Machine Learning and AI Frameworks: What’s the Difference and How to Choose?
There are many machine learning frameworks. Given that each takes much time to learn, and given that some have a wider user base than others, which one should you use? Here we look briefly at some of the major ones.

In picking a tool, you need to ask what is your goal: machine learning or deep learning? Deep learning has come to mean using neural networks to do, for the most part it seems, image recognition. These can be used to solve other problems too—like predicting what word will come next when one uses a Swype keyboard (That one works very well.)—but most of the tutorials, use cases, and even engineering in these newer frameworks seems targeted toward building the framework that will train itself on image databases in the fastest time, using the least amount of memory, and run on both GPUs and CPUs.

But not all problems are image classification problems. If they were then one could ask the logical question of why require the programmer to handle all the lower level details? Why not provide just one overarching API, say, the image classification API, and let data scientists simply drop image databases into that? Or provide that as a web service, like Google’s natural language web service (which we will discuss in an upcoming post.).

Other data scientists are interested in more than just handwriting recognition. They are interested in tools to solve problems applicable to business, like linear and logistic regression, k-mean clustering, and, yes, neural networks.

Some of the ML frameworks available are.

- Spark ML
- TensorFlow
- Caffe
- Torch
- Mahout
Some of these are more mathematically oriented, and thus geared more to statistical and other modeling than neural networks. Mahout, TensorFlow, and Mahout provide a rich set of linear algebra tools, whereas Caffe is focused on deep learning. TensorFlow does that too but it also does regression analysis, as we show here. Scikit-learn has been around a long time and would be most familiar to R programmers. But it is not built to run across a cluster. Spark ML, of course, is built for running on a cluster, since that is what Spark is all about. In other words, it can handle really large matrix multiplication by taking slices of the matrix and running that calculation on different servers. (Every grammar school kid knows, or should know, that multiplication and addition are associative and distributive.) Matrix multiplication is the most important operation for most ML. That requires a distributed architecture so the computer does not run out of memory and does not take too long to run when working with large amounts of data.

TensorFlow

TensorFlow was developed at the Google Brain research facility and then made into an open source project. It does regression, classification, neural networks, etc. and runs both on CPUs and GPUs.

But TensorFlow is very complex, much more complicated than Spark ML. TF requires you understand Numpy arrays intimately. Numpy is a Python framework for working with n-dimensional arrays (A 1-dimensional array is a vector. A 2-dimensional array is a matrix, and so forth.) Instead of doing things like converting arrays to one-hot vectors (a true-false representation) you are expected to do that yourself.

But it has a rich set of, for example, activation functions
for neural networks. This means it does all the hard work of statistics.

If we define **deep learning** as the ability to do neural networks, then TensorFlow does that. But it also hands more every day problems, like regression.

**Caffe**

Caffe seems to focus mainly on image classification and voice recognition. It, like TensorFlow, scales, as it can run on clusters. Caffe is written in C++, for speed. You can work with it from the command line or with Python (pycaffe) or Matlab (matcaffe).

Caffe is complicated. That it works with Matlab means it will be comfortable for those graduate students and data scientists who have for many years been using that large and comprehensive mathematical computing platform. Even many high school students have been exposed to that as it does things like integration and differentiation and solve differential equations.

**Spark ML**

We have [written at length](https://example.com) about how to use Spark ML. This is complicated too, but instead of having to work with Numpy arrays it lets you work with Spark RDD data structures, which anyone using Spark in its big data role will understand. And you can use it to work with Spark SQL dataframes, which most Python programmers know. So it creates dense and spark feature-label vectors for you thus taking away some of the complexity of preparing data to feed into the ML algorithms.
Torch

Torch says it is the easiest ML framework. That’s like saying it’s the easiest way to learn ancient Greek, as there is nothing easy about that.

Torch’s relative simplicity comes from its Lua programming language interface (There are other interfaces, like QT, and iPython/Jupyter.). Lua is indeed simple. There are no floats or integers, just numbers. And all objects in Lua are tables. So it’s easy to create data structures. And it provides a rich set of easy-to-understand features to slice tables and add to them.

Like TensorFlow, the basic data element in Torch is the tensor. You create one of those just by writing torch.Tensor.

The command line interface provides inline help and it helps with indentation. People who have used Python will be relieved there as this means you can type functions in situ without having to start over at the beginning when you make a mistake. And for those who like complexity and sparse code, it supports functional programming.

Beyond that, Torch still has a steep learning curve. But that cannot be avoided as using ML requires understanding maths and statistics. But this is inherent in the entirety of the ML beast. If you do not know what a gradient is then you need to spend a few years learning advanced maths first.