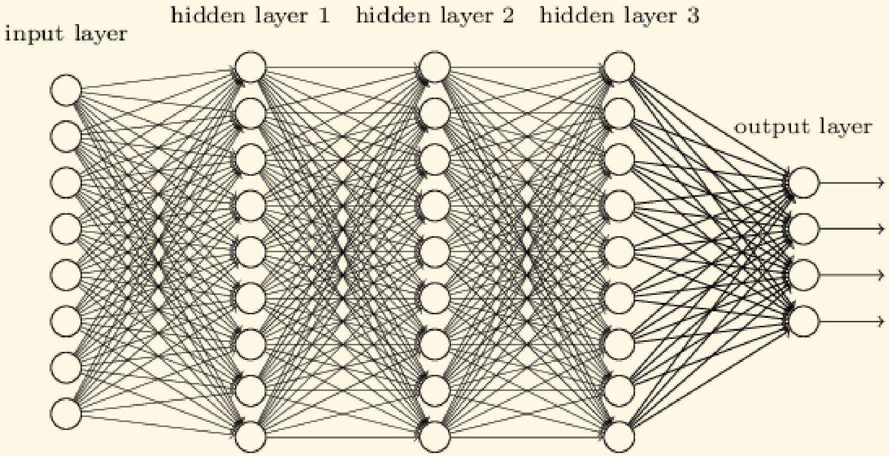
Lecture 16
Neural Network Software

28 March 2016

Taylor B. Arnold
Yale Statistics
STAT 365/665

Housekeeping:

- Problem set 6 is online and due **next** Friday, April 8th
- Problem sets 7,8, and 9 will be due on the remaining Fridays
- No class on April 11th or 13th
- Grades for problem sets 3-5 are online
- TA assignments have been slightly updated; see ClassesV2


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

For an input vector x and a response y, we can view a neural network as simply being something like:

$$\begin{align} z^1 &= W^1 a^0 + b^1 \tag{1}\\ a^1 &= \sigma(z^1) \tag{2}\\ z^2 &= W^2 a^1 + b^2 \tag{3}\\ a^2 &= \sigma(z^2) \tag{4}\\ z^3 &= W^3 a^2 + b^3 \tag{5}\\ a^3 &= \sigma(z^3) \tag{6}\\ \text{Cost} &= (y - a^3)^2 \tag{7} \end{align}$$

Where each level can be described by a *module*. The z layers are called linear layers, the a's as sigmoid layers, and the last line is simply the cost function. Notice that the activation functions are now their own layers, which actually simplifies things mathematically.

In our formulation, you can see that many of the layers consist of applying matrix products. When using minibatches, or doing convolution, which we will see shortly, we will often need to use tensor products. These types of mathematical operations have highly optimized libraries, often written to take special advantage of the particular architecture of a given CPU.

As you have already seen on problem set 5, training neural networks can be quite time consuming. Taking advantage of these benefits is quite important for even relatively modest training tasks.

BLAS: Basic Linear Algebra Subprograms

A specification giving consistent API for common operations between scalars, vectors, and matrices. For example, multiplying a matrix by a vector. Code written over the BLAS specification can be recompiled with a specific version of BLAS optimized for a given system. For example, R and NumPy can both be compiled with custom BLAS implementations.

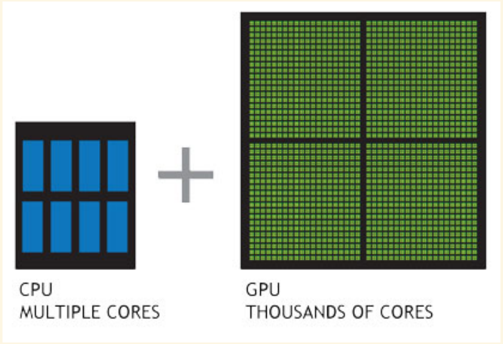
BLAS Implementations

Popular implementations include:

- Netlib BLAS
- OpenBLAS
- Automatically Tuned Linear Algebra Software (ATLAS)
- Intel's Math Kernel Library (MKL)

