# Software Packages for Deep Learning

Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

# Outline

# Machine Learning

- Unlike traditional numerical simulation, "ML gives computers the ability to learn without being explicitly programmed" [Samuel 1959]
- As a research field, ML explores the study and construction of algorithms that can learn from and make predictions on data
- Related fields: data mining, computational statistics, optimization, ...
- Fourth paradigm, big data, artificial intelligence, Internet of things, deep learning, ...

# General Tasks of ML

- Classification: Inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes
- Clustering: Inputs are divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task
- Regression: Similar to classification, but the outputs are continuous rather than discrete
- Density estimation
- Dimensionality reduction
- ...

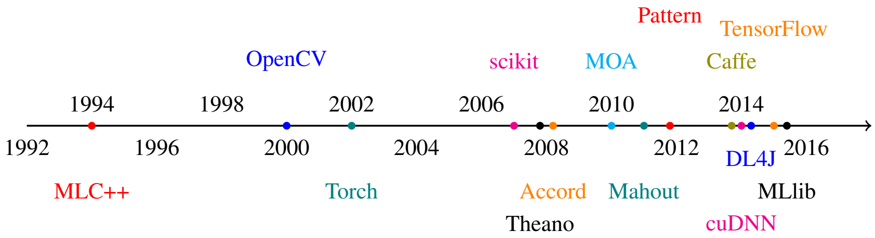
# Packages for General Machine Learning

## What is the purpose?

- Solving problems from practical applications (user interface)
- Developing algorithms and optimizing implementation (development)
- Theoretical analysis for machine learning

## What do we want for a ML package?

- Easy for new tasks and new network structures (less steep learning curve)
- Easy for debugging (with good support and large community)
- Performance and scalability



# Deep Learning: Pros and Cons

Deep Learning has been introduced with the objective of moving ML closer to one of its original goals—AI. The main motivations includes:

- Insufficient depth can hurt
- The brain has a deep architecture
- Cognitive processes seem deep

Pros:

- conceptually simple
- nonlinear
- highly flexible and configurable
- learned features can be extracted
- can be fine-tuned with more data
- efficient for multi-class problems
- world-class at pattern recognition

Cons:

- hard to interpret
- theory not well understood
- slow to train and score
- overfits, needs regularization
- more parameters
- inefficient for categorical variables
- data hungry, learns slowly

https://www.slideshare.net/0xdata/transform-your-business-with-ai-deep-learning-and-machine-learning

# Comparison: Basic Information

| Viewpoint | Torch | Caffe | TensorFlow | MXNet |
|---|---|---|---|---|
| Started | 2002 | 2013 | 2015 | 2015 |
| Main Developers | Facebook, Twitter, Google, ... | BVLC (Berkeley) | Google | DMLC |
| License | BSD | BSD | Apache | Apache |
| Core Languages | C/Lua | C++ | C++ Python | C++ |
| Supported Interface | Lua | C++/Python Matlab | C++/Python R/Java/Go | C++/Python R/Julia/Scala |

- BVLC, Berkeley Vision and Learning Center
- DMLC, Distributed (Deep) Machine Learning Community, supported by Amazon, Intel, Microsoft, nVidia, Baidu, ...

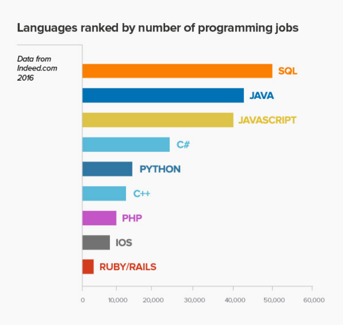
# Comparison: Performance

| Viewpoint | Torch | Caffe | TensorFlow | MXNet |
|---|---|---|---|---|
| Pretrained Models | Yes | Yes | No | Yes |
| High-level Support | Good | Good | Good | Good |
| Low-level Operators | Good | Good | Fairly good | Very few |
| Speed One-GPU | Great | Great | Not so good | Excellent |
| Memory Management | Great | Great | Not so good | Excellent |
| Parallel Support | Multi-GPU | Multi-GPU | Multi-GPU | Distributed |

