# A Quick Survey on Deep Learning Engines

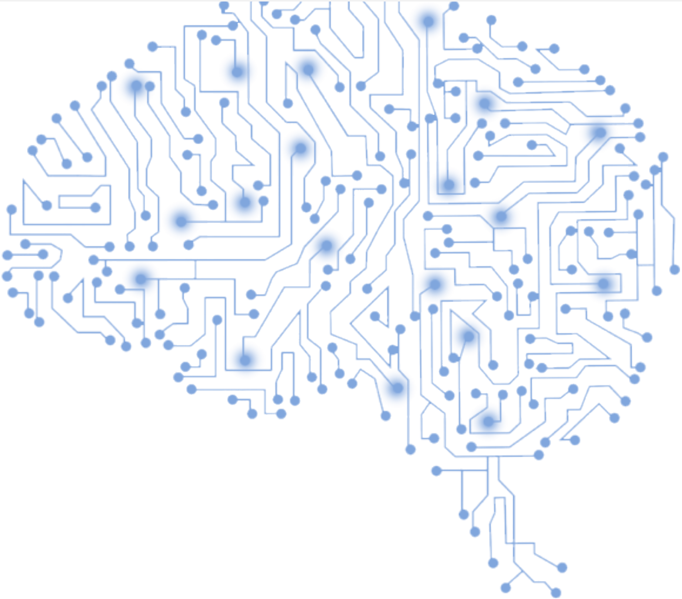
Chensong Zhang

with Zheng Li and Ronghong Fan

DL Seminar — April 27, 2017

# Outline

- Introduction
- Comparison
- Python
- TensorFlow
- MXNet
- Torch
- Caffe



# §1. Introduction

1 Machine Learning

2 ML Software Packages

3 Deep Learning

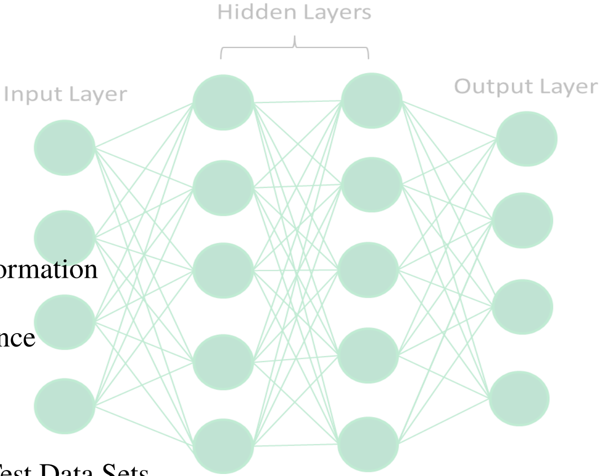
4 Goal of This Study

5 DL Engines: Basic Information

6 DL Engines: Performance

7 Hardware Platforms

8 Neural Networks and Test Data Sets



# Machine Learning

## What is ML?

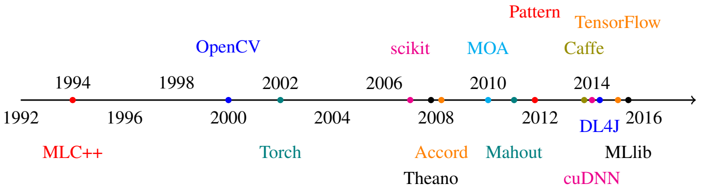
- Unlike traditional numerical simulation, “ML gives computers the ability to learn without being explicitly programmed” [Samuel 1959]
- As a research field, ML explores the study and construction of algorithms that can learn from and make predictions on data
- Fourth paradigm, big data, Internet of things, artificial intelligence, ...

## 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 of these classes
- ☞ Regression: Similar to classification, but the outputs are continuous
- ✎ Clustering: Inputs are divided into several groups (Unlike in classification, the groups are not known beforehand, making this an unsupervised task)
- ✎ Density estimation, dimension reduction, ...

Deep Learning, by I. Goodfellow, Y. Bengio, and A. Courville, Chinese version available online
https://github.com/exacity/deeplearningbook-chinese

# ML Software Packages




Tour of TensorFlow, by Peter Goldsborough, arXiv, 2016

## Other worth-noting ML (DL) packages:

- PyTorch (still in beta stage)
- Theano (Lasagne/Keras): Slow in graph compilation
- ☞ MXNet (DMLC): First released on Jan 2015, scalable distributed computing
- ☞ CNTK/DMTK (Microsoft): First released on April 2015
  - Windows/Linux, no official OS X support thought
  - C++/Python front-end
- Neon (Nervana & Intel): First released on May 2015, professional support
- Caffe2 (Google, Facebook, ...): Release on late 2016
- Digits (Nvidia, powered by Caffe): Web interface

# Deep Learning

Deep Learning has been introduced with the objective of moving ML closer to one of its original goals—Artificial Intelligence. Main motivations:

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

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
- good at pattern recognition

Cons:

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

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

# Goal of This Study

## What is our purpose?

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

## How to use a ML package?

- Train models with large data set on high-performance computing platforms
- Deploy applications on various (computing) platforms, like smart phones

## What we want for a ML package?

- Easy for debugging (with good documentation, support, and active community)
- Good flexibility
  - Easy to add new tasks
  - Easy to build new network structures
- Good performance and scalability
  - Multicore CPU
  - Single or multiple GPU(s)
  - Cluster