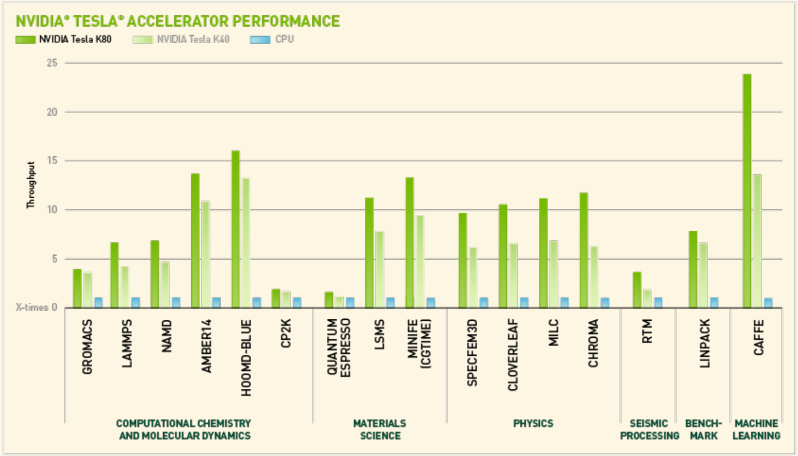
General Purpose GPU Programming

For years, specialized chips have been developed specifically for doing fast matrix operations. This research was driven by the need to offload compute heavy graphics functions from the main CPU onto a specialized GPU. These have recently been marketed for general purpose programming, with chips (termed GPGPUs) now being specifically built and designed for such tasks.



Example: NVIDIA Tesla

These are now virtually required for doing cutting-edge neural network research:

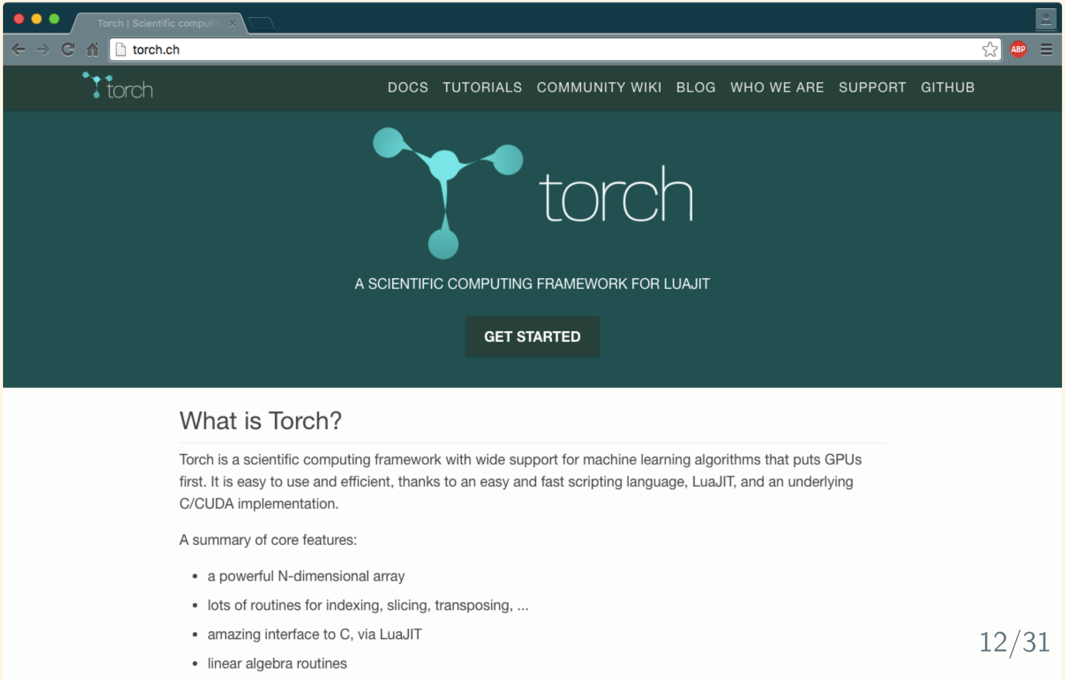


BLAS and GPGPUs

Libraries such as CULA, cuBLAS, cuSPARSE, and LibSciACC provide BLAS and other functionalities using GPU chips.

Neural network software / libraries

Back to neural networks, many libraries have been written for training classes of neural networks. Most of these try to support custom BLAS implementations, with the possibility of being compiled to a GPU. As understanding the landscape is important, I'll explain some of the currently popular examples and give my personal thoughts on them.