http://blog.revolutionanalytics.com/2016/08/deep-learning-part-1.html

# Python: A general-purpose programming language

- Created by Guido van Rossum in 1989 and first released in 1991
- Named after “the Monty Python” (British comedy group)
- An interpreted language—simple, clear, and readable
- Python has many excellent packages for machine learning
- The language of choice in introductory programming courses




| Feb 2017 | Change | Programming language | Share | Trends |
|---|---|---|---|---|
| 1 | | Java | 22.6 % | -1.3 % |
| 2 | | Python | 14.7 % | +2.8 % |
| 3 | | PHP | 9.4 % | -1.2 % |
| 4 | | C# | 8.3 % | -0.3 % |
| 5 | ↑↑ | Javascript | 7.7 % | +0.4 % |
| 6 | | C | 7.0 % | -0.2 % |
| 7 | ↓↓ | C++ | 6.9 % | -0.6 % |
| 8 | | Objective-C | 4.2 % | -0.6 % |
| 9 | ↑ | R | 3.4 % | +0.4 % |
| 10 | ↓ | Swift | 2.9 % | +0.1 % |

# Python for Scientific Computing

NCMIS

## Why Python for scientific computing?

- Dynamic data types and automatic memory management
- Full modularity, supporting hierarchical packages
- Strong introspection capabilities[1]
- Exception-based error handling

## Why consider such a slow language for simulation?

- Good for proof-of-concept prototyping
- Implementation time versus execution time
- Code readability and maintenance — short code, fewer bugs
- Well-written Python code is “fast enough” for most computational tasks
- Time critical parts executed through compiled language or available packages

[1] Code introspection is the ability to examine classes, functions and keywords to know what they are, what they do and what they know. Python provides several functions and utilities for code introspection, like dir(), help(), type().

# Functions and Modules

## Defining functions

```python
def square(x):
    return x*x
```

## Using modules

1. import math: This will only introduce the name math into the name space in which the import command was issued. The names within the math module will not appear in the enclosing namespace: they must be accessed through the name math. For example: math.sin(3.14).
2. from math import *: This does not introduce the name math into the current namespace. It does however introduce all public names of the math module into the current namespace, directly using: sin(3.14)
3. from math import sin: This will only import the sin function from math module and introduce the name sin into the current namespace, but it will not introduce the name math into the current namespace, directly using: sin(3.14)

# Built-in Data Structures

## Numeric types: int, float, complex

```
b=1L     # long int
c=0xf    # int (hex format)
d=010    # int (octal format)
e=1.0    # float
f=1+2j   # complex
```

## Sequence types: list, tuple, str, dict

```
t=(3.14, True, 'Yes', [1], ())                      # tuple example
l=[3.14, True, 'Yes', [1], (1L, 0xf)] + [None]*3    # list example
s='Hello' + ", " + 'world!'                         # str example 1
s=("Hello, " "world!")                              # str example 2
d={1: 'int', 'pi': 3.14}                            # dict example
s="Python"; s.find('thon')                          # find substring
```

## Formatted output

```
print('%(lang)s has %(num)02d quote types.'
...     %{'lang':"Python", "num":3})
```

# Control Flow

## If-then-else

```
a = 1
if a > 0:
    print "a is positive"
elif a=0:
    print "a is zero"
else:
    print "a is negative"
```

## For loop

```
for i in range(10):
    print i
```

## While loop

```
sum = 0; i = 0
while i < 10:
    sum += i
    i += 1
```

http://www.pythontutor.com

# Programming interface

1. Wide range applications
   - Speech, image and video applications
   - Large-scale machine-learning applications
2. Fastest scripting language Lua is used
3. Easily ported to any platform
   - Torch can run on iPhone with no modification to scripts
4. Easy extensibility
   - Easy to integrate any library into Torch