# DL Engines: Basic Information

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

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

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

# DL Engines: Performance

| Viewpoint | Torch7 | Caffe | TensorFlow | MXNet |
|---|---|---|---|---|
| Pretrained Models | Yes | Yes | No | Yes |
| High-level Support | Good | Good | Good | Good |
| Low-level Operators | Good | Good | Fairly good | Increasing fast |
| Speed One-GPU | Great | Great | Good | Good |
| Memory Management | Great | Great | Not so good | Excellent |
| Parallel Support | Multi-GPU | Multi-GPU | Multi-GPU | Distributed |
| Coding Style | Imperative | Declarative | Declarative | Mixed |
| GitHub Watching | 649/268 | 1856 | 4939 | 887 |

# Hardware Platforms

Four test platforms were used in the following tests

- CPU: one quad-core desktop CPU (Intel i7-3820 CPU @3.60GHz) and two 8-core server-grade CPUs (Intel Xeon CPU E5-2630 v3 @2.40GHz)
- GPU: GTX 1080 @1607MHz with Pascal architecture, and Telsa K80 @562MHz with Kepler architecture

| Computational Unit | Cores | Memory | OS | CUDA |
|---|---|---|---|---|
| Intel CPU i7-3820 | 4 | 64 GB | Ubuntu 14.04 | – |
| Intel CPU E5-2630x2 | 16 | 128 GB | CentOS 7.2 | – |
| GTX 1080 | 2560 | 8 GB | Ubuntu 14.04 | 8 |
| Telsa K80 GK210 | 2496 | 12 GB | CentOS 7.2 | 8 |

Figure: The experimental hardware settings for numerical tests

Benchmarking State-of-the-Art Deep Learning Software Tools, by S.-H. Shi, et al., arXiv, 2017

# Neural Networks and Test Data Sets

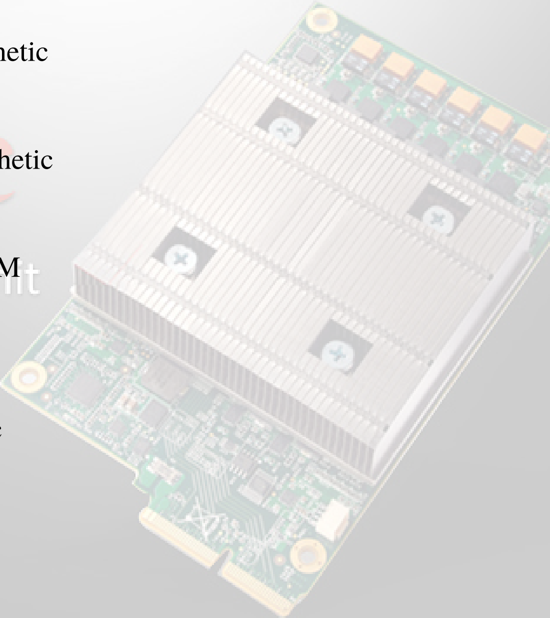
- A large fully-connected neural network (FCN-S) with around 55 million parameters is used to evaluate the performance of FCN
- The classical AlexNet (AlexNet-S) is used as an representative of CNN
- A smaller FCN (FCN-R) is constructed for MNIST data set
- An AlexNet (AlexNet-R) architecture is used for Cifar10 data set
- For RNNs, considering that the main computation complexity is related to the length of input sequence, 2 LSTM layers with input length of 32.

| Networks | | Input | Output | Layers | Parameters |
|---|---|---|---|---|---|
| FCN | FCN-S | 26752 | 26752 | 5 | ~55 millions |
| FCN | FCN-R | 784 | 10 | 5 | ~31 millions |
| CNN | AlexNet-S | 150528 | 1000 | 4 | ~61 millions |
| CNN | AlexNet-R | 3072 | 10 | 4 | ~81 thousands |
| RNN | LSTM | 10000 | 10000 | 2 | ~13 millions |

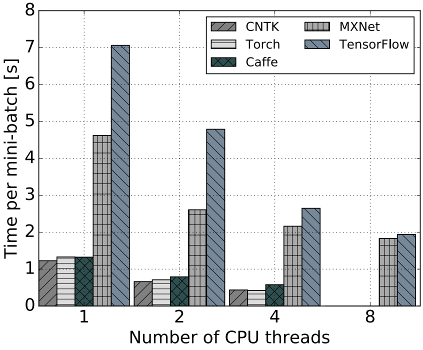
Figure: The experimental setup of neural networks for synthetic and real data

# §2. Comparison

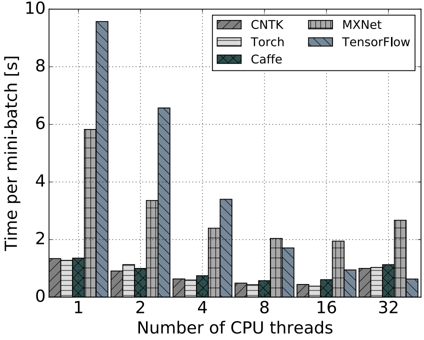




# CPU Scalability: FCN Synthetic




(a) Results on i7-3820.

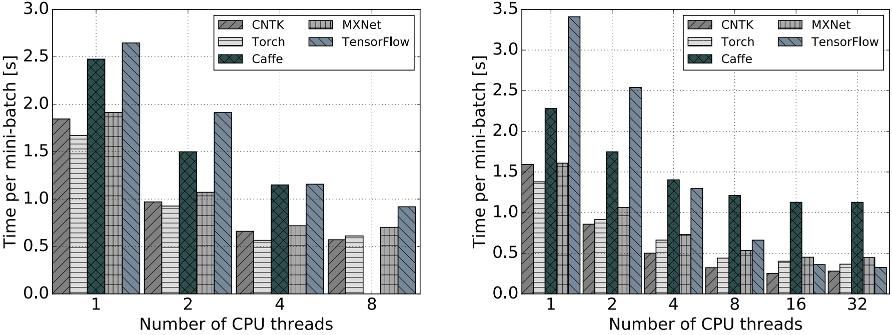



(b) Results on E5-2630.