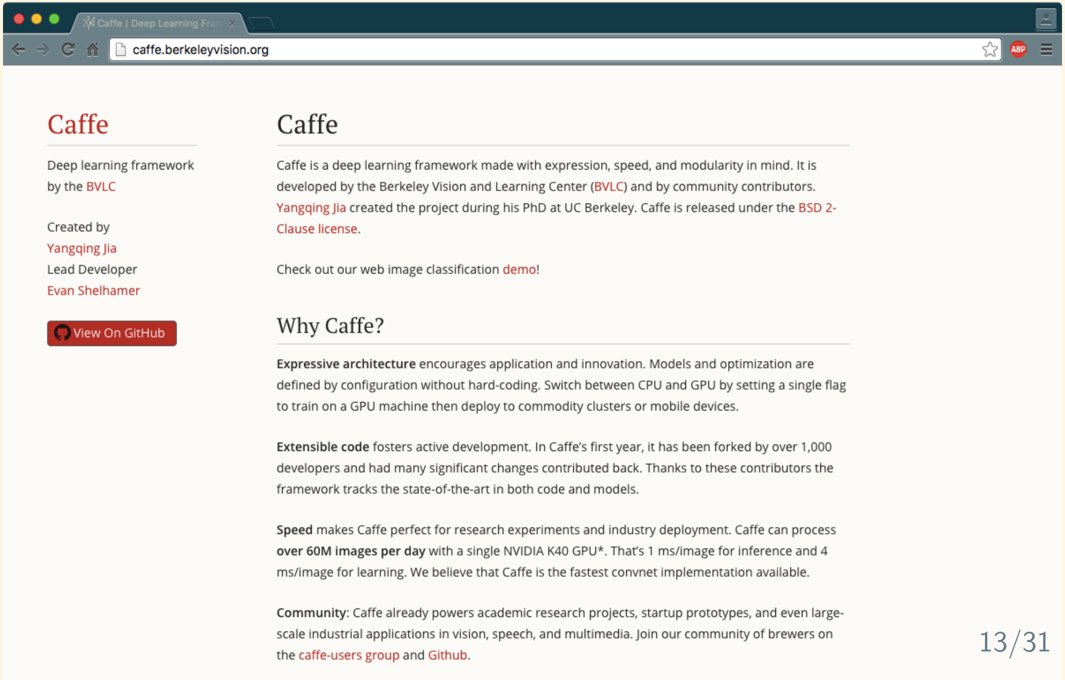
A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT

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



Caffe

Deep learning framework by the BVLC

Created by
Yangqing Jia
Lead Developer
Evan Shelhamer

View On GitHub

Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

Check out our web image classification demo!

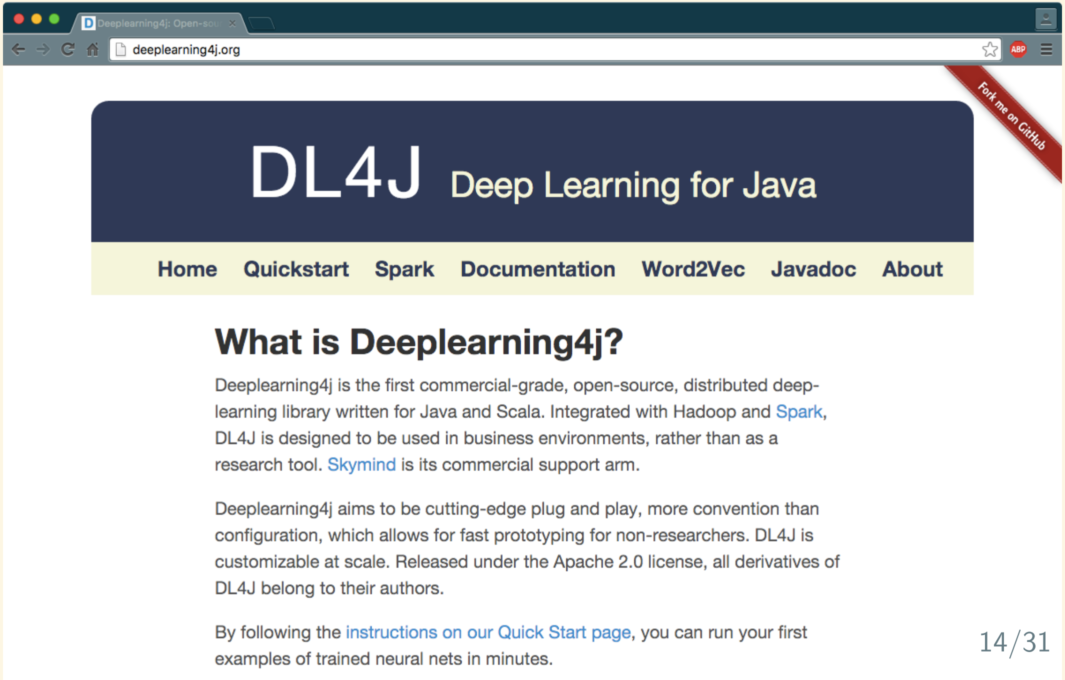
Why Caffe?

Expressive architecture encourages application and innovation. Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices.

Extensible code fosters active development. In Caffe's first year, it has been forked by over 1,000 developers and had many significant changes contributed back. Thanks to these contributors the framework tracks the state-of-the-art in both code and models.

Speed makes Caffe perfect for research experiments and industry deployment. Caffe can process **over 60M images per day** with a single NVIDIA K40 GPU*. That's 1 ms/image for inference and 4 ms/image for learning. We believe that Caffe is the fastest convnet implementation available.

Community: Caffe already powers academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia. Join our community of brewers on the caffe-users group and Github.



