think about app development for iOS or MacOS. If you’re into deep learning, then you must have heard about Swift for Tensorflow (abbreviated as S4TF).

By integrating directly with a general purpose programming language, Swift for TensorFlow enables more powerful algorithms to be expressed like never before.

* **First-class autodiff**. Differentiable programming gets first-class support in a general-purpose programming language. Take derivatives of any function, or make custom data structures differentiable at your fingertips.
* **Next-generation APIs**. New APIs informed by the best practices of today, and the research directions of tomorrow, are both easier to use and more powerful.
* **Building on TensorFlow**, the Swift APIs give you transparent access to all low-level TensorFlow operators.
* **High-quality tooling.** Building upon Jupyter and LLDB, Swift in Colab improves your productivity with helpful tooling such as context-aware autocomplete.

#### What Is It Good For?

* A great choice if dynamic languages are not good for your tasks. If you have a problem arises when you have training running for hours and then your program encounters a type error and it all comes crashing down, enter Swift, a statically typed-language. Here you will know ahead of any line of code running that the types are correct.

5. **Chainer :** Until the advent of DyNet at CMU, and PyTorch at Facebook, Chainer was the leading neural network framework for dynamic computation graphs or nets that allowed for input of varying length, a popular feature for NLP tasks

* The code is written in pure **Python** on top of the Numpy and CuPy libraries. Chainer is the first framework to use a dynamic architecture model (as in PyTorch).
* Chainer several times beat records on the effectiveness of scaling when modeling problems solved by neural networks.

#### What Is It Good For?

* By its own benchmarks, Chainer is notably faster than other Python-oriented frameworks, with TensorFlow the slowest of a test group that includes MxNet and CNTK.
* Better GPU & GPU data center performance than TensorFlow. (TensorFlow is optimized for TPU architecture) Recently, Chainer became the world champion for GPU data center performance.
* Good Japanese support.
* OOP like programming style.

6. **DL4J** : Those who are on a short leg with Java or Scala should pay attention to DL4J (short for Deep Learning for Java).

* Training of neural networks in DL4J is carried out in parallel through iterations through clusters.
* The process is supported by **Hadoop and Spark** architectures.
* Using **Java** allows you to use the library in the development cycle of programs for Android devices.

#### What Is It Good For?

* A very good platform if you are looking for a good Deep Learning Framework in Java.

7. **ONNX** : Those who are on a short leg with Java or Scala should pay attention to DL4J (short for Deep Learning for Java).

* Training of neural networks in DL4J is carried out in parallel through iterations through clusters.
* The process is supported by **Hadoop and Spark** architectures.
* Using **Java** allows you to use the library in the development cycle of programs for Android devices.

#### What Is It Good For?

* A very good platform if you are looking for a good Deep Learning Framework in Java.