Figure: FCN-S performance on CPU platform with a mini-batch size of 64

- CNTK/Torch/Caffe have similar CPU performance
- TensorFlow has excellent scalability

# CPU Scalability: FCN Real



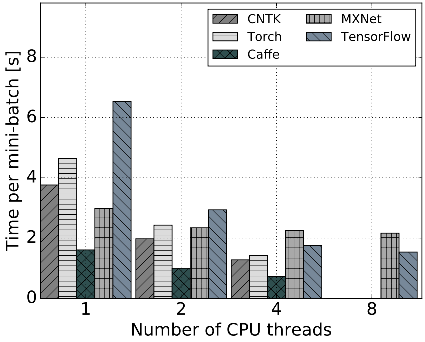

(a) Results on i7-3820.

(b) Results on E5-2630.

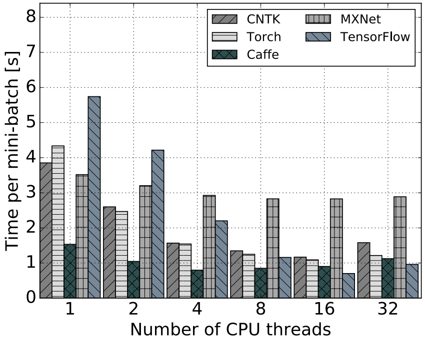
Figure: The FCN-R performance on CPU platform with a mini-batch size of 1024

- All engines have good CPU performance
- TensorFlow has good scalability but considerably slower

# CPU Scalability: CNN Synthetic




(a) Results on i7-3820.




(b) Results on E5-2630.

Figure: AlexNet-S performance on CPU platform with a mini-batch size of 16

- Caffe has best CNN performance as promised
- MXNet does not scale well for this test

# CPU Scalability: CNN Real

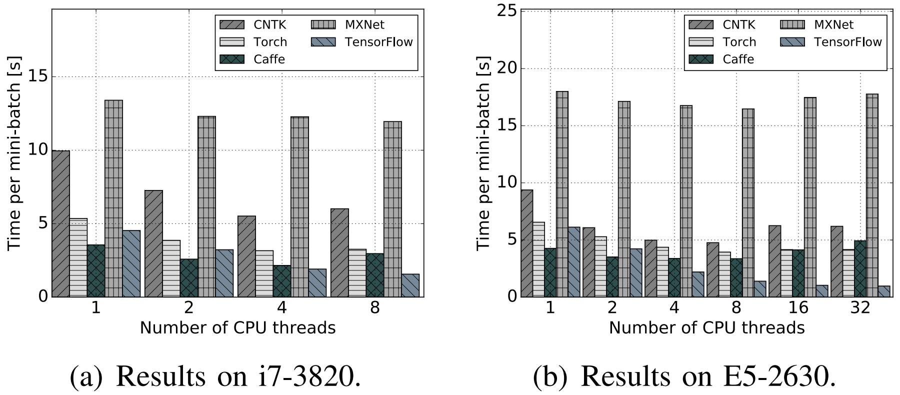
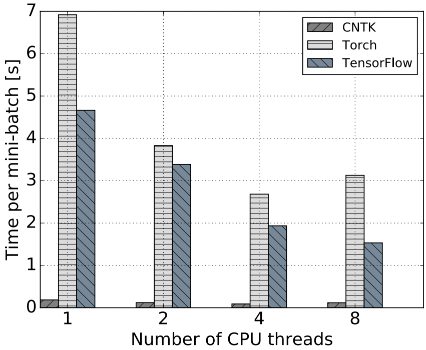


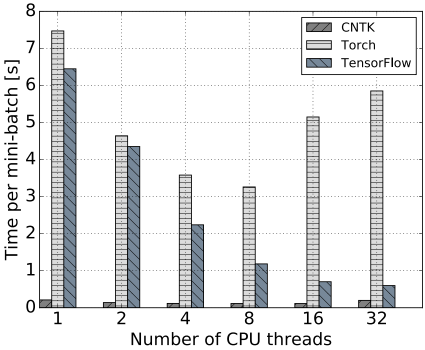

Figure: AlexNet-R performance on CPU platform with a mini-batch size of 1024

- Good scalability of TensorFlow kicks in
- Caffe does not scale well on multicore CPUs

# CPU Scalability: RNN LSTM




(a) Results on i7-3820.

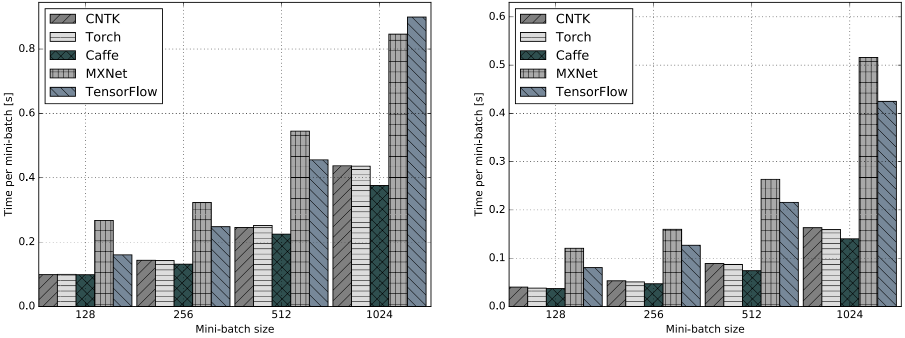



(b) Results on E5-2630.