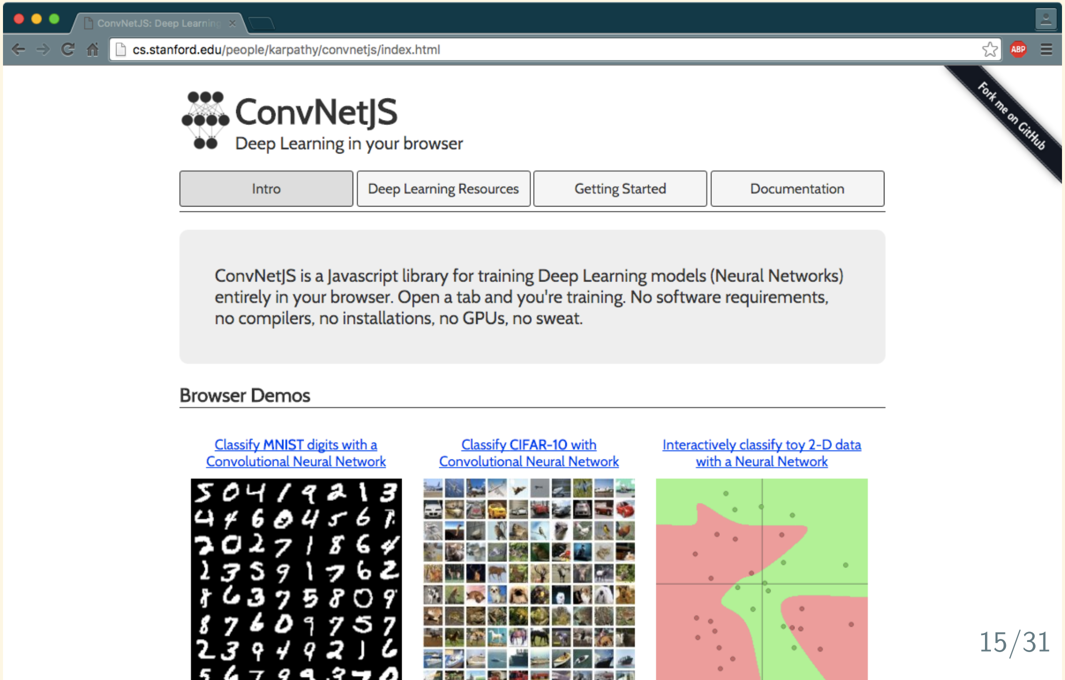
DL4J Deep Learning for Java

What is Deeplearning4j?

Deeplearning4j is the first commercial-grade, open-source, distributed deep-learning library written for Java and Scala. Integrated with Hadoop and Spark, DL4J is designed to be used in business environments, rather than as a research tool. Skymind is its commercial support arm.

Deeplearning4j aims to be cutting-edge plug and play, more convention than configuration, which allows for fast prototyping for non-researchers. DL4J is customizable at scale. Released under the Apache 2.0 license, all derivatives of DL4J belong to their authors.

By following the instructions on our Quick Start page, you can run your first examples of trained neural nets in minutes.



ConvNetJS: Deep Learning

cs.stanford.edu/people/karpathy/convnetjs/index.html

Fork me on GitHub

ConvNetJS

Deep Learning in your browser

Intro | Deep Learning Resources | Getting Started | Documentation

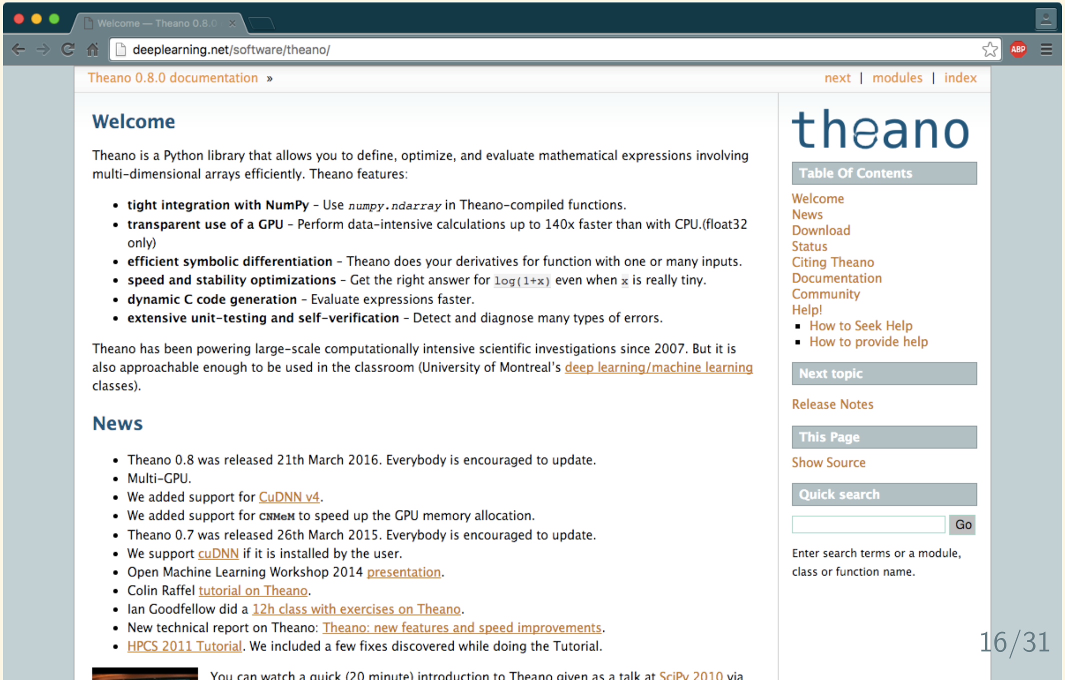
ConvNetJS is a Javascript library for training Deep Learning models (Neural Networks) entirely in your browser. Open a tab and you're training. No software requirements, no compilers, no installations, no GPUs, no sweat.