# Example 1:Linear-Regression

```
require 'torch'
require 'optim'
require 'nn'

# write the loss to a text file and read from there
#  to plot the loss as training proceeds
logger = optim.Logger('loss_log.txt')

# input data
data = torch.Tensor{{40,  6,  4},{44, 10,  4},{46, 12,  5},
{48, 14,  7},{52, 16,  9},{58, 18, 12},{60, 22, 14},
{68, 24, 20},{74, 26, 21},{80, 32, 24}}

# define the container
model = nn.Sequential()
ninputs = 2; noutputs = 1

# define the only module
model:add(nn.Linear(ninputs, noutputs))

# Define a loss function
criterion = nn.MSECriterion()
```

# Example 1:Linear-Regression

```
# retrieve its trainable parameters
x, dl_dx = model:getParameters()

# compute loss function and its gradient
feval = function(x_new)
   # set x to x_new, if differnt
   if x ~= x_new then
      x:copy(x_new)
   end

# select a new training sample
   _nidx_ = (_nidx_ or 0) + 1
   if _nidx_ > (#data)[1] then _nidx_ = 1 end

   local sample = data[_nidx_]
   local target = sample[{ {1} }]
   local inputs = sample[{ {2,3} }]

# reset gradients
   dl_dx:zero()
```

# Example 1:Linear-Regression

```
# evaluate the loss function and its derivative wrt x
   local loss_x = criterion:forward(model:forward(inputs), target)
   model:backward(inputs, criterion:backward(model.output, target))

# return loss(x) and dloss/dx
   return loss_x, dl_dx
end

# define SGD
sgd_params = {
   learningRate = 1e-3,
   learningRateDecay = 1e-4,
   weightDecay = 0,
   momentum = 0
}

# we cycle 1e4 times over our training data
for i = 1,1e4 do
   #this variable is used to estimate the average loss
   current_loss = 0
   #an epoch is a full loop over our training data
   for i = 1,(#data)[1] do
      # return new x and value of the loss functions
      _,fs = optim.sgd(feval,x,sgd_params)
```

# Example 1:Linear-Regression

```
        # update loss
        current_loss = current_loss + fs[1]
    end

    # report average error on epoch
    current_loss = current_loss / (#data)[1]
    print('current loss = ' .. current_loss)

    logger:add{['training error'] = current_loss}
    logger:style{['training error'] = '-'}
    logger:plot()
end

# Test the trained model
text = {40.32, 42.92, 45.33, 48.85, 52.37, 57, 61.82, 69.78,
        72.19, 79.42}

for i = 1,(#data)[1] do
    local myPrediction = model:forward(data[i][{{2,3}}])
    print(string.format("%2d  %6.2f %6.2f", i, myPrediction[1], text[i]
end
```

# Example 2:Two-layler network

```
import torch
from torch.autograd import Variable

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold inputs and outputs, and wrap them in V
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

# Use the nn package to define our model as a sequence of layers.
model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out),
        )

# The nn package also contains definitions of popular loss functions;
loss_fn = torch.nn.MSELoss(size_average=False)
```

# Example 2:Two-layler network

```
learning_rate = 1e-4
for t in range(500):
    # Forward pass: compute predicted y by passing x to the model.
    y_pred = model(x)

        # Compute and print loss.
        loss = loss_fn(y_pred, y)
        print(t, loss.data[0])

        # Zero the gradients before running the backward pass
         model.zero_grad()

         # Backward pass: compute gradient of the loss
         loss.backward()

         # Update the weights using gradient descent
         for param in model.parameters():
                 param.data -= learning_rate * param.grad.data
```

https://github.com/jcjohnson/pytorch-examples

# Programming interface

1. Expressive architecture
    - Define models and optimization by configuration without hard-coding
    - With protocol tool to define parameters for nets and solvers . . .
2. Support GPUs
3. Mainly focus CNN for images
4. Not well documented