Figure: LSTM performance on CPU platform with a mini-batch size of 256

- Pay more attention to CNTK in the future
- Caffe/MXNet does not support LSTM on CPUs

# GPU Speed: FCN Synthetic

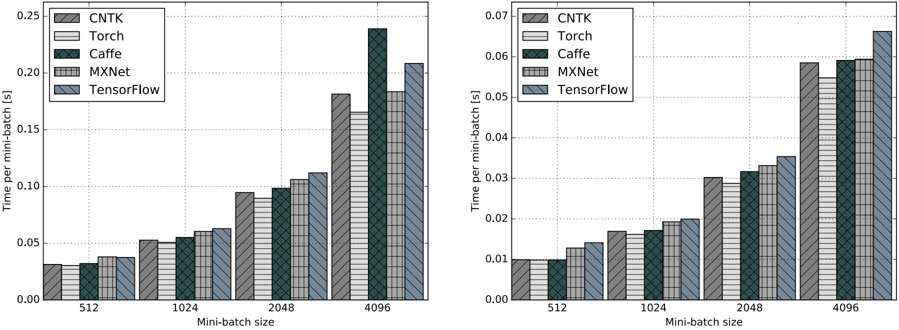



(a) Results on Tesla K80. (b) Results on GTX1080.

Figure: The performance comparison of FCN-S on GPU platforms

- CNTK/Torch/Caffe out-perform the others

# GPU Speed: FCN Real

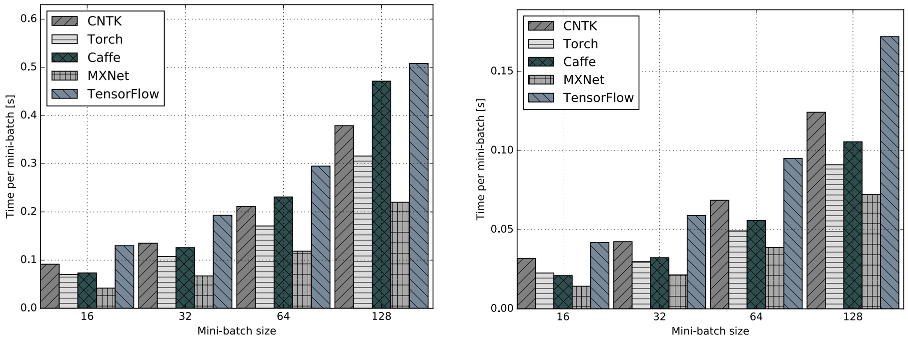



(a) Results on Tesla K80. (b) Results on GTX1080.

Figure: The performance comparison of FCN-R on GPU platforms

- All packages have similar performance

# GPU Speed: CNN Synthetic

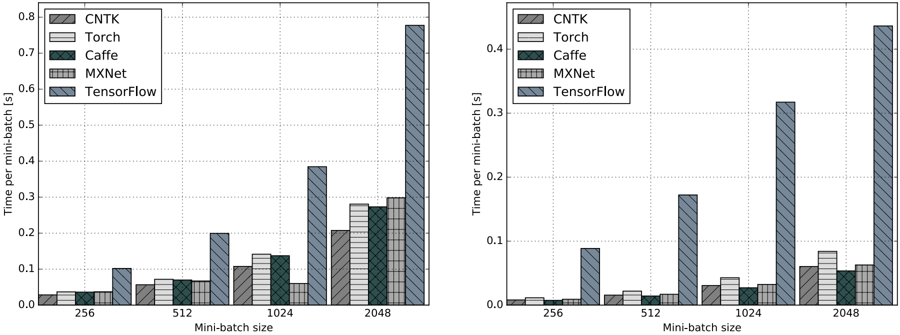



(a) Results on Tesla K80. (b) Results on GTX1080.

Figure: The performance comparison of AlexNet-S on GPU platforms

- MXNet out-perform the others for CNN on GPUs

# GPU Speed: CNN Real



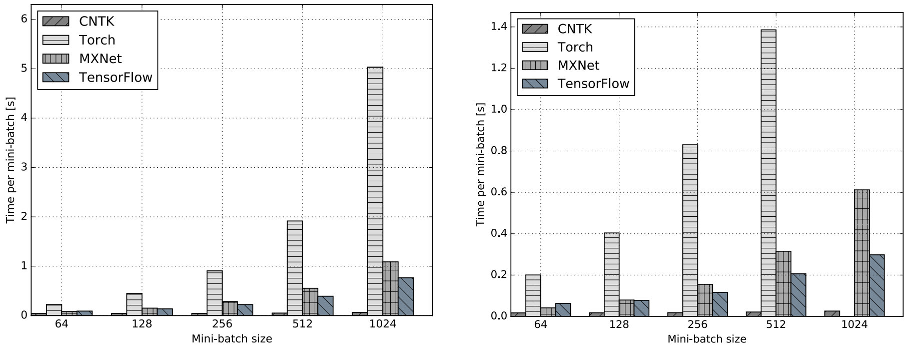

(a) Results on Tesla K80.

(b) Results on GTX1080.