Browser Demos

Classify MNIST digits with a Convolutional Neural Network

Classify CIFAR-10 with Convolutional Neural Network

Interactively classify toy 2-D data with a Neural Network


deeplearning.net/software/theano/

Theano 0.8.0 documentation » next | modules | index

Welcome

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

- **tight integration with NumPy** – Use *numpy.ndarray* in Theano-compiled functions.
- **transparent use of a GPU** – Perform data-intensive calculations up to 140x faster than with CPU.(float32 only)
- **efficient symbolic differentiation** – Theano does your derivatives for function with one or many inputs.
- **speed and stability optimizations** – Get the right answer for `log(1+x)` even when `x` is really tiny.
- **dynamic C code generation** – Evaluate expressions faster.
- **extensive unit-testing and self-verification** – Detect and diagnose many types of errors.

Theano has been powering large-scale computationally intensive scientific investigations since 2007. But it is also approachable enough to be used in the classroom (University of Montreal's deep learning/machine learning classes).

News

- Theano 0.8 was released 21th March 2016. Everybody is encouraged to update.
- Multi-GPU.
- We added support for CuDNN v4.
- We added support for `CNMeM` to speed up the GPU memory allocation.
- Theano 0.7 was released 26th March 2015. Everybody is encouraged to update.
- We support cuDNN if it is installed by the user.
- Open Machine Learning Workshop 2014 presentation.
- Colin Raffel tutorial on Theano.
- Ian Goodfellow did a 12h class with exercises on Theano.
- New technical report on Theano: Theano: new features and speed improvements.
- HPCS 2011 Tutorial. We included a few fixes discovered while doing the Tutorial.

You can watch a quick (20 minute) introduction to Theano given as a talk at SciPy 2010 via

theano

Table Of Contents

Welcome
News
Download
Status
Citing Theano
Documentation
Community
Help!
- How to Seek Help
- How to provide help