# Example 1: Image Classification

```
import caffe
import matplotlib.pyplot as plt

# paste your image URL here
my_image_url = "https://wikipedia/Orang_Utan/2C_Malaysia.JPG"
!wget -O image.jpg $my_image_url

# transform it and copy it into the net
image = caffe.io.load_image('image.jpg')
caffe.net.blobs['data'].data[...]=transformer.preprocess('data',image)

# perform classification
caffe.net.forward()

# obtain the output probabilities
output_prob = net.blobs['prob'].data[0]

# sort top five predictions from softmax output
top_inds = output_prob.argsort()[::-1][:5]

plt.imshow(image)
print 'probabilities and labels:'
zip(output_prob[top_inds], labels[top_inds])
```

# Example 2: Extend layer in Caffe

```
import caffe
import numpy as np

class EuclideanLoss(caffe.layer):
    def setup(self, bottom, top):
        #check input pair
        if len(bottom) != 2:
            raise Exception(”Need two inputs to compute distance”)

    def reshape(self, bottom, top):
        #check input dimensions match
        if bottom[0].count != bottom[1].count:
            raise Exception(”Inputs must have the same dimension”)
        #difference in shape of inputs
        self.diff = np.zeros_like(bottom[0].data, dtype=np.float32)
        # loss output is scalar
        top[0].reshape(1)

    def forward(self, bottom, top):
        self.diff[...] = bottom[0].data − bottom[1].data
        top[0].data[...] = np.sum(self.diff**2)/bottom[0].num/2.
```

# Example 2: Extend layer in Caffe

```
    def backward(self, top, propagate_down, bottom):
        for i in range(2):
            if not propagate_down[i]:
                continue
            if i == 0:
                sign = 1
            else:
                sign = -1
            bottom[i].diff[...] = sign.self.diff / bottom[1].num
```

Define a class in Python to extend Layer

```
layer{
type: "Python"
python_param {
    module: "laylers"
    layer: "EuclideanLoss"
}
}
```

https://docs.google.com/presentation/d/1UeKXVgRvvxg9OUdh_UiC5G71UMscNPlvArsWER41PsU/edit#slide=id.gc2fcdcce7_216_0

# Computational graph

TensorFlow computations are expressed as stateful dataflow graphs.

- each node corresponds to an operation (eg tensor, add, sub etc)
- each edge corresponds to tensor flowing direction

```
import tensorflow as tf

graph = tf.Graph()

with graph.as_default():
    with tf.name_scope('input_var'):
        a = tf.Variable(tf.random_uniform
                        ([1], -1.0, 1.0))
        tf.summary.histogram('a', a)
        b = tf.Variable(tf.random_uniform
                       ([1], -1.0, 1.0))
        tf.summary.histogram('b', b)
    with tf.name_scope('output_var'):
        c = tf.multiply(a, b)
        tf.summary.histogram('c', c)

    merged = tf.summary.merge_all()
    writer = tf.summary.FileWriter
              ('/home/fan/board', graph)
```

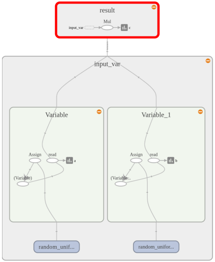



Figure: Computaion graph

# Programming interface

In this section we provide a discussion of the computational graph architecture underlying the TensorFlow software library.

- Graph: In TensorFlow, machine learning algorithms are represented as computational graph. A computational or dataflow graph is a form of directed graph where vertices or nodes describe operations, while edges represent data flowing between these operations.
- Operation: An opreation may represent a mathematical equation, a variable or constant, a control flow directive, a file I/O operation or even a network communication port.
- Tensor: A tensor is a multi-dimensional collection of homogeneous values with a fixed, static type.
- Variable: Variables can be described as persistent, mutable handles to in-memory buffers storing tensors.
- Session: In TensorFlow the execution of operations and evaluation of tensors may only be preformed in a special environment called session.