Figure: The performance comparison of AlexNet-R on GPU platforms

- TensorFlow does not have good GPU performance in general

# GPU Speed: RNN LSTM

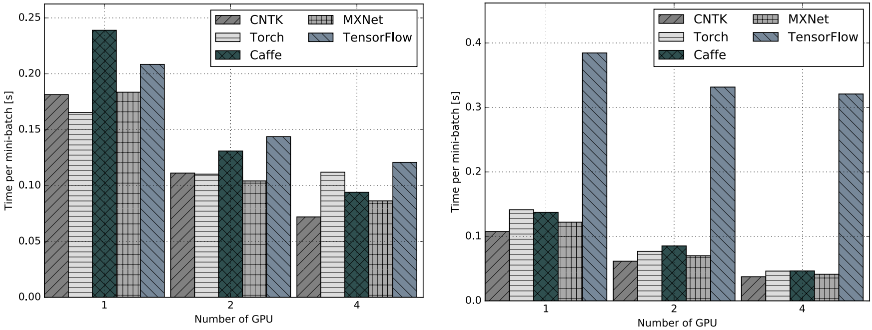



(a) Results on Tesla K80. (b) Results on GTX1080.

Figure: The performance comparison of LSTM on GPU platforms

- CNTK has excellent RNN performance both on CPU and GPU

# Multi-GPU Scalability




(a) FCN-R (b) CNN-R

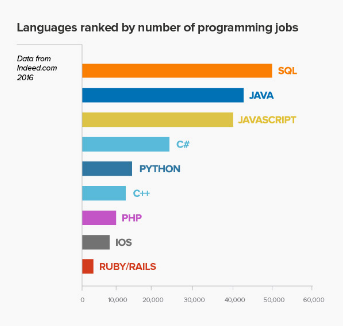
Figure: The scalability on a multi-GPU platform (2 × K80)

- Multi-GPU can greatly boost the training process of network
- MXNet shows overwhelming advantage over the others
- TensorFlow does not scale well on multi-GPU platform

# §3. Python

# Python 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

## 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?

- Code readability and maintenance — short code, fewer bugs
- Good for proof-of-concept prototyping
- Implementation time versus execution time
- 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 objects to know what they are, what they do and what they know. Python provides several utilities for code introspection, e.g. dir(), help(), type().

# Built-in Data Structures

## Numeric types: int, float, complex

```
a=1      # int
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], (0xf,))                # 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})
```

## User defined functions[2,3]

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

[2] Function overhead is high. Not to call a function repeatedly; Using aggregation instead.

[3] Python always passes (the value of) the reference to the object to the function.

# Control Flow

NCMIS

## 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

```
# loop from 0 to 9
for i in range(10):
    print i

# loop over the list named by oldlist
newlist = [s.upper() for s in oldlist]

a = range(5)              # create a new list a = [0,1,2,3,4]
b = a                     # b points to the list a
c = [item for item in a]  # copy list a to a new list
```

## While loop

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

# Modules

- This way will only introduce the module name into the name space in which the import command was issued. The names within the module will not appear in the enclosing namespace: they must be accessed through the module name.

```
import math
math.sin(3.14)
```

- This way does not introduce the name math into the current namespace. It does introduce all public names of the math module into the current namespace.

```
from math import *
sin(3.14)
```

- 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 use

```
from math import sin
sin(3.14)
```

- Make it as local as possible to avoid import overhead; But avoid calling it repeatedly; If possible, avoid it!

# Additional Comments

1. In Python, everything (including functions, modules, and files) are objects. A variable is created through assignment:

```
x = y = z = 0.1
```

2. help() is a function which gives information about the object. For example,

```
help('modules')        # generate a list of all modules that can be imported
help('modules time')   # generate a list of modules with 'time' in description
```

3. Use a profiler to find optimization possibilities

```
import profile         # cProfile is now recommended
profile.run('main()')
```

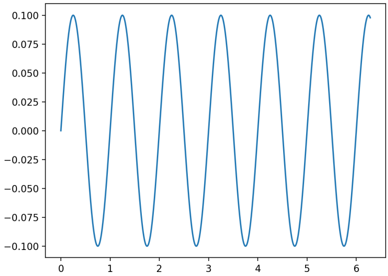
4. Some useful and important packages
   - NumPy: for scientific computing
   - Matplotlib: for visualising data
   - SciPy: for numerical algorithms
   - SymPy: for symbolic mathematics

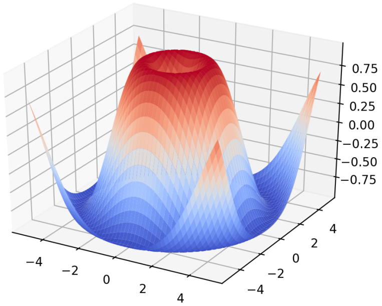
http://www.pythontutor.com

# Visualization

```python
# import pyplot module for visualization
import matplotlib.pyplot as plt
import numpy as np

# generate points (x,y)
dx = 0.001
x = np.arange(0, 2*np.pi, dx)
y = 0.1*np.sin(2*np.pi*x)

# plot (x,y) curve
plt.plot(x, y)
plt.show()
```

```python
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np

X = np.arange(-5, 5, 0.01)
Y = np.arange(-5, 5, 0.01)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

# Plot the surface
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z,
                       cmap=cm.coolwarm)
plt.show()
```



# §4. TensorFlow

# Basic Concepts

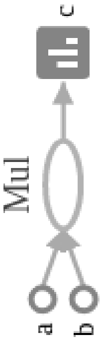
- Graph: In TensorFlow, ML algorithms are represented as computational graph. A computational graph (data flow graph) is a form of directed graph where vertices (nodes) describe operations, while edges represent data flowing between these operations.
  - Operation: An operation may represent a variable or constant, a control flow directive, a mathematical function, a file I/O, or 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.