Next topic

Release Notes

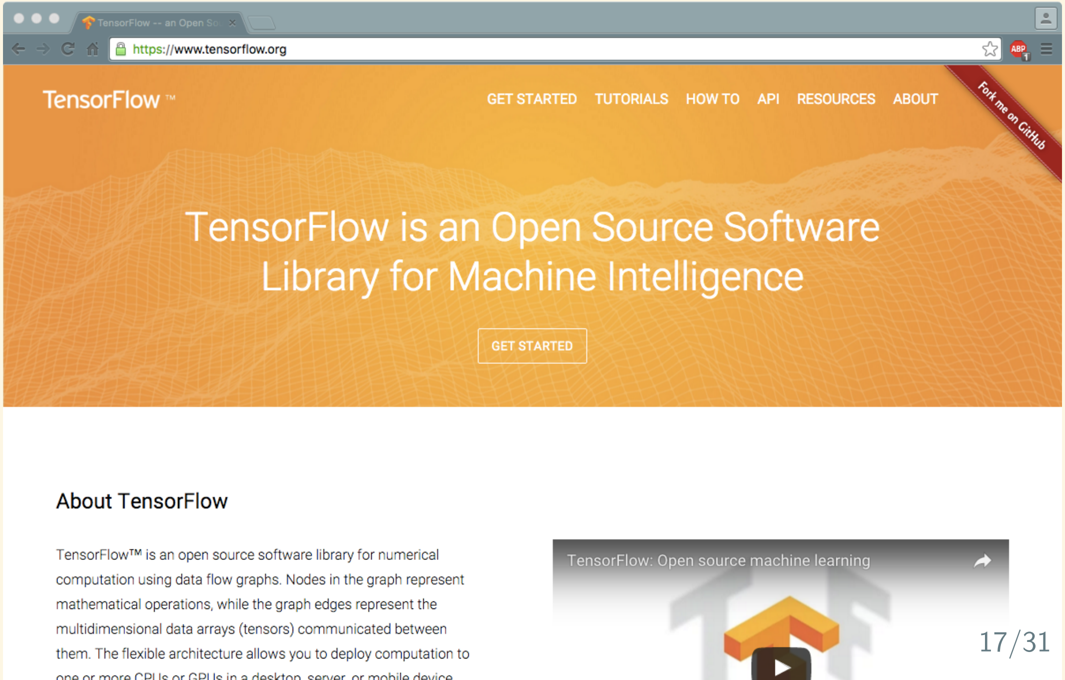
This Page

Show Source

Quick search

Go

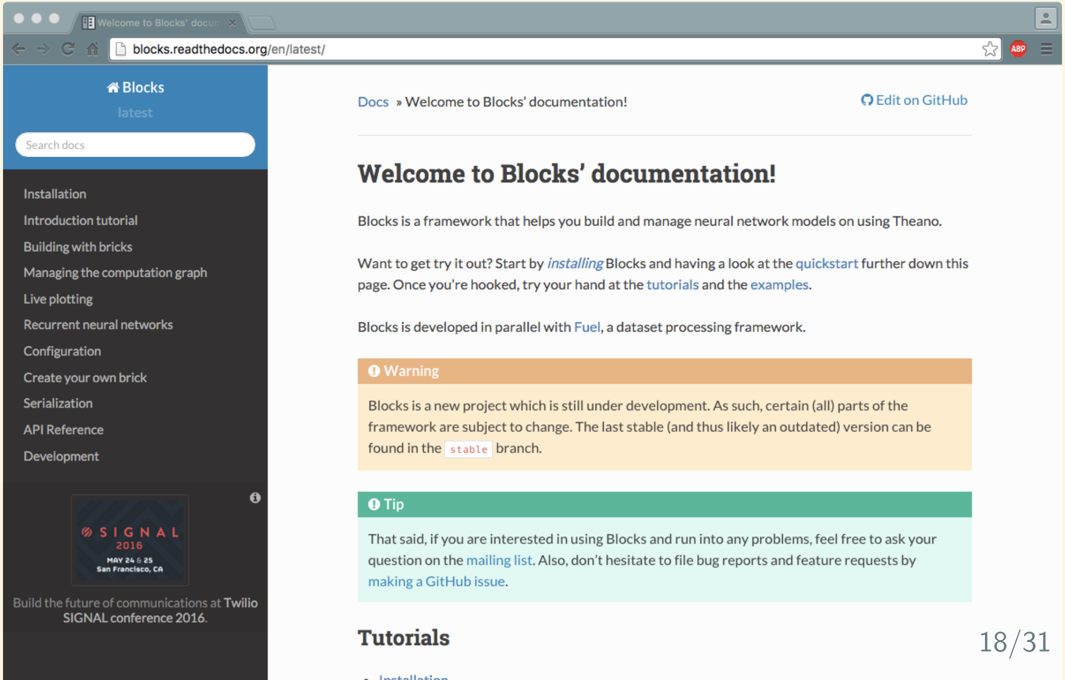
Enter search terms or a module, class or function name.



TensorFlow is an Open Source Software Library for Machine Intelligence

About TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device



Welcome to Blocks' docum

blocks.readthedocs.org/en/latest/

Blocks

latest

Search docs

- Installation
- Introduction tutorial
- Building with bricks
- Managing the computation graph
- Live plotting
- Recurrent neural networks
- Configuration
- Create your own brick
- Serialization
- API Reference
- Development

Build the future of communications at Twilio SIGNAL conference 2016.

Docs » Welcome to Blocks' documentation!

Edit on GitHub

Welcome to Blocks' documentation!

Blocks is a framework that helps you build and manage neural network models on using Theano.

Want to get try it out? Start by *installing* Blocks and having a look at the quickstart further down this page. Once you're hooked, try your hand at the tutorials and the examples.

Blocks is developed in parallel with Fuel, a dataset processing framework.