# Visualization: TensorBoard

Computation graphs are powerful but complicated

- thousands of nodes or more
- network is deep
- graph visualization tool TensorBoard is helpful

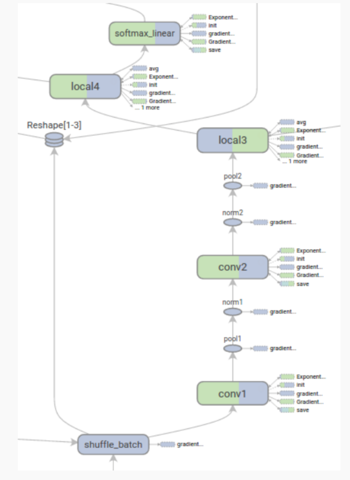



Figure: Graph Visualization

# Example 1: SoftMax

```
import tensorflow as tf

# Import the training data (MNIST)
import tf.examples.tutorials.mnist.input_data as input_data

# Possibly download and extract the MNIST data set
# Retrieve the labels as one-hot-encoded vectors
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# Create a new graph
graph = tf.Graph()

# Set our graph as the one to add nodes to
with graph.as_default():
    # Placeholder for input variables (None = variable dimension)
    x = tf.placeholder("float", shape=[None, 784])
    # Placeholder for labels
    y_ = tf.placeholder("float", shape=[None, 10])

    # Weights and bias
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
```

# Example 1:SoftMax

```
    # Apply softmax regression model
    y = tf.nn.softmax(tf.matmul(x, W) + b)

    # Compute the cross entropy of y_ and y
    entropy = -tf.reduce_sum(y_*tf.log(y))
    # Create a gradient-descent optimizer
    train_step =
        tf.train.GradientDescentOptimizer(0.01).minimize(entropy)

    # Find the indices where the predictions were correct
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

with tf.Session(graph=graph) as session:
    # Initialize all variables
    tf.global_variables_initializer().run()

    # Train the model
    for step in range(1000):
        batch_x, batch_y = mnist.train.next_batch(100)
        train_step.run(feed_dict={x: batch_x, y_: batch_y})
    # Print the accuracy using the model
    print accuracy.run(feed_dict={x: mnist.test.images,
                                  y_:mnist.test.labels})
```

# Programming interface

1. Mxnet.ndarray
   - Similar to numpy.ndarray
   - Supports both CPU and GPU
2. Support building neural network graphs
   - Call mx.viz.plot_network( )
3. Mixed programing
   - Suport both imperative and declarative programming
4. Provide intermediate-level and high-level interface modules
5. Provide data parallelism with multi-devices
6. Provide abundant IO functions
7. Support many scope applications(e.g. computer vision, natural language processing, speech recoginition , unsupervised machine learning, support embedded APIs, visualization)

# Example 1: SoftMax

```
import mxnet
import mxnet.symbol as sym
import numpy as np
import numpy.random as random
import time
from minpy.core import function
from minpy.core import grad_and_loss

# define softmax symbol
x_shape = (num_samples, num_classes)
label_shape = (num_samplesm,)
softmax_symbol = sym.SoftmaxOutput(data=sym.Variable('x'),
                    name='softmax', grad_scale=1.0/num_samples)

# convert MXNet symbol into a callable function
# corresponding gradient function
softmax = function(softmax_symbol, [('x', x_shape),
                  ('softmax_label', label_shape)])

# make softmax_label;
# MXNet's softmax operator does not use one-of-many label format
softmax_label = np.argmax(label, axis=1)
```

# Example 1: SoftMax

```python
# Redefine loss function using softmax as one operator
def train_loss(w, x):
    y = np.dot(x, w)
    prob = softmax(x=y, softmax_label=softmax_label)
    loss = -np.sum(label * np.log(prob)) / num_samples
    return loss

# Initialize weight matrix (again)
weight = random.randn(num_features, num_classes)

# Calculate gradient function automatically
grad_function = grad_and_loss(train_loss)

# Now training it for 100 iterations
start_time = time.time()
for i in range(100):
    dw, loss = grad_function(weight, data)
    if i % 10 == 0:
        print 'Iter {}, training loss {}'.format(i, loss)
    weight -= 0.1 * dw
print 'Training time: {}s'.format(time.time() - start_time)
```

# Numerical tests


Thank You!