# TensorFlow Operations

| Operation groups | Operations Examples |
| --- | --- |
| Maths | Add, Sub, Mul, Dic, Exp, Log, Greater than, Less than, Equal |
| Array | Concat, Slice, Constant, Rank, Shape |
| Matrix | MatMul, MatrixInverse, MatrixDeterminant |
| Neuronal Network | SoftMax, Sigmoid ReLU, Convolution2D, MaxPool |
| Checkpointing | Save, Restore |
| Queues and sincronizations | Enqueue, Dequeue, MutexAcquire, MutexRelease |
| Flow control | Merge, Switch, Enter, Leave, NextIteration |

# Computational Graph

```python
import tensorflow as tf

# Define a computational graph for TensorFlow
graph = tf.Graph()

# Define operation nodes in the graph
with graph.as_default():

    with tf.name_scope('input'):
        # Define two constant symbols (two nodes)
        a = tf.constant(1, name='a')
        b = tf.constant(2, name='b')

    with tf.name_scope('output'):
        # Define an operation node: c = a * b
        c = tf.multiply(a, b)
        tf.summary.histogram('c', c)

    # Merge all summaries and write to logdir
    logdir = '/tmp/tensorboard'
    merged = tf.summary.merge_all()
    writer = tf.summary.FileWriter(logdir, graph)

with tf.Session(graph=graph) as sess:
    # Compute and output the result (c=a*b)
    result = sess.run(c)
    print result
    # Write TensorBoard data to logdir
    writer.add_summary(merged.eval())

writer.close()
```



# Example: SoftMax in TF

```python
import tensorflow as tf

# Import training and test data from MNIST
import tensorflow.examples.tutorials.mnist.input_data as input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

graph = tf.Graph()

with graph.as_default():

    # Nodes can be grouped into visual blocks for TensorBoard
    with tf.name_scope('input_features'):
        x = tf.placeholder(tf.float32, shape=[None, 784], name='input_x')

    with tf.name_scope('input_labels'):
        y_ = tf.placeholder(tf.float32, shape=[None, 10], name='labels')

    with tf.name_scope('parameters'):
        W = tf.Variable(tf.zeros([784, 10]), name='weights')
        b = tf.Variable(tf.zeros([10]), name='biases')
        tf.summary.histogram('WEIGHTS', W)
        tf.summary.histogram('BIASES', b)

    with tf.name_scope('use_softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b)

    with tf.name_scope('train'):
        # Compute the cross entropy of real label y_ and prediction label y
        cross_entropy = -tf.reduce_sum(y_*tf.log(y))
        # Create a gradient-descent optimizer with learning rate = 0.01
        train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

# Example: SoftMax in TF (Cont)

```python
    with tf.name_scope('test'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
        # Track accuracy over time for TensorBoard
        tf.summary.scalar('Accuracy', accuracy)

    logpath = '/tmp/tensorboard'  # temporary path for storing TB summaries
    merged = tf.summary.merge_all()  # Merge all the summaries
    writer = tf.summary.FileWriter(logpath, graph)  # Write summaries

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

    for step in range(1,501):
        if (step%10) == 0:
            feed = {x: mnist.test.images, y_: mnist.test.labels}
            _, acc = sess.run([merged, accuracy], feed_dict=feed)
            print('Accuracy at %s step: %s' % (step, acc))
        else:
            batch_x, batch_y = mnist.train.next_batch(100)
            sess.run(train_step, feed_dict={x: batch_x, y_: batch_y})
            writer.add_summary(merged.eval(feed_dict={x: batch_x, y_: batch_y}),
                               global_step=step)

    writer.close()