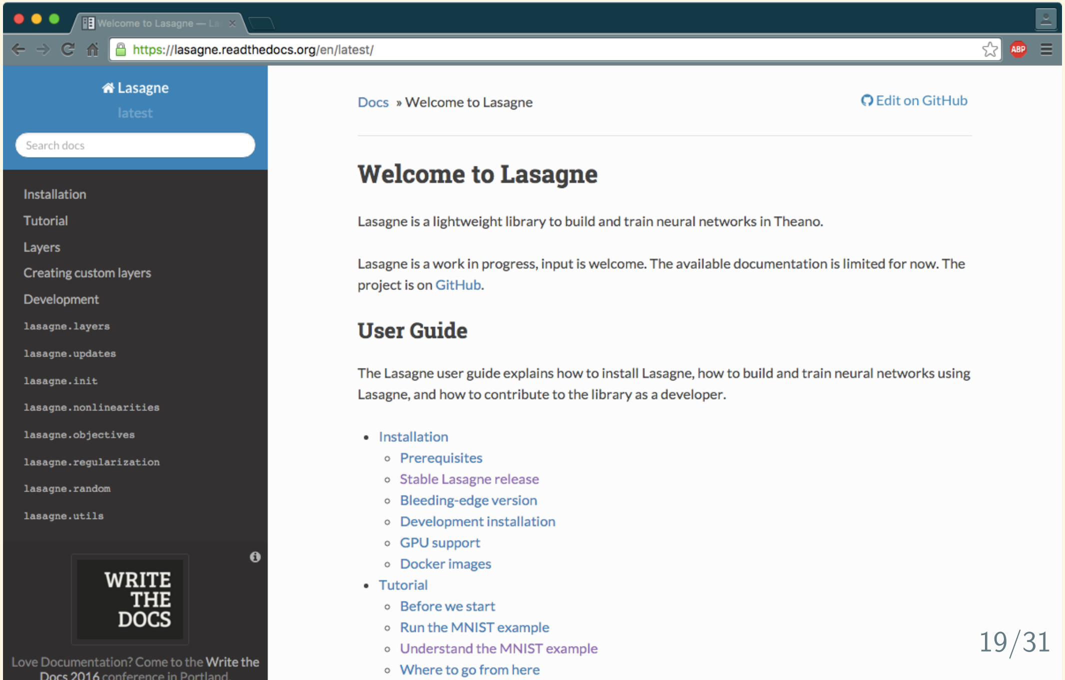
Warning

Blocks is a new project which is still under development. As such, certain (all) parts of the framework are subject to change. The last stable (and thus likely an outdated) version can be found in the `stable` branch.

Tip

That said, if you are interested in using Blocks and run into any problems, feel free to ask your question on the mailing list. Also, don't hesitate to file bug reports and feature requests by making a GitHub issue.

Tutorials



Love Documentation? Come to the Write the Docs 2016 conference in Portland.

Welcome to Lasagne

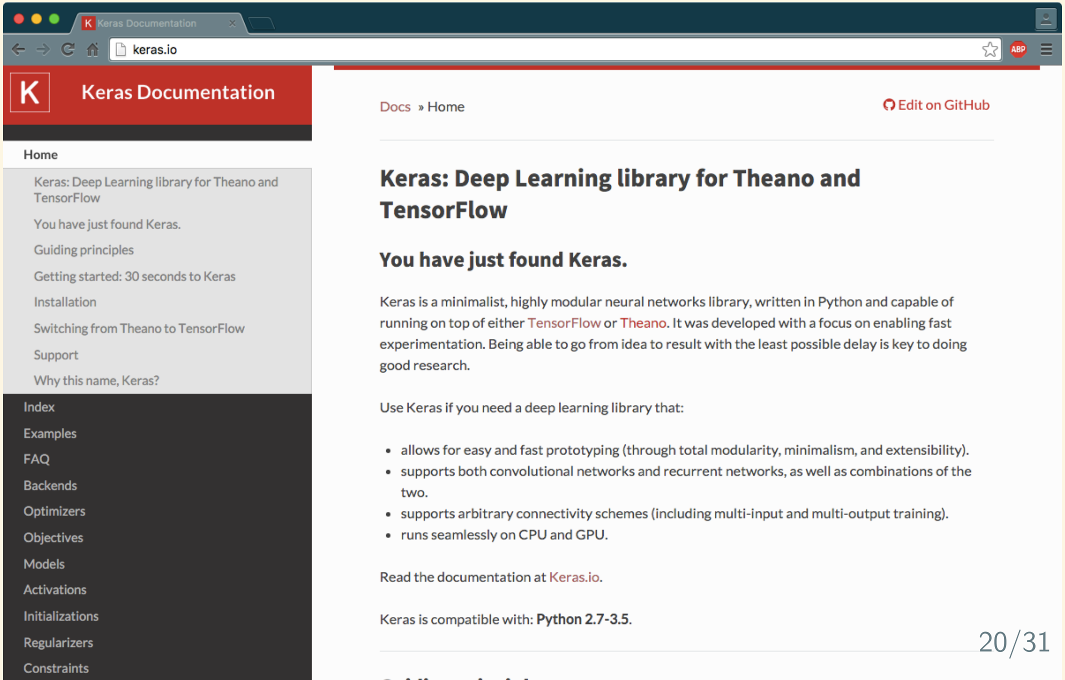
Lasagne is a lightweight library to build and train neural networks in Theano.

Lasagne is a work in progress, input is welcome. The available documentation is limited for now. The project is on GitHub.

User Guide

The Lasagne user guide explains how to install Lasagne, how to build and train neural networks using Lasagne, and how to contribute to the library as a developer.