print("Run the command line to start TensorBoard:\n" \
      "(TensorFlow) $ tensorboard --logdir=/tmp/tensorboard" \
      "\nThen open http://0.0.0.0:6006/ into your web browser")
```

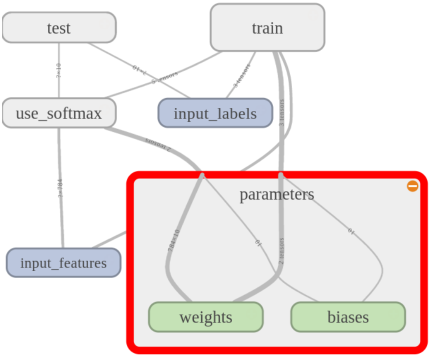
# Visualization: TensorBoard

Declarative style

- Same style as Teano
- Easy to understand
- Possible for graph optimization
- More difficult for debugging

Computation graphs are powerful but complicated

- Thousands of nodes or more
- Network is deep
- Visualization is helpful for debugging

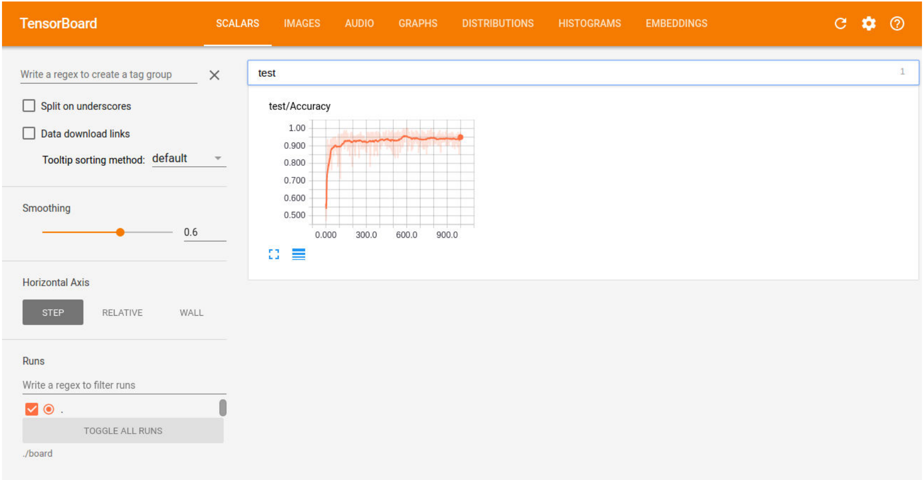



To run TensorBoard, use the following command:

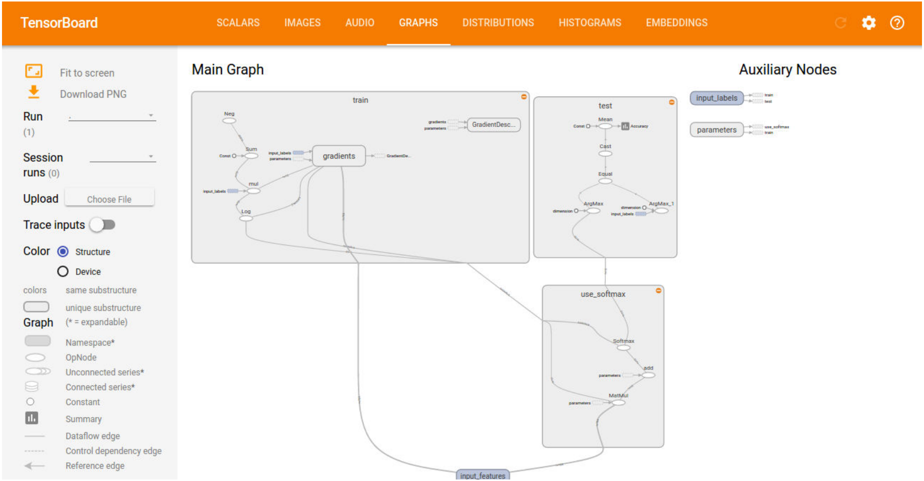
```
$ tensorboard --logdir=path/to/log-directory
```

Then navigate your web browser to localhost:6006 to view the TensorBoard.

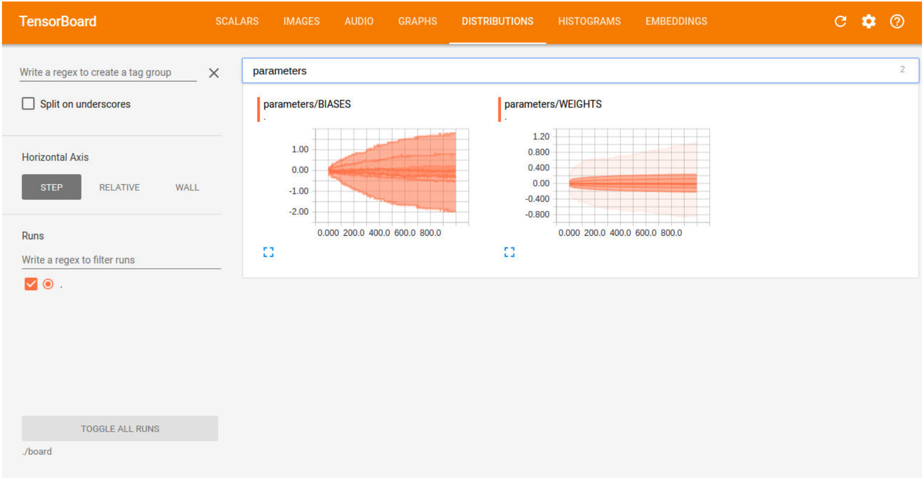
# TensorBoard: Scalars



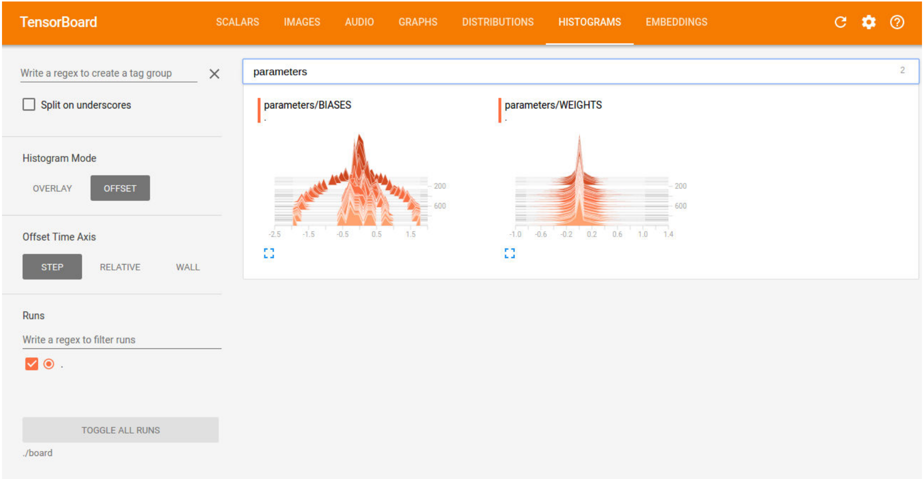
# TensorBoard: Graphs



# TensorBoard: Distributions



# TensorBoard: Histograms



# §5. MXNet

# General Comments

1. Used to power Amazon Web Services (AWS)
2. Support many different applications (e.g. computer vision, natural language processing, speech recognition, unsupervised machine learning, support embedded APIs, visualization)
3. Mixed programming style: imperative and declarative
   - Light-weighted (around 50K lines of core code)
   - Data parallelism with multi-devices: 88% scalability with 256 GPUs
   - Support many front-ends, including JavaScript (run on web browsers)
   - Provide intermediate-level and high-level interface modules
   - Provide abundant IO functions
4. Fully compatible with Torch: modules and operators
5. Visualize neural network graphs
   - Call mx.viz.plot_network( )
6. Not well documented, code not easy to read

# Example: SoftMax in MXNet

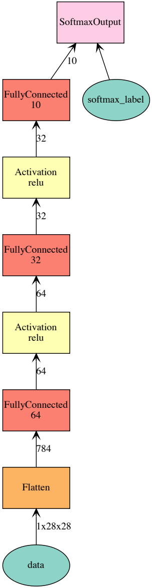
```python
import os, gzip, struct
import numpy as np
import mxnet as mx

# Read data from the MNIST dataset
def read_data(label_url, image_url):
    with gzip.open(label_url) as flbl:
        magic, num = struct.unpack(">II", flbl.read(8))
        label = np.fromstring(flbl.read(), dtype=np.int8)
    with gzip.open(image_url, 'rb') as fimg:
        magic, num, row, col = struct.unpack(">IIII", fimg.read(16))
        image = np.fromstring(fimg.read(),dtype=np.uint8).reshape(len(label),row,col)
    return (label, image)

train_lbl_file = 'train-labels-idx1-ubyte.gz'
train_img_file = 'train-images-idx3-ubyte.gz'
train_lbl, train_img = read_data(train_lbl_file, train_img_file)

test_lbl_file = 't10k-labels-idx1-ubyte.gz'
test_img_file = 't10k-images-idx3-ubyte.gz'
test_lbl,  test_img  = read_data(test_lbl_file, test_img_file)

# Create data iterators for MXNet
def to4d(img):
    return img.reshape(img.shape[0], 1, 28, 28).astype(np.float32)/255

batch_size = 100
train_iter = mx.io.NDArrayIter(to4d(train_img), train_lbl, batch_size, shuffle=True)
test_iter  = mx.io.NDArrayIter(to4d(test_img), test_lbl, batch_size)
```

# Example: SoftMax in MXNet (Cont)

```
# Define the network
data = mx.sym.Variable('data') # Create a place holder variable for the input data
data = mx.sym.Flatten(data=data)# Flatten the data from 4-D shape into 2-D
fc1  = mx.sym.FullyConnected(data=data, name='fc1', num_hidden=64)
act1 = mx.sym.Activation(data=fc1, name='relu1', act_type="relu")
fc2  = mx.sym.FullyConnected(data=act1, name='fc2', num_hidden = 32)
act2 = mx.sym.Activation(data=fc2, name='relu2', act_type="relu")
fc3  = mx.sym.FullyConnected(data=act2, name='fc3', num_hidden=10)
out  = mx.sym.SoftmaxOutput(data=fc3, name='softmax')
mod  = mx.mod.Module(out)

# Plot the network graph
mx.viz.plot_network(symbol=out, shape={'data': (batch_size, 1, 28, 28)}).view()

# Prepare output log infomation
import logging
logging.getLogger().setLevel(logging.INFO)

# Train the network
mod.fit(train_data=train_iter, eval_data=test_iter, num_epoch=25)
```

https://github.com/dmlc/mxnet/blob/master/example/svm_mnist/svm_mnist.py

# Visualization of the SoftMax Example




Thank You!

# §6. Torch

# Programming Interface

1. Very fast: Fastest scripting language LuaJIT is used
2. Flexible with wide range of applications
   - Speech, image, and video applications
   - Large-scale machine-learning applications
   - Used by Facebook, Twitter, Deepmind
   - Easy extensibility: integrate any library into Torch
3. Portable to any platform
   - Torch can run on iPhone with no modification to scripts
   - Embeddable, with ports to iOS, Android and FPGA backends
4. No automatic differentiation

# Example: Two-Layer Network

```python
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 Variables.
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)

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()
```

# Example: Two-Layer Network (Cont)

```
    # 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
    end
end
```

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

# Example: Linear Regression in Lua

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

# write the loss to a text file and read from there to plot it 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()

# 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
```

# Example: Linear Regression in Lua (Cont)

```
   # 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()

   # 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 10,000 times over our training data
for i = 1,1e4 do
   #this variable is used to estimate the average loss
   current_loss = 0
```

# Example: Linear Regression in Lua (Cont)

```lua
   #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)
      # 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
```

# §7. Caffe

# General Comments

1. Mainly focus on (and well suited for) CNN and image recognition
   - Use an expressive architecture
   - Provide simple command line tools for training and prediction
   - Allow defining models and optimization by configuration
   - Use Google protocol tool to define parameters for nets and solvers
2. Not so convenient to extend
   - Write layers in C++ or Python, handwritten CUDA code
   - Not well documented
   - No automatic differentiation
   - Not as flexible as other engines
   - Python interface not transparent
3. Lots of dependencies; can be tricky to install

# Example: 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: Extend Layers

```
import caffe
import numpy as np

class EuclideanLoss(caffe.layer):
    def setup(self, bottom, top):
        if len(bottom) != 2: #check number of input data
            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.

    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
```

# Example: Extend Layers (Cont)

## 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