- Installation
  - Prerequisites
  - Stable Lasagne release
  - Bleeding-edge version
  - Development installation
  - GPU support
  - Docker images
- Tutorial
  - Before we start
  - Run the MNIST example
  - Understand the MNIST example
  - Where to go from here



Home
Keras: Deep Learning library for Theano and TensorFlow
You have just found Keras.
Guiding principles
Getting started: 30 seconds to Keras
Installation
Switching from Theano to TensorFlow
Support
Why this name, Keras?
Index
Examples
FAQ
Backends
Optimizers
Objectives
Models
Activations
Initializations
Regularizers
Constraints

Keras: Deep Learning library for Theano and TensorFlow

You have just found Keras.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Keras is compatible with: **Python 2.7-3.5**.

keras

We are going to use the keras library in this course; it uses python (a language many of you are familiar with and is relatively close to R), is easy to install across different operating systems, and supports both convolution and recurrent neural networks. As it uses either the Theano or TensorFlow backends, it is likely to be relatively up-to-date for at least the immediate future.

Importing functions

I will let you all consult the keras website for getting the library installed on your machine. Before diving into the tutorial, I wanted to make a note of a difference between python and R.

In R, when we call `library` or `require` with a package name, all of the public functions in that package are added directly to the search path. For example:

```
> library(class)
> knn(train, test, cl, k = 3, prob=TRUE)
```

We could instead explicatively call the knn function, without ever loading the class library, by doing the following:

```
> class::knn(train, test, cl, k = 3, prob=TRUE)
```

This would work (to the user) exactly the same as the other form.

In python, this process works differently. We cannot call a function without first importing the module it is contained in. Even once a function is imported, we still need to explicitly let python know from which module the function is coming from:

```
>>> import numpy
>>> numpy.array((1,2))
array([1, 2])
```

In python, this process works differently. We cannot call a function without first importing the module it is contained in. Even once a function is imported, we still need to explicitly let python know from which module the function is coming from:

```
>>> import numpy
>>> numpy.array((1,2))
array([1, 2])
```

You can imagine that this can become quite verbose, and it is possible the give a shortened name to the module instead:

```
>>> import numpy as np
>>> np.array((1,2))
array([1, 2])
```

With some abbreviations, such as this one, becoming fairly canonical.

It is possible to avoid having to type even this shortened form, by explicitly importing a given function:

```
>>> from numpy import array
>>> array((1,2))
array([1, 2])
```

It is possible to avoid having to type even this shortened form, by explicitly importing a given function:

```
>>> from numpy import array
>>> array((1,2))
array([1, 2])
```

Multiple functions can be imported all at once:

```
>>> from numpy import array, zeros
>>> zeros((3,4))
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

It is possible, but considered very poor form, to load all of the functions from a module by doing:

```
>>> from numpy import *
>>> array((1,2))
array([1, 2])
```

Which gives a more R-like behavior.

Finally, some modules contain additional submodules. For example, SciPy has a stats submodule:

```
>>> from scipy.stats import beta
>>> a, b = 1., 2.
>>> beta.rvs(a, b, size=4)
array([ 0.04541352,  0.82622532,  0.10097377,  0.37001809])
```

You can generally think of these as completely independent modules.

Minimal working example

Speaking of importing functions from python modules, let's load these four functions from keras:

```
>>> from keras.models import Sequential
>>> from keras.layers.core import Dense, Activation
>>> from keras.optimizers import SGD
```

Minimal working example, cont.

We start by constructing a Sequential model, and adding three layers to it: a linear, or dense, layer; the activation layer; and the output layer.

```
>>> model = Sequential()
>>> model.add(Dense(512, input_shape=(28 * 28,)))
>>> model.add(Activation("sigmoid"))
>>> model.add(Dense(10))
```

The input shape parameter needs to be set for the first layer, but can be inferred for all of the other layers.

Minimal working example, cont.

Now, we will compile the model (this may take a minute or more) using stochastic gradient descent.

```
>>> sgd = SGD(lr = 0.02, momentum = 0.01, nesterov = True)
>>> model.compile(loss='mse', optimizer=sgd)
```

By setting various environment variables, we could do this over Tensorflow or Theano, and could make it run over various CPU or GPU architectures.

Minimal working example, cont.

To train the actual model, we run the fit method on the model, giving the desired batch size and number of epochs:

```
>>> model.fit(X_train, Y_train, batch_size=32, nb_epoch=25,
...           verbose=1, show_accuracy=True)
```

This will produce a running output of the model training process.

Minimal working example, cont.

To train the actual model, we run the fit method on the model, giving the desired batch size and number of epochs:

```
>>> model.fit(X_train, Y_train, batch_size=32, nb_epoch=25,
...           verbose=1, show_accuracy=True)
```

This will produce a running output of the model training process.

Let's now actually run this is the Python interpreter and see the various advanced features than can be easily added to adapt the